UNIVERSIDADE DO PORTO
FACULDADE DE ENGENHARIA

Desenvolvimento de um Sistema de Gestão de AGVs

Dissertação de Mestrado

Rui Paulo Pinto da Rocha

Setembro de 1998



FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Desenvolvimento de um Sistema de Gestão de AGVs

Rui Paulo Pinto da Rocha

Licenciado em Engenharia Electrotécnica e de Computadores pela Faculdade de Engenharia da Universidade do Porto

Dissertação submetida para satisfação parcial dos requisitos do grau de mestre em

Engenharia Electrotécnica e de Computadores

(Área de especialização de Informática Industrial)

Dissertação realizada sob a orientação do

Professor Doutor Adriano Silva Carvalho

do Departamento de Engenharia Electrotécnica e de Computadores

da Faculdade de Engenharia da Universidade do Porto

Trabalho financiado pela Bolsa de Mestrado Praxis XXI / BM / 12963 / 97, atribuída pelo Ministério da Ciência e da Tecnologia, no âmbito do Subprograma Ciência e Tecnologia do 2º Quadro Comunitário de Apoio

Porto, Setembro de 1998

(81.31063)/ROC/DES
UNIVERSIDADE DO PORTO
Faculdade de Engenharia
BIBLIOTECA
N.º 50860
CDU
Data 04/07/2000
cat

Aos meus Pais

Resumo

Nos actuais sistemas flexíveis de fabrico, o transporte e manuseamento de materiais na planta fabril é frequentemente assegurado por um sistema de AGVs. O desempenho de um sistema de fabrico depende muito da optimização do fluxo de materiais no seio do processo produtivo. Por isso, o desempenho do sistema automático de transporte tem uma importância crucial na obtenção de um bom desempenho global do sistema de fabrico.

Neste documento, começa-se por enquadrar os sistemas de AGVs no objectivo de integração e flexibilização dos sistemas de fabrico. De seguida, define-se o que é um AGV e apresenta-se uma arquitectura para a gestão e controlo de um sistema de AGVs.

A concepção de um SAGVs envolve questões muito diversas, que este documento procura analisar sistematicamente, apresentando os problemas, indicando a sua influência no desempenho do SAGVs e comparando soluções conhecidas para a sua resolução. Os aspectos abordados são: a tecnologia do próprio AGV e do seu sistema de navegação, a concepção do *layout* dos AGVs e do plano de trajectórias, a escolha do número de veículos, a gestão de tráfego do SAGVs, os *deadlocks*, o problema de encaminhamento dos AGVs e o escalonamento ou afectação dos AGVs às tarefas de movimentação.

A avaliação do desempenho de um SAGVs é inseparável da avaliação do desempenho do sistema de fabrico global, onde se encontra inserido. Devido ao elevado número de graus de liberdade do sistema, à variedade de eventos que regem o seu comportamento e à natureza estocástica da ocorrência desses eventos, a avaliação do desempenho é quase sempre impossível pela via analítica. Neste quadro, as técnicas de simulação de sistemas dinâmicos de eventos discretos assumem particular importância, como ferramenta de especificação, concepção, análise e prototipagem de sistemas de fabrico.

As Redes de Petri são uma das ferramentas mais rigorosas e sistemáticas para modelizar e estudar sistemas dinâmicos de eventos discretos. Por essa razão, as Redes de Petri foram a base da experimentação realizada no trabalho apresentado e discutido neste documento. São apresentados os conceitos teóricos básicos das Redes de Petri autónomas, bem como as extensões não autónomas ao modelo básico, a saber: Redes de Petri sincronizadas, temporizadas, interpretadas, estocásticas, coloridas e orientadas a objectos. Ainda no contexto das Redes de Petri, é apresentada uma lista de ferramentas de *software* e respectivas características, que suportam a utilização das Redes de Petri como ferramenta de análise e de simulação.

Com base na arquitectura definida para o sistema de AGVs, é descrita a construção de um modelo, baseado em Redes de Petri coloridas e estocásticas, de um sistema de AGVs inserido num sistema de fabrico flexível, utilizando a ferramenta Artifex. O ambiente de simulação construído permitiu a realização de simulações do sistema de fabrico, suportado por um sistema de AGVs. Através da simulação do mesmo sistema em vários cenários de estratégias de escalonamento, procurou-se apurar a influência da política de escalonamento no desempenho global do sistema. Com base nos resultados experimentais obtidos nas simulações efectuadas, é feita a sua apresentação e discussão.

Finalmente, são indicadas as principais conclusões do trabalho desenvolvido.

Abstract

In the current flexible manufacturing systems (FMS), the material transportation and handling tasks on the shop floor is frequently assured by an automatic guided vehicle system (AGVS). The manufacturing system performance is very dependent on the material flow optimization in the heart of the production process. Thus, the AGVS performance is crucial in order to get a good global manufacturing system performance.

This document starts by putting in frame the automatic guided vehicle systems in the global objective of integrating and improving the flexibility of the manufacturing systems. The text proceeds with the presentation of technological aspects of an automatic guided vehicle (AGV) and with the definition of an architecture, for the management and control of an AGVS.

The design of an AGVS involves several theoretical subjects, which are studied in a systematic way in this document. It is pointed the influence of each design subject on the AGVS performance, and it is also presented known solutions for their resolution. The subjects discussed within this document are: AGVS layout design and trajectory planning, estimation of the number of vehicles required, traffic management, deadlocks, routing algorithms and scheduling algorithms for the assignment of tasks to AGVs.

The AGVS performance evaluation is inseparable from the performance evaluation of the global manufacturing system, where it is implanted. The high number of freedom degrees, the event diversity and the stochastic nature of the events occurrence in a manufacturing system, make the performance evaluation by the analytical way very difficult or almost impossible. In this context, the simulation techniques of dynamic discrete event systems are very important for the specification, design, analysis and prototyping of manufacturing systems.

Petri Nets are one of most rigorous and powerful modeling tools for event-driven systems. For this reason, Petri Nets are the experimentation basis of the research work presented and discussed within this document. It's presented some theoretical concepts about autonomous Petri Nets, as well non-autonomous extensions of the model: synchronized, timed, interpreted, stochastic, coloured and object-oriented Petri Nets. Still in the context of Petri Nets, it's presented a number of software tools and their essential characteristics, which make possible the Petri Nets application to the systems analysis and simulation.

Based in the AGVS architecture previously defined, it's described the construction of a coloured and stochastic Petri Nets model for an AGVS, implanted in a flexible manufacturing system, using the Artifex software package. The simulation environment built in the Artifex enabled the execution of simulations. The AGVS and flexible manufacturing system model was simulated with different scheduling strategies, in order to study the scheduling policy influence in the global manufacturing system performance. The main experimental results, given by the simulations, are presented and discussed.

Finally, it's indicated the main conclusions of the research work.

Résumé

Dans les présents systèmes flexibles de fabrique, le transport et maniement des matériaux dans la plante de fabrique est fréquemment assuré par un système d'AGVs (de la sigle anglaise "*Automatic Guided Vehicles*"). Le dégagement d'un système de fabrique est très dépendant de la optimisation du flux des matériaux dans l'intérieur du procès productif. Pour cela, le dégagement du système des AGVs (SAGVs) a une importance crucial en obtenir un bon dégagement global du système de fabrique.

En ce document, on se commence par encadrer les SAGVs dans l'objectif d'intégration et amélioration de la flexibilité des systèmes de fabrique. Ensuite, on se définit qu'est ce que c'est un AGV et se présente une architecture par la gérance et contrôle d'un SAGVs.

La conception d'un SAGVs enveloppe questions beaucoup divers, que ce document analyse systématiquement. On se présente les problèmes, s'indique sa influence dans le dégagement du SAGVs et se confronte solutions connues pour sa résolution. Les aspects abordés sont: la conception du *layout* d'AGVs et du plan des trajectoires, la choix du nombre des véhicules, la gérance de trafic du SAGVs, les *deadlocks*, le problème d'acheminement des AGVs et l'échelonnement ou affectation des AGVs à les tâches de mouvement.

L'appréciation de le dégagement d'un SAGVs est inséparable de l'appréciation de le dégagement du système de fabrique global, où il est inséré . Dû au élevé nombre des degrés de liberté du système, à la diversité d'événements que régent sa conduite et à la nature stochastique de l'occurrence de ces événements, l'évaluation de le dégagement est presque toujours impossible par la voie analytique. En ce carré, les techniques de simulation des systèmes dynamiques des événements discrets assument particulaire importance, comme outil de spécification, conception, analyse et implémentation des prototypes des systèmes de fabrique.

Les Réseaux de Petri sont une des outils plus rigoureux et systématiques pour modeler et étudier systèmes dynamiques des événements discrets. Pour cette raison, les Réseaux de Petri étaient la base d'expérimentation réalisé dans le travail présenté et discuté en ce document. On se présente les concepts théoriques basiques de les Réseaux de Petri autonomes, bien comme les extensions au modèle basique non autonomes, nommément: Réseaux de Petri synchronisés, temporisés, interprétés, stochastiques, colorés et à objets. Encore dans le contexte de les Réseaux de Petri, on se présente une liste d'outils de *software*, bien comme ses caractéristiques, que supportent l'utilisation de les Réseaux de Petri, comme outils d'analyse et de simulation.

L'architecture définit pour le SAGVs a été utilisé pour la construction d'un modèle basé en Réseaux de Petri colorés et stochastiques d'un SAGVs, insérait dans un système de fabrique flexible, avec l'utilisation de l'outil Artifex. L'ambiant de simulation construit a été utilisé pour la réalisation des simulations du même système, avec divers stratégies de échelonnement. L'objectif de les simulations a été étudier l'influence de la politique d'échelonnement dans le dégagement global du système. Les principaux résultats expérimentales obtenu avec les simulations sont présentés et discutés.

Finalement, on se indique les principaux conclusions du travail développé.

Prefácio

Uma parte significativa dos progressos tecnológicos na área dos sistemas de fabrico tem-se centrado em níveis próximos ou muito próximos do *hardware*, permitindo a concepção de máquinas com menores tempos de maquinação e menores tempos de *setup*. Em face dessa evolução tecnológica, o planeamento da produção começou a dar menos importância ao tempo de execução das operações de produção, passando a valorizar mais a eficiência com que os materiais são movimentados entre estações de trabalho e entre estas e os armazéns.

Habitualmente os sistemas de fabrico flexíveis são suportados por um sistema automático de manuseamento e de transporte de materiais baseado em AGVs. Por isso, os sistemas de AGVs merecem ser estudados com atenção por aqueles que concebem e gerem sistemas de fabrico flexíveis, pois a optimização do fluxo de materiais no interior da fábrica é bastante condicionado pela qualidade do serviço prestado pelo sistema de transporte automático.

Este trabalho procura apontar as principais dificuldades que se levantam na concepção de um sistema de AGVs, apresentando e discutindo as várias soluções existentes para a sua resolução. Para além disso é apresentada uma arquitectura de um sistema de AGVs e é descrito um modelo de um sistema de AGVs baseado em Redes de Petri estocásticas e coloridas, que serviu de base à simulação de um sistema de AGVs inserido num sistema de fabrico.

Para a realização deste trabalho contei com a colaboração de pessoas e de entidades cujo auxílio foi precioso e inestimável e às quais quero prestar o meu agradecimento.

Quero manifestar em primeiro lugar o meu grande e sincero reconhecimento ao Sr. Prof. Doutor Adriano Silva Carvalho, meu orientador científico e responsável pelo Grupo de Automação do I.S.R. – Instituto de Sistemas e Robótica, Pólo do Porto, que desde sempre me tem privilegiado com o seu valioso apoio e confiança, enriquecendo o meu trabalho com os seus ensinamentos e estímulos e proporcionando as melhores condições para a sua realização.

Gostaria também de agradecer a camaradagem, os comentários construtivos e o incentivo dos meus colegas do I.S.R., em especial aos Engenheiros Armando Sousa, Carlos Martins, Paulo Costa, Joaquim Alves e Carlos João Ramos.

Agradeço também o apoio financeiro prestado pelo Programa Praxis XXI do Ministério da Ciência e da Tecnologia, relativo à Formação Avançada de Recursos Humanos, no âmbito do Subprograma Ciência e Tecnologia do 2º Quadro Comunitário de Apoio, por me ter concedido uma Bolsa de Mestrado durante um ano, para a realização desta Dissertação de Mestrado.

Quero prestar uma homenagem muito especial aos meus queridos pais que sempre me acarinharam e me incentivaram a lutar pela realização dos meus sonhos, e a quem devo tudo aquilo que consegui até hoje. Quero também agradecer à minha querida irmã o afecto e a força que sempre me deu naqueles momentos mais difíceis, em que necessitei de um ombro amigo. Finalmente, quero agradecer o incentivo, o amor e a paciência da minha querida Nocas, que me ensinou que a coisa mais fascinante da vida é precisamente viver a vida.

Índice

RESUMO V

ABSTRACT VII

RÉSUMÉ IX

PREFÁCIO XI

ÍNDICE XIII

ÍNDICE DE FIGURAS XXV

LISTA DE TABELAS XXIX

LISTA DE ACRÓNIMOS XXXI

GLOSSÁRIO XXXIII

1. INTRODUÇÃO 1
1.1 O PAPEL DOS SISTEMA DE AGVS NA INTEGRAÇÃO DOS SISTEMAS DE FABRICO FLEXÍVEIS 1
1.1.1 Evolução do conceito de automatização industrial em direcção à integração empresarial 1
1.1.2 A importância dos SAGVs no actual panorama de integração de sistemas de fabrico 3
1.1.3 Dificuldades inerentes à concepção de SAGVs 4
1.2 APRESENTAÇÃO DOS OBJECTIVOS E DA METODOLOGIA SEGUIDA NA REALIZAÇÃO DO TRABALHO 4
1.3 ROTEIRO DA DISSERTAÇÃO 5

2. FUNCIONALIDADES DE UM SAGVS E AVALIAÇÃO DO SEU DESEMPENHO 7
2.1 O AGV COMO DISPOSITIVO DE AUTOMAÇÃO 7
2.2 ARQUITECTURA DE UM SAGVS EM 3 CAMADAS 10
2.2.1 Especificação do SPAGV 11
2.2.2 Especificação do SGAGV 12
2.2.3 Especificação do SGM 13
2.3 MEDIDAS QUANTITATIVAS DO DESEMPENHO DE UM SAGVS 13

3. CONCEPÇÃO DE UM SAGVS 17
3.1 LAYOUT DAS TRAJECTÓRIAS 17
3.1.1 Modelização de um layout de trajectórias 18
3.2 NÚMERO MÍNIMO DE VEÍCULOS NECESSÁRIOS 20
3.3 GESTÃO DE TRÁFEGO 23
3.3.1 Possível modelo de operação para o movimento de um AGV 23
3.3.2 Regras de controlo utilizadas na gestão de tráfego 24
3.3.3 Informação relativa ao layout utilizada na gestão de tráfego 24
3.3.4 Deadlocks 29
3.3.4.1 Definição e identificação de um deadlock 29
3.3.4.2 Apresentação e comparação das estratégias existentes para o tratamento dos deadlocks 34
3.4 ENCAMINHAMENTO 35
3.4.1 Definição do problema 35
3.4.2 Soluções existentes para o problema de encaminhamento 36
3.4.2.1 Algoritmo do caminho mais curto 38
3.5 ESCALONAMENTO 39
3.5.1 Definição do problema 40
3.5.2 Fases da execução de uma tarefa de transporte por um SAGVs 41
3.5.3 Objectivo de um escalonador e avaliação do seu desempenho 41
3.5.4 Descrição de regras de despacho habitualmente utilizadas nos SAGVs 41
3.5.4.1 Regras para o despacho desencadeado por um centro de trabalho 42
3.5.4.1.1 Regra do veículo mais próximo (VMP) 42
3.5.4.1.2 Regra do veículo há mais tempo disponível 43

3.5.4.1.3 Regra do veículo menos utilizado (VMU) 43
3.5.4.2 Regras para o despacho desencadeado por um veículo 43
3.5.4.2.1 Regra do tempo mais curto (ou da distância mais curta) de viagem (TMCV) 43
3.5.4.2.2 Regra do máximo tamanho da fila de saída (MTFS) 44
3.5.4.2.3 Regra do mínimo espaço livre na fila de saída (MELFS) 44
3.5.4.2.4 Regra do primeiro a chegar primeiro a ser servido modificada (PCPSM) 45
3.5.5 Formulação matemática do problema de afectação de AGVs a tarefas 45
3.5.5.1 O algoritmo húngaro 46
3.5.5.2 Reafectação dinâmica 47

4. TEORIA DAS REDES DE PETRI AUTÓNOMAS 49
4.1 HISTÓRIA DO APARECIMENTO DAS RP 49
4.2 CONCEITOS BÁSICOS SOBRE RP 50
4.2.1 Definição de uma RP 50
4.2.2 Elementos constituintes de uma RP 50
4.2.3 Marcação de uma RP 51
4.2.4 Mecanismo de disparo de uma transição 52
4.3 RP AUTÓNOMAS E NÃO AUTÓNOMAS 53
4.4 ESTRUTURAS PARTICULARES DE RP 54
4.4.1 Grafo de estados 54
4.4.2 Grafo de eventos 56
4.4.3 RP livre de conflitos 56
4.4.4 RP de escolha livre 56
4.4.5 RP simples 56
4.4.6 RP puras 57
4.5 PROPRIEDADES DE UMA RP 57
4.5.1 RP limitada e RP segura 60
4.5.2 RP viva 61
4.5.3 RP livre de deadlocks 62
4.5.4 RP com conflitos efectivos e conflitos efectivos persistentes 63
4.5.5 Invariantes de uma RP 63
4.6 TÉCNICAS DE ANÁLISE DAS PROPRIEDADES DE UMA RP 64
4.6.1 Grafo de marcações e árvore de cobertura 64
4.6.2 Métodos baseados em álgebra linear 65
4.6.3 Métodos de redução 65
4.7 COMPORTAMENTOS DOS SISTEMAS DISCRETOS FACILMENTE IDENTIFICÁVEIS ATRAVÉS DE RP 66
4.7.1 Sincronização 66
4.7.2 Partilha de recursos ou exclusão mútua 66
4.7.3 Memorização 67
4.7.4 Leitura 67
4.7.5 Escolha 68
4.7.6 Sequência 68
4.7.7 Paralelismo e concorrência 69
4.7.8 Circuito ou auto-ciclo 69

5. REDES DE PETRI NÃO AUTÓNOMAS 71
5.1 RP SINCRONIZADAS 71
5.1.1 Relação entre as propriedades de uma RP sincronizada e da RP autónoma equivalente sem sincronização 72
5.2 RP TEMPORIZADAS 73
5.2.1 RP P-temporizadas 73
5.2.1.1 Princípio de funcionamento de uma RP P-temporizada 73
5.2.1.2 Evolução da marcação de uma RP P-temporizada 74
5.2.2 RP T-temporizadas 74
5.2.2.1 Princípio de funcionamento de uma RP T-temporizada 75
5.2.2.2 Evolução da marcação de uma RP T-temporizada 76
5.2.3 Equivalência entre RP P-temporizadas e T-temporizadas 76
5.3 RP INTERPRETADAS 77
5.4 RP ESTOCÁSTICAS 78
5.5 RP COLORIDAS 79

5.5.1 Apresentação intuitiva das RP coloridas 80
5.5.1.1 O conceito de cor 81
5.5.1.2 O conceito de função de um arco 82
5.5.2 Extensões das RP coloridas 83
5.6 RP ORIENTADAS A OBJECTOS 83

6. FERRAMENTAS DE SUPORTE À MODELIZAÇÃO BASEADA EM REDES DE PETRI 85
6.1 BREVE DESCRIÇÃO DE ALGUNS PACOTES DE SOFTWARE ACTUALMENTE DISPONÍVEIS 89
6.1.1 ALPHA/Sim 89
6.1.2 Artifex 89
6.1.3 CodeSign 89
6.1.4 Design/CPN 90
6.1.5 ELSIR 91
6.1.6 ExSpect 92
6.1.7 F-net 92
6.1.8 Looping 92
6.1.9 Netmate 92
6.1.10 PEP 93
6.1.11 PETRI Maker 93
6.1.12 PNtalk 93
6.1.13 POSES++ 93
6.1.14 SYROCO 94
6.1.15 THORN/DE 94
6.1.16 Visual SimNet 94
6.2 O SOFTWARE ARTIFEX 95
6.2.1 Principais caracteristicas da ferramenta 95
6.2.2 Componentes mais importantes do Artifex 96
6.2.2.1 Artifex.Model 96
6.2.2.1.1 Possibilidade de definição de objectos 98
6.2.2.1.2 Introdução do factor tempo nos modelos 99
6.2.2.2 Artifex.Validate 100
6.2.2.3 Artifex.Measure 100

7. ESPECIFICAÇÃO DE UM MODELO DE UM SAGVS 103
7.1 PRINCIPAIS ENTIDADES DE UM SAGVS 103
7.1.1 Estados possíveis para um AGV 104
7.1.2 Estados possíveis para uma tarefa e tipos de tarefas possíveis 105
7.1.3 O layout 106
7.2 ESPECIFICAÇÃO DOS BLOCOS CONSTITUINTES DO SAGVS 106
7.2.1 Mensagens trocadas entre os blocos do SAGVs 106
7.2.1.1 Mensagens trocadas entre o SGM e o SGAGV 107
7.2.1.2 Mensagens trocadas entre o SGAGV e o SPAGV 109
7.2.2 Especificação do SGM 111
7.2.2.1 Informação sobre um AGV a guardar no SGM 111
7.2.2.2 Informação sobre uma tarefa a guardar no SGM 112
7.2.2.3 Informação sobre o layout a guardar no SGM 112
7.2.3 Especificação do SGAGV 112
7.2.3.1 Informação sobre um AGV a guardar no SGAGV 113
7.2.3.2 Informação sobre uma tarefa a guardar no SGAGV 114
7.2.3.3 Informação sobre o layout a guardar no SGAGV 114
7.2.4 Especificação do SPAGV 115
7.2.4.1 Informação sobre um AGV a guardar no SPAGV 116
7.2.4.2 Informação sobre o layout a guardar no SPAGV 116

8. RP ESTOCÁSTICA E COLORIDA DAS 3 CAMADAS DE GESTÃO DE UM SAGVS 117
8.1 IMPLEMENTAÇÃO DE UM TEMPORIZADOR FIÁVEL 118
8.2 CLASSE SGM 119
8.2.1 Sub-rede 'leitura_fich_nos_tar' do SGM 120
8.2.2 Sub-rede 'gere_AGVs' do SGM 122
8.2.2.1 Sub-rede 'gera_AGVs_DISABLED' do SGM 122
8.2.2.2 Sub-rede 'gere_AGVs_DISABLED' do SGM 123

8.2.2.3 Sub-rede 'gere_AGVs_ENABLED' do SGM 124
8.2.2.4 Sub-rede 'gere_AGVs_ACTIVOS' do SGM 125
8.2.3 Sub-rede 'gere_Tarefas' do SGM 126
8.2.3.1 Sub-rede 'lanca_Tarefas' do SGM 127
8.2.3.2 Sub-rede 'gere_cancelamento_Tarefas' do SGM 128
8.2.3.3 Sub-rede 'gere_Tarefas_FILA' do SGM 129
8.2.3.4 Sub-rede 'gere_Tarefas_PASSO1' do SGM 130
8.2.3.5 Sub-rede 'gere_Tarefas_PASSO2' do SGM 132
8.2.3.6 Sub-rede 'gere_Tarefas_PASSO3' do SGM 132
8.2.3.7 Sub-rede 'gere_Tarefas_PASSO4' 133
8.2.3.8 Sub-rede 'proc_tar_canc_FILA_ou_PASSO1' do SGM 133
8.3 CLASSE SGAGV 134
8.3.1 Sub-rede 'leitura_fich_config' do SGAGV 135
8.3.2 Sub-rede 'cria_lista_AGVs' do SGAGV 137
8.3.3 Sub-rede 'proc_mens_receb_SGM' do SGAGV 138
8.3.3.1 Sub-rede 'proc_mens_relativas_a_AGVs' do SGAGV 138
8.3.3.2 Sub-rede 'proc_mens_relativas_a_Tarefas' do SGAGV 141
8.3.3.2.1 Sub-rede 'proc_novas_tarefas' do SGAGV 141
8.3.3.2.2 Sub-rede 'proc_cancel_tarefas' 142
8.3.4 Sub-rede 'cancela_movim_avan_AGVs' do SGAGV 143
8.3.5 Sub-rede 'escalona_e_despacha_tar_normais' do SGAGV 144
8.3.6 Sub-rede 'gere_AGVs' do SGAGV 145
8.3.6.1 Sub-rede 'introducao_AGVS' do SGAGV 147
8.3.6.2 Sub-rede 'transicao_PARADO_BLOQUEADO' do SGAGV 147
8.3.6.3 Sub-rede 'proc_erros_AGVs' do SGAGV 148
8.3.6.4 Sub-rede 'passagem_modo_manual_AGVs' do SGAGV 148
8.3.6.5 Sub-rede 'gere_AGVs_est_BLOQUEADO' do SGAGV 149
8.3.6.6 Sub-rede 'gere_AGVs_MOVIMENTO' do SGAGV 151
8.3.6.7 Sub-rede 'resolve_conflitos_AGVs_PARADOS' do SGAGV 152
8.3.6.8 Sub-rede 'resolve_Deadlock_Circular' do SGAGV 153
8.3.6.9 Sub-rede 'resolve_conflitos_AGVs_ERRO' do SGAGV 153
8.3.7 Sub-rede 'proc_tarefas_AGVs_desactivados' do SGAGV 154
8.4 CLASSE SPAGV 155
8.4.1 Sub-rede 'leitura_fich_config' do SPAGV 157
8.4.2 Sub-rede 'gera_AGVs_DISABLED' do SPAGV 158
8.4.3 Sub-rede 'cancela_movim_avanco' do SPAGV 159
8.4.4 Sub-rede 'processa_AGVs_desactivados' do SPAGV 160
8.4.5 Sub-rede 'gere_AGVs_DISABLED' do SPAGV 161
8.4.6 Sub-rede 'gere_AGVs_ENABLED' do SPAGV 162
8.4.7 Sub-rede 'gere_AGVs_PARADOS' do SPAGV 163
8.4.8 Sub-rede 'gere_AGVs_MOVIMENTO' do SPAGV 165
8.4.9 Sub-rede 'gere_AGVs_ERRO' do SPAGV 167

9. RP ESTOCÁSTICA E COLORIDA DE UM SISTEMA DE FABRICO DOTADO DE UM SAGVS ... 169
9.1 DESCRIÇÃO DA PLANTA FABRIL MODELIZADA 169
9.2 MODELIZAÇÃO DAS ENTIDADES DO SISTEMA DE FABRICO 171
9.2.1 Classe CENTRO_RECEPCAO 171
9.2.1.1 Sub-rede 'inicializa_modelo' do CENTRO_RECEPCAO 174
9.2.2 Classe ARMAZEM_AUTOMATICO 175
9.2.2.1 Sub-rede 'inicializa_modelo' da classe ARMAZEM_AUTOMATICO 178
9.2.2.2 Sub-rede 'entrada_paletes' da classe ARMAZEM_AUTOMATICO 179
9.2.2.3 Sub-rede 'saida_paletes' da classe ARMAZEM_AUTOMATICO 180
9.2.3 Classe POSTO_TRABALHO 182
9.2.3.1 Sub-rede 'inicializa_modelo' da classe POSTO_TRABALHO 184
9.2.3.2 Sub-rede 'entrada_paletes' da classe POSTO_TRABALHO 185
9.2.3.3 Sub-rede 'processamento_paletes' da classe POSTO_TRABALHO 187
9.2.3.4 Sub-rede 'saida_paletes' da classe POSTO_TRABALHO 188
9.2.4 Classe CENTRO_EXPEDICAO 188
9.2.4.1 Sub-rede 'inicializa_modelo' da classe CENTRO_EXPEDICAO 191
9.3 MODELIZAÇÃO DO SISTEMA DE FABRICO (CLASSE SISTEMA_MANUFACTURA) 192

9.3.1 Descrição da sub-rede 'Controlo_CRMP' da classe SISTEMA_MANUFACTURA 195
9.3.2 Figuras com as restantes sub-redes da classe SISTEMA_MANUFACTURA 197
9.3.2.1 Sub-rede 'Controlo_AAMP' da classe SISTEMA_MANUFACTURA 198
9.3.2.2 Sub-rede 'Controlo_CM1' da classe SISTEMA_MANUFACTURA 200
9.3.2.3 Sub-rede 'Controlo_CM2' da classe SISTEMA_MANUFACTURA 202
9.3.2.4 Sub-rede 'Controlo_PM1' da classe SISTEMA_MANUFACTURA 204
9.3.2.5 Sub-rede 'Controlo_PM2' da classe SISTEMA_MANUFACTURA 206
9.3.2.6 Sub-rede 'Controlo_PM3' da classe SISTEMA_MANUFACTURA 208
9.3.2.7 Sub-rede 'Controlo_PM4' da classe SISTEMA_MANUFACTURA 210
9.3.2.8 Sub-rede 'Controlo_PM5' da classe SISTEMA_MANUFACTURA 212
9.3.2.9 Sub-rede 'Controlo_PM6' da classe SISTEMA_MANUFACTURA 214
9.3.2.10 Sub-rede 'Controlo_CM3' da classe SISTEMA_MANUFACTURA 216
9.3.2.11 Sub-rede 'Controlo_CM4' da classe SISTEMA_MANUFACTURA 218
9.3.2.12 Sub-rede 'Controlo_CM' da classe SISTEMA_MANUFACTURA 220
9.3.2.13 Sub-rede 'Controlo_CQ' da classe SISTEMA_MANUFACTURA 222
9.3.2.14 Sub-rede 'Controlo_AAPA' da classe SISTEMA_MANUFACTURA 224
9.4 CONSTRUÇÃO DO MODELO GLOBAL DO SAGVS 225

10. APRESENTAÇÃO E ANÁLISE DE RESULTADOS DAS SIMULAÇÕES 227
10.1 CONFIGURAÇÃO DO SISTEMA DE FABRICO UTILIZADO NAS SIMULAÇÕES 227
10.2 METODOLOGIA UTILIZADA NA REALIZAÇÃO DAS SIMULAÇÕES 228
10.3 APRESENTAÇÃO E DISCUSSÃO DOS RESULTADOS OBTIDOS NAS SIMULAÇÕES 232
10.3.1 Estatísticas referentes ao controlo do SAGVs 233
10.3.2 Estatísticas referentes aos tempos de execução de uma tarefa 239
10.3.3 Estatísticas relacionadas com os AGVs 243
10.3.4 Estatísticas relacionadas com as células da planta fabril 249

11. CONCLUSÕES 255
11.1 DESENVOLVIMENTOS FUTUROS 258

LISTA DE REFERÊNCIAS 261

ANEXOS 265

ANEXO A – CÓDIGO C INCLUÍDO NAS CLASSES DO MODELO DO SAGVS 267
A.1 FICHEIROS DE CABEÇALHO (HEADER FILES) 267
A.1.1 ulint.h 267
A.1.2 lefich.h 267
A.1.3 listas.h 267
A.1.4 rand_fun.h 270
A.1.5 layout.h 270
A.1.6 mens_log.h 271
A.1.7 sgm_sga.h 272
A.1.8 sga_spa.h 272
A.1.9 sistagvs.h 272
A.1.10 sgm.h 276
A.1.11 sgagv.h 278
A.1.12 spagv.h 285
A.2 FICHEIROS FONTE (SOURCE FILES) 287
A.2.1 fun_aux.c 287
A.2.2 listas.c 287
A.2.3 rand_fun.c 303
A.2.4 layout.c 305
A.2.5 mens_log.c 321
A.2.6 sgm_sga.c 322
A.2.7 sga_spa.c 325
A.2.8 sistagvs.c 327
A.2.9 sgm.c 329
A.2.10 dettraj.c 341
A.2.11 esc_desp.c 350
A.2.12 sgm.c 380

A.2.13 sgagv.c 392
A.2.14 spagv.c 425

ANEXO B – COMMUNICATION UNITS 435
B.1 ULINT 435
B.1.1 Ficheiros de cabeçalho incluídos 435
B.1.2 Definição 435
B.2 BOOLEANO 435
B.3 MENS_SGM_SGAGV 435
B.3.1 Ficheiros de cabeçalho incluídos 435
B.3.2 Definição 435
B.4 MENS_SGAGV_SPAGV 435
B.4.1 Ficheiros de cabeçalho incluídos 435
B.4.2 Definição 435
B.5 TEMPORIZACAO 436
B.5.1 Ficheiros de cabeçalho incluídos 436
B.5.2 Definição 436
B.5.3 Descrição dos campos 436
B.6 ORDEM 436
B.7 CAPAC_FILA 436
B.7.1 Definição 436
B.7.2 Descrição dos campos 436
B.8 PALETE 436
B.8.1 Definição 436
B.8.2 Descrição dos campos mais importantes 436

ANEXO C – CLASSES ARTIFEX DAS 3 CAMADAS DE GESTÃO DO SAGVS 439
C.1 CLASSE TEMPORIZADOR 439
C.1.1 CMN units incluídas 439
C.1.2 Posições de entrada 439
C.1.3 Posições de saída 439
C.1.4 Ficheiros de cabeçalho incluídos 439
C.1.5 Funções C definidas internamente 439
C.1.4 Acções iniciais 439
C.1.5 Acções finais 440
C.1.6 Tipos de tokens 440
C.1.6.1 AP_NO_L_UID 440
C.1.7 Posições 440
C.1.7.1 TEMPOR_ACTIVA:AP_NO_L_UID 440
C.1.8 Transições 440
C.1.8.1 INICIA_TEMPORIZACAO 440
C.1.8.2 INTERR_TEMPOR_AGORA_INICIADA 441
C.1.8.3 INTERR_TEMPOR_A_DECORRER 441
C.1.8.4 TEMPORIZA 441
C.2 CLASSE SGM 442
C.2.1 CMN units incluídas 442
C.2.2 Posições de entrada 442
C.2.3 Posições de saída 442
C.2.4 Parâmetros 442
C.2.5 Ficheiros de cabeçalho incluídos 443
C.2.6 Funções definidas internamente 443
C.2.7 Acções iniciais 444
C.2.8 Acções finais 445
C.2.9 Tipos de tokens 445
C.2.9.1 AP_NOS_TAREFAS 445
C.2.9.2 AP_AGV 445
C.2.9.3 AP_TAREFA 445
C.2.9.4 DOUBLE 445
C.2.10 Variáveis locais 445
C.2.11 Posições partilhadas 446

C.2.12 Entidades da RP da página principal da classe SGM 446
C.2.12.1 Objectos 446
C.2.12.2 Acção da transição 'LIMPA_FIM_TEMP_MUD_EST_ALOC' 447
C.2.13 Entidades da RP da página 'gera_AGVs_DISABLED' 447
C.2.13.1 Posições 447
C.2.13.2 Transições 447
C.2.14 Transições da RP da página 'gere_AGVs_ACTIVOS' 448
C.2.14.1 DISABLE_AGV_ACTIVO 448
C.2.14.2 MUDA_EST_ALOC_AGV_ACT 448
C.2.14.3 REC_MUD_EST_AGV 449
C.2.15 Entidades da RP da página 'gere_AGVs_DISABLED' 450
C.2.15.1 Posições 450
C.2.15.2 Transições 450
C.2.16 Transições da RP da página 'gere_AGVs_ENABLED' 452
C.2.16.1 DISABLE_AGV_ENABLED 452
C.2.16.2 MUDA_EST_ALOC_AGV_EN 452
C.2.16.3 REC_PASSAG_AGV_AUTOM 453
C.2.17 Transições na RP da página 'gere_Tarefas' 453
C.2.17.1 PROC_T_POSIC_ESPEC 453
C.2.18 Transições na RP da página 'gere_Tarefas_FILA' 454
C.2.18.1 REC_CANCEL_TAR_FILA 454
C.2.18.2 TAR_FILA_PASSO1 454
C.2.18.3 TAR_FILA_PASSO3 455
C.2.18.4 TAR_FILA_REJEITADA 455
C.2.19 Transições da RP da página 'gere_Tarefas_PASSO1' 456
C.2.19.1 REC_CANCEL_TAR_PASSO1 456
C.2.19.2 REC_FIM_T_POSICION 456
C.2.19.3 REC_REJEICAO_TAR_PASSO1 457
C.2.19.4 TAR_PASSO1_FILA 457
C.2.19.5 TAR_PASSO1_PASSO2 457
C.2.19.6 TAR_PASSO1_REAF 458
C.2.20 Transições da RP da página 'gere_Tarefas_PASSO2' 458
C.2.20.1 REC_CANCEL_TAR_PASSO2 458
C.2.20.2 REC_FIM_T_CARGA 459
C.2.20.3 TAR_PASSO2_PASSO3 459
C.2.21 Transições da RP da página 'gere_Tarefas_PASSO3' 460
C.2.21.1 REC_CANCEL_TAR_PASSO3 460
C.2.21.2 TAR_PASSO3_PASSO4 460
C.2.22 Transições da RP da página 'gere_Tarefas_PASSO4' 461
C.2.22.1 REC_CANCEL_TAR_PASSO4 461
C.2.22.2 REC_FIM_T_NORMAL_OU_DESCARGA 461
C.2.23 Entidades da RP da página 'gere_cancelamento_Tarefas' 462
C.2.23.1 Posições 462
C.2.23.2 Transições 462
C.2.24 Entidades da RP da página 'lanca_Tarefas' 464
C.2.24.1 Posições 464
C.2.24.2 Transições 464
C.2.25 Entidades da RP da página 'leitura_fich_nos_tar' 466
C.2.25.1 Posições 466
C.2.25.2 Transições 466
C.2.26 Entidades da RP da página 'proc_tar_canc_FILA_ou_PASSO1' 467
C.2.26.1 Posição 'TAREFA_A_REMOVER:AP_TAREFA' 467
C.2.26.2 Transições 467
C.3 CLASSE SGAGV 467
C.3.1 CMN units incluídas 467
C.3.2 Posições de entrada 468
C.3.3 Posições de saída 468
C.3.4 Parâmetros 468
C.3.5 Ficheiros de cabeçalho incluídos 469

C.3.6 Funções definidas internamente 469
C.3.7 Acções iniciais 471
C.3.8 Acções finais 476
C.3.9 Tipos de tokens 477
C.3.9.1 AP_LAYOUT 477
C.3.9.2 L_AGVS 478
C.3.9.3 L_BLOQ 478
C.3.9.4 APO_AGV 478
C.3.9.5 APO_TAREFA 478
C.3.9.6 L_TAREFAS 478
C.3.9.7 TAREFA 478
C.3.9.8 L_AP_TAREFAS 478
C.3.9.9 L_AP_AGVS 478
C.3.9.10 L_ESTADO_FILAS 478
C.3.10 Variáveis locais 478
C.3.11 Posições partilhadas 482
C.3.12 Entidades da RP principal da classe SGAGV 482
C.3.12.1 Posição 'ESTADO_FILAS:L_ESTADO_FILAS' 482
C.3.12.2 Transições 482
C.3.13 Entidades da RP da página 'cancela_movim_avan_AGVs' 483
C.3.13.1 Posição 'AGV_ESPERA_ELIMIN_MOV_AVAN:APO_AGV' 483
C.3.13.2 Transições 483
C.3.14 Entidades da RP da página 'cria_lista_AGVs' 484
C.3.14.1 Posição "ID_AGV:ULINT' 484
C.3.14.2 Transições 484
C.3.15 Entidades da RP da página 'escalona_e_despacha_tar_normais' 484
C.3.15.1 Posições 484
C.3.15.2 Transições 485
C.3.16 Entidades da RP da página 'gere_AGVs_MOVIMENTO' 487
C.3.16.1 Posições 487
C.3.16.2 Transições 487
C.3.17 Entidades da RP da página 'gere_AGVs_est_BLOQUEADO' 495
C.3.17.1 Posição 'AGV_FIM_POSICION:APO_AGV' 495
C.3.17.2 Transições 495
C.3.18 Entidades da RP da página 'introducao_AGVs' 501
C.3.18.1 Posições 501
C.3.18.2 Transições 501
C.3.19 Entidades da RP da página 'leitura_fich_config' 503
C.3.19.1 Posições 503
C.3.19.2 Transições 503
C.3.20 Entidades da RP da página 'passagem_modo_manual_AGVs' 504
C.3.20.1 Posição 'MENS_ESTADO_AGV:MENS_SGM_SGAGV' 504
C.3.20.2 Transições 504
C.3.21 Entidades da RP da página 'proc_cancel_tarefas' 506
C.3.21.1 Posições 506
C.3.21.2 Transições 506
C.3.22 Transições da RP da página 'proc_erros_AGVs' 509
C.3.22.1 REC_ERRO_AGV 509
C.3.22.2 REC_FIM_ERRO_AGV 510
C.3.23 Entidades da RP da página 'proc_mens_relativas_a_AGVs' 512
C.3.23.1 Posições 512
C.3.23.2 Transições 512
C.3.24 Entidades da RP da página 'proc_novas_tarefas' 516
C.3.24.1 Posições 516
C.3.24.2 Transições 516
C.3.25 Transições da RP da página 'proc_tarefas_AGVs_desactivados' 522
C.3.25.1 Acção de 'LIMPA_TAREFA_AGV_DESACTIVADO' 522
C.3.25.2 TRATA_TAR_AGV_DESACTIVADO 522
C.3.26 Transições da RP da página 'resolve_Deadlock_Circular' 524

C.3.26.1 RESOLVE_DEADLOCK_CIRCULAR 524
C.3.27 Entidades da RP da página 'resolve_conflitos_AGVs_ERRO' 524
C.3.27.1 Posições 524
C.3.27.2 Transições 524
C.3.28 Transições da RP da página 'resolve_conflitos_AGVs_PARADOS' 525
C.3.28.1 MOVE_AGV_PARADO_BLOQUEANTE 525
C.3.29 Transições da RP da página 'transicao_PARADO_BLOQUEADO' 527
C.3.29.1 AGV_PARADO_BLOQUEADO 527
C.4 CLASSE SPAGV 527
C.4.1 CMN units incluídas 527
C.4.2 Posições de entrada 528
C.4.3 Posições de saída 528
C.4.4 Parâmetros 528
C.4.5 Ficheiros de cabeçalho incluídos 529
C.4.6 Funções definidas internamente 529
C.4.7 Acções iniciais 530
C.4.8 Acções finais 530
C.4.9 Tipos de tokens 530
C.4.9.1 AP_LAYOUT 530
C.4.9.2 AP_NOS_INTR 531
C.4.9.3 AGV 531
C.4.10 Variáveis locais 531
C.4.11 Posições partilhadas 532
C.4.12 Entidades da RP da página principal 532
C.4.12.1 Objectos 532
C.4.12.2 Transições 533
C.4.13 Entidades da RP da página 'cancela_movim_avanco' 533
C.4.13.1 Posição 'ID_AGV_MOV_AVAN_CANCEL:ULINT' 533
C.4.13.2 Transições 533
C.4.14 Entidades da RP da página 'gera_AGVs_DISABLED' 534
C.4.14.1 Posição 'ID_AGV:ULINT' 534
C.4.14.2 Transições 534
C.4.15 Transições da RP da página 'gere_AGVs_DISABLED' 535
C.4.15.1 COLOCA_AGV_ENABLED 535
C.4.15.2 IGN_MENS_MOV_AGV_DIS 536
C.4.16 Entidades da RP da página 'gere_AGVs_ENABLED' 536
C.4.16.1 Posição 'AGV_ESPERA_CONF_INTR:AGV' 536
C.4.16.2 Transições 536
C.4.17 Entidades da RP da página 'gere_AGVs_ERRO' 539
C.4.17.1 Posições 539
C.4.17.2 Transições 539
C.4.18 Entidades da RP da página 'gere_AGVs_MOVIMENTO' 543
C.4.18.1 Posição 'AGV_MOV_ERRO:AGV' 543
C.4.18.2 Transições 543
C.4.19 Entidades de 'gere_AGVs_PARADOS' 546
C.4.19.1 Posição 'AGV_TRANSF_CARGA:AGV' 546
C.4.19.2 Transições 546
C.4.20 Entidades da RP da página 'leitura_fich_config' 549
C.4.20.1 Posições 549
C.4.20.2 Transições 549
C.4.21 Entidades da RP da página 'processa_AGVs_desactivados' 550
C.4.21.1 Posição 'AGV_DISABLED_OU_ENABLED:AGV' 550
C.4.21.2 Transições 550

ANEXO D – CLASSES ARTIFEX DO SISTEMA DE FABRICO 551
D.1 CLASSE CENTRO_RECEPCAO 551
D.1.1 CMN units incluídas 551
D.1.2 Posições de entrada 551
D.1.3 Posições de saída 551

D.1.4 Parâmetros 551
D.1.5 Ficheiros de cabeçalho incluídos 552
D.1.6 Funções definidas internamente 552
D.1.7 Acções iniciais 552
D.1.8 Acções finais 553
D.1.9 Variáveis locais 553
D.1.10 Posições partilhadas 554
D.1.11 Entidades da RP da página principal 554
D.1.11.1 Objectos 554
D.1.11.2 Posições 554
D.1.11.3 Transições 555
D.1.12 Entidades da RP da página 'inicializa_modelo' 559
D.1.12.1 Posições 559
D.1.12.2 Transições 559
D.2 CLASSE ARMAZEM_AUTOMATICO 561
D.2.1 CMN units incluídas 561
D.2.2 Posições de entrada 561
D.2.3 Posições de saída 561
D.2.4 Parâmetros 561
D.2.5 Ficheiros de cabeçalho incluídos 562
D.2.6 Funções definidas internamente 562
D.2.7 Acções iniciais 562
D.2.8 Acções finais 564
D.2.9 Variáveis locais 564
D.2.10 Posições partilhadas 565
D.2.11 Entidades da RP da página principal 565
D.2.11.1 Objecto 'TEMP_CD_AGV:TEMPORIZADOR' 565
D.2.11.2 Transições 565
D.2.12 Entidades da RP da página 'entrada_paletes' 566
D.2.12.1 Posições 566
D.2.12.2 Transições 566
D.2.13 Entidades da RP da página 'inicializa_modelo' 571
D.2.13.1 Posições 571
D.2.13.2 Transições 571
D.2.14 Entidades da RP da página 'saida_paletes' 573
D.2.14.1 Posições 573
D.2.14.2 Transições 573
D.3 CLASSE POSTO_TRABALHO 578
D.3.1 CMN units incluídas 578
D.3.2 Posições de entrada 578
D.3.3 Posições de saída 578
D.3.4 Parâmetros 579
D.3.5 Ficheiros de cabeçalho incluídos 579
D.3.6 Funções definidas internamente 579
D.3.7 Acções iniciais 580
D.3.8 Variáveis locais 581
D.3.9 Posições partilhadas 581
D.3.10 Entidades da RP da página principal 582
D.3.10.1 Objectos 582
D.3.10.1.1 TEMP_CD_AGV:TEMPORIZADOR 582
D.3.10.2 Transições 582
D.3.11 Entidades da RP da página 'entrada_paletes' 582
D.3.11.1 Posições 582
D.3.11.2 Transições 582
D.3.12 Entidades da RP da página 'inicializa_modelo' 586
D.3.12.1 Posições 586
D.3.12.2 Transições 586
D.3.13 Entidades da RP da página 'processamento_paletes' 587
D.3.13.1 Posições 587

D.3.13.2 Transições 587
D.3.14 Entidades da RP da página 'saida_paletes' 589
D.3.14.1 Posições 589
D.3.14.2 Transições 589
D.4 CLASSE CENTRO_EXPEDICAO 592
D.4.1 CMN units incluídas 592
D.4.2 Posições de entrada 592
D.4.3 Posições de saída 592
D.4.4 Parâmetros 592
D.4.5 Ficheiros de cabeçalho incluídos 593
D.4.6 Funções definidas internamente 593
D.4.7 Acções iniciais 593
D.4.8 Variáveis locais 594
D.4.9 Posições partilhadas 595
D.4.10 Entidades da RP da página principal 595
D.4.10.1 Objectos 595
D.4.10.2 Posições 595
D.4.10.3 Transições 595
D.4.11 Entidades da RP da página 'inicializa_modelo' 600
D.4.11.1 Posições 600
D.4.11.2 Transições 601
D.5 CLASSE SISTEMA_MANUFACTURA 602
D.5.1 CMN units incluídas 602
D.5.2 Posições de entrada 602
D.5.3 Posições de saída 602
D.5.4 Parâmetros 602
D.5.5 Ficheiros de cabeçalho incluídos 603
D.5.6 Funções definidas internamente 603
D.5.7 Acções iniciais 603
D.5.8 Variáveis locais 604
D.5.9 Objectos partilhados 604
D.5.10 Posições partilhadas 605
D.5.11 Entidades da RP da página 'encam_pal_AAMP_CM1_ou_CM2' 607
D.5.11.1 Posições 607
D.5.11.2 Transições 607
D.5.12 Entidades da RP da página 'encam_pal_AAPA_CQ_CEPA' 609
D.5.12.1 Posições 609
D.5.12.2 Transições 609
D.5.13 Entidades da RP da página 'encam_pal_CM1_PM1_PM2' 610
D.5.13.1 Posições 610
D.5.13.2 Transições 611
D.5.14 Entidades da RP da página 'encam_pal_CM2_PM3_PM4' 612
D.5.14.1 Posições 612
D.5.14.2 Transições 612
D.5.15 Entidades da RP da página 'encam_pal_CM4_CM' 614
D.5.15.1 Posições 614
D.5.15.2 Transições 614
D.5.16 Entidades da RP da página 'encam_pal_CM_CQ_AAPA' 615
D.5.16.1 Posições 615
D.5.16.2 Transições 615
D.5.17 Entidades de RP da página 'encam_pal_CQ_AAPA_CEPA' 616
D.5.17.1 Posições 616
D.5.17.2 Transições 617
D.5.18 Entidades da RP da página 'encam_pal_CRMP_AAMP_ou_CM1' 618
D.5.18.1 Posições 618
D.5.18.2 Transições 618
D.5.19 Entidades da RP da página 'encam_pal_PM1_PM3_CM3' 620
D.5.19.1 Posições 620
D.5.19.2 Transições 620

D.5.20 Entidades da RP da página 'encam_pal_PM2_PM4_CM3' 621
D.5.20.1 Posições 621
D.5.20.2 Transições 621
D.5.21 Entidades da RP da página 'encam_pal_PM3_PM5_PM6' 623
D.5.21.1 Posições 623
D.5.21.2 Transições 623
D.5.22 Entidades da RP da página 'encam_pal_PM4_PM5_PM6' 624
D.5.22.1 Posições 624
D.5.22.2 Transições 624
D.5.23 Entidades da RP da página 'encam_pal_PM5_CM4_CM' 626
D.5.23.1 Posições 626
D.5.23.2 Transições 626
D.5.24 Entidades da RP da página 'encam_pal_PM6_CM4_CM' 627
D.5.24.1 Posições 627
D.5.24.2 Transições 627
D.5.25 Entidades da RP da página 'escam_pal_CM3_PM5_PM6' 629
D.5.25.1 Posições 629
D.5.25.2 Transições 629

ANEXO E – CLASSE ARTIFEX SISTEMA_DE_AGVS 631
E.1 PARÂMETROS 631
E.2 FICHEIROS DE CABEÇALHO INCLUÍDOS 631
E.3 ACÇÕES INICIAIS 631
E.4 ACÇÕES FINAIS 631
E.5 VARIÁVEIS LOCAIS 631
E.6 OBJECTOS 631

ANEXO F – FICHEIROS DE CONFIGURAÇÃO DO SAGVS 633
F.1 PARÂMETROS DO SAGVS ('SISTAGVS.INI') 633
F.2 PARÂMETROS DO SISTEMA DE FABRICO ('SISMANUF.INI') 634
F.3 CONFIGURAÇÃO DO LAYOUT ('LAYOUT.TXT') 637
F.4 CONFIGURAÇÃO DOS PONTOS DE CARGA/DESCARGA ('MESAS.TXT') 643
F.5 CONFIGURAÇÃO DOS NÓS PARA INTRODUÇÃO E POSICIONAMENTO DE AGVS ('NINTRTAR.TXT') 644

ANEXO G – EXCERTOS DO FICHEIRO DE LOG DE UMA SIMULAÇÃO DO SAGVS 645

Índice de Figuras

Figura 2.1 – O AGV do CCP – Centro de CIM do Porto – com fios de seguimento implantados no solo. 8
Figura 2.2 – Princípio de funcionamento dos AGVs filoguiados 9
Figura 2.3 – Diagrama de blocos da arquitectura de 3 camadas de um SAGVs. 11
Figura 3.1 – Exemplo de um grafo representativo de um layout de trajectórias. 19
Figura 3.2 – Determinação do sentido do bloqueio de um nó do layout. 26
Figura 3.3 – Exemplo de um layout muito simples. 27
Figura 3.4 – Layout com áreas partilhadas. 28
Figura 3.5 – Solução para o problema do layout com áreas partilhadas. 29
Figura 3.6 – Grafo de estados representativo de uma situação de deadlock. 30
Figura 3.7 – Deadlock causado por uma gestão de tráfego inconveniente. 31
Figura 3.8 – Deadlock causado por um planeamento de trajectórias descuidado. 32
Figura 3.9 – Deadlock causado por más decisões no despacho dos AGVs. 32
Figura 3.10 – Deadlock causado por um mau controlo do fluxo de material. 33
Figura 4.1 – Exemplo de uma RP. 51
Figura 4.2 – Exemplo de uma RP marcada. 52
Figura 4.3 – Disparo de uma transição. 53
Figura 4.4 – Exemplo de uma RP não autónoma. 54
Figura 4.5 – Exemplo da transformação de uma RP impura numa RP pura. 57
Figura 4.6 – RP com a sua marcação inicial. 58
Figura 4.7 – Marcações atingíveis de uma RP. 59
Figura 4.8 – Uma RP limitada. 60
Figura 4.9 – Uma RP não limitada. 60
Figura 4.10 – Ilustração das propriedades transição viva e RP viva. 61
Figura 4.11 – RP quase-vivas: (a) Com um deadlock; (b) Livre de deadlocks. 62
Figura 4.12 – RP com conflitos. 63
Figura 4.13 – Modelização de sincronização com uma RP. 66
Figura 4.14 – Modelização da partilha de um recurso com uma RP. 67
Figura 4.15 – Modelização da memorização com uma RP: a) De um disparo; b) De um número 67
Figura 4.16 – Modelização da leitura numa RP. 68
Figura 4.17 – Modelização de uma escolha numa RP. 68
Figura 4.18 – Modelização de uma sequência de eventos numa RP. 68
Figura 4.19 – Modelização de paralelismo e de concorrência numa RP. 69
Figura 4.20 – RP com um circuito simples e um auto-ciclo. 69
Figura 5.1 – Disparo de uma transição sincronizada. 72
Figura 5.2 – Disponibilidade de uma marca numa RP P-temporizada. 74
Figura 5.3 – Reserva de uma marca numa RP T-temporizada. 75
Figura 5.4 – Transformação de uma RP T-temporizada numa RP P-temporizada 76
Figura 5.5 – Transformação de uma RP P-temporizada numa RP T-temporizada 77
Figura 5.6 – RP interpretada. 78
Figura 5.7 – Exemplo de uma RP estocástica. 79
Figura 5.8 – Dois sistemas e suas RP correspondentes. 81
Figura 5.9 – (a) Modelo não descrevendo correctamente o funcionamento dos dois carrinhos. b) Modelo RPC sem funções explícitas. c) Modelo RPC com as funções dos arcos. 82
Figura 5.10 – Modelo RPC compacto do funcionamento de ambos os carrinhos. 83
Figura 6.1 – Exemplo de uma RPC construída com a ferramenta Design/CPN. 91
Figura 6.2 – Elementos constituintes de uma RP do Artifex. 96
Figura 7.1 – Definição da posição relativa de uma mesa de carga/descarga em relação ao respectivo nó do layout e determinação do lado para o qual o AGV trabalha. 115
Figura 8.1 – Diagrama Artifex de um modelo de um SAGVs inserido num sistema de fabrico. 117
Figura 8.2 – RP de um objecto da classe TEMPORIZADOR. 119
Figura 8.3 – Página principal da RP do SGM. 120
Figura 8.4 – Sub-rede 'leitura_fich_nos_tar' do SGM. 121

Figura 8.5 – Sub-rede 'gere_AGVs' do SGM. 122
Figura 8.6 – Sub-rede 'gera_AGVs_DISABLED' do SGM. 123
Figura 8.7 – Sub-rede 'gere_AGVs_DISABLED' do SGM. 124
Figura 8.8 – Sub-rede 'gere_AGVs_ENABLED' do SGM 125
Figura 8.9 – Sub-rede 'gere_AGVs_ACTIVOS' do SGM. 125
Figura 8.10 – Sub-rede 'gere_Tarefas' do SGM. 126
Figura 8.11 – Sub-rede 'lanca_Tarefa' do SGM. 127
Figura 8.12 – Sub-rede 'gere_cancelamento_Tarefas' do SGM. 129
Figura 8.13 – Sub-rede 'gere_Tarefas_FILA' do SGM. 130
Figura 8.14 – Sub-rede 'gere_Tarefas_PASSO1' do SGM 131
Figura 8.15 – Sub-rede 'gere_Tarefas_PASSO2' do SGM. 132
Figura 8.16 – Sub-rede 'gere_Tarefas_PASSO3' do SGM. 132
Figura 8.17 – Sub-rede 'gere_Tarefas_PASSO4' do SGM. 133
Figura 8.18 – Sub-rede 'proc_tar_canc_FILA_ou_PASSO1' do SGM. 133
Figura 8.19 – Página principal da RP do SGAGV. 135
Figura 8.20 – Sub-rede 'leitura_fich_config' do SGAGV. 136
Figura 8.21 – Sub-rede 'cria_lista_AGVs' do SGAGV. 137
Figura 8.22 – Sub-rede 'proc_mens_receb_SGM' do SGAGV. 138
Figura 8.23 – Sub-rede 'proc_mens_relativas_a_AGVs' do SGAGV. 140
Figura 8.24 – Sub-rede 'proc_mens_relativas_a_Tarefas' do SGAGV. 141
Figura 8.25 – Sub-rede 'proc_novas_tarefas' do SGAGV. 142
Figura 8.26 – Sub-rede 'proc_cancel_tarefas' do SGAGV. 143
Figura 8.27 – Sub-rede 'cancela_movim_avan_AGVs' do SGAGV. 144
Figura 8.28 – Sub-rede 'escalona_e_despacha_tar_normais' do SGAGV. 145
Figura 8.29- Sub-rede 'gere_AGVs' do SGAGV. 146
Figura 8.30 – Sub-rede 'introducao_AGVS' do SGAGV. 147
Figura 8.31 – Sub-rede 'transicao_PARADO_BLOQUEADO' do SGAGV. 148
Figura 8.32 – Sub-rede 'proc_erros_AGVs' do SGAGV. 148
Figura 8.33 – Sub-rede 'passagem_modo_manual_AGVs' do SGAGV 149
Figura 8.34 – Sub-rede 'gere_AGVs_est_BLOQUEADO' do SGAGV. 150
Figura 8.35 – Sub-rede 'gere_AGVs_MOVIMENTO' do SGAGV. 152
Figura 8.36 – Sub-rede 'resolve_conflitos_AGVs_PARADOS' do SGAGV. 152
Figura 8.37 – Sub-rede 'resolve_Deadlock_Circular' do SGAGV. 153
Figura 8.38 – Sub-rede 'resolve_conflitos_AGVs_ERRO' do SGAGV. 154
Figura 8.39 – Sub-rede 'proc_tarefas_AGVs_desactivados' do SGAGV. 155
Figura 8.40 – Página principal da RP do SPAGV. 156
Figura 8.41 – Sub-rede 'leitura_fich_config' do SPAGV. 158
Figura 8.42 – Sub-rede 'gera_AGVs_DISABLED' do SPAGV. 159
Figura 8.43 – Sub-rede 'cancela_movim_avanco' do SPAGV. 160
Figura 8.44 – Sub-rede 'processa_AGVs_desactivados' do SPAGV 161
Figura 8.45 – Sub-rede 'gere_AGVs_ENABLED' do SPAGV. 162
Figura 8.46 – Sub-rede 'gere_AGVs_PARADOS' do SPAGV. 164
Figura 8.47 – Sub-rede 'gere_AGVs_MOVIMENTO' do SPAGV. 166
Figura 8.48 – Sub-rede 'gere_AGVs_ERRO' do SPAGV. 167
Figura 9.1 – Desenho da planta fabril exemplo utilizada na realização do trabalho. 170
Figura 9.2 – RP da página principal da classe CENTRO_RECEPCAO. 173
Figura 9.3 - Sub-rede 'inicializa_modelo' do CENTRO_RECEPCAO. 174
Figura 9.4 – RP da página principal da classe ARMAZEM_AUTOMATICO. 177
Figura 9.5 – Sub-rede 'inicializa_modelo' da classe ARMAZEM_AUTOMATICO. 178
Figura 9.6 – Sub-rede 'entrada_paletes' da classe ARMAZEM_AUTOMATICO. 180
Figura 9.7 – Sub-rede 'saida_paletes' da classe ARMAZEM_AUTOMATICO. 181
Figura 9.8 – Página principal da RP da classe POSTO_TRABALHO. 183
Figura 9.9 – Sub-rede 'inicializa_modelo' da classe POSTO_TRABALHO. 185
Figura 9.10 – Sub-rede 'entrada_paletes' da classe POSTO_TRABALHO. 186
Figura 9.11 – Sub-rede 'processamento_paletes' da classe 'POSTO_TRABALHO'. 187
Figura 9.12 – Sub-rede 'saida_paletes' da classe POSTO_TRABALHO. 188
Figura 9.13 – RP da página principal da classe CENTRO_EXPEDICAO. 190
Figura 9.14 – Sub-rede 'inicializa_modelo' da classe CENTRO_EXPEDICAO. 191
Figura 9.15 – RP da página principal da classe SISTEMA_MANUFACTURA. 194

Figura 9.16 – Objectos instanciados na RP da classe SISTEMA_MANUFACTURA. 195
Figura 9.17 – Sub-rede 'Controlo_CRMP' da classe SISTEMA_MANUFACTURA. 196
Figura 9.18 – Sub-rede 'encam_pal_CRMP_AAMP_ou_CM1 da classe SISTEMA_MANUFACTURA. 197
Figura 9.19 – Sub-rede 'Controlo_AAMP' da classe SISTEMA MANUFACTURA. 198
Figura 9.20– Sub-rede 'encam_pal_AAMP_CM1_ou_CM2' da classe SISTEMA MANUFACTURA. 199
Figura 9.21 – Sub-rede 'Controlo_CM1' da classe SISTEMA_MANUFACTURA. 200
Figura 9.22 – Sub-rede 'encam_pal_CM1_PM1_PM2' da classe SISTEMA_MANUFACTURA. 201
Figura 9.23 – Sub-rede 'Controlo_CM2' da classe SISTEMA_MANUFACTURA. 202
Figura 9.24 – Sub-rede 'encam_pal_CM2_PM3_PM4' da classe SISTEMA_MANUFACTURA. 203
Figura 9.25 – Sub-rede 'Controlo_PM1' da classe SISTEMA_MANUFACTURA. 204
Figura 9.26 – Sub-rede 'encam_pal_PM1_PM3_CM3' da classe SISTEMA_MANUFACTURA. 205
Figura 9.27 – Sub-rede 'Controlo_PM2' da classe SISTEMA_MANUFACTURA. 206
Figura 9.28 – Sub-rede 'encam_pal_PM2_PM4_CM3' da classe SISTEMA_MANUFACTURA. 207
Figura 9.29 – Sub-rede 'Controlo_PM3' da classe SISTEMA_MANUFACTURA. 208
Figura 9.30 – Sub-rede 'encam_pal_PM3_PM5_PM6' da classe SISTEMA_MANUFACTURA. 209
Figura 9.31 – Sub-rede 'Controlo_PM4' da classe SISTEMA_MANUFACTURA. 210
Figura 9.32 – Sub-rede 'encam_pal_PM4_PM5_PM6' da classe SISTEMA_MANUFACTURA. 211
Figura 9.33 – Sub-rede 'Controlo_PM5' da classe SISTEMA_MANUFACTURA. 212
Figura 9.34 – Sub-rede 'encam_pal_PM5_CM4_CM' da classe SISTEMA_MANUFACTURA. 213
Figura 9.35 – Sub-rede 'Controlo_PM6' da classe SISTEMA_MANUFACTURA. 214
Figura 9.36 – Sub-rede 'encam_pal_PM6_CM4_CM' da classe SISTEMA_MANUFACTURA. 215
Figura 9.37 – Sub-rede 'Controlo_CM3' da classe SISTEMA_MANUFACTURA. 216
Figura 9.38 – Sub-rede 'encam_pal_CM3_PM5_PM6' da classe SISTEMA_MANUFACTURA. 217
Figura 9.39 – Sub-rede 'Controlo_CM4' da classe SISTEMA_MANUFACTURA. 218
Figura 9.40 – Sub-rede 'encam_pal_CM4_CM' da classe SISTEMA_MANUFACTURA. 219
Figura 9.41 – Sub-rede 'Controlo_CM' da classe SISTEMA_MANUFACTURA. 220
Figura 9.42 – Sub-rede 'encam_pal_CM_CQ_AAPA' da classe SISTEMA_MANUFACTURA. 221
Figura 9.43 – Sub-rede 'Controlo_CQ' da classe SISTEMA_MANUFACTURA. 222
Figura 9.44 – Sub-rede 'encam_pal_CQ_AAPA_CEPA' da classe SISTEMA_MANUFACTURA. 223
Figura 9.45 – Sub-rede 'Controlo_AAPA' da classe SISTEMA_MANUFACTURA. 224
Figura 9.46 – Sub-rede 'encam_pal_AAPA_CQ_CEPA' da classe SISTEMA_MANUFACTURA. 225
Figura 10.1 – Exemplificação da definição de um nó fictício no layout do SAGVs simulado. 227
Figura 10.2 – Planta do layout do SAGVs simulado. 228
Figura 10.3 – Gráfico do número de Tarefas do tipo 'NORMAL' executadas em cada simulação. 234
Figura 10.4 – Gráfico representativo do desempenho da gestão de tráfego. 239
Figura 10.5 – Distribuição média dos tempos de permanência de uma tarefa em cada estado. 241
Figura 10.6 – Valor médio do tempo de execução total e do tempo de operação total de uma tarefa. 243
Figura 10.7 – Gráfico da evolução da taxa de utilização de um AGV ao longo de um dia de trabalho. 245
Figura 10.8 – Distribuição média do tempo de permanência de um AGV em cada estado. 247
Figura 10.9 – Gráfico com a média de arranques por trajecto para um AGV descarregado em cada uma das simulações efectuadas. 248
Figura 10.10 – Número de paletes existentes no AAMP ao longo de uma simulação. 250
Figura 10.11 – Gráfico do número de paletes produzidas de produto acabado. 254

Lista de Tabelas

Tabela 3.1 – Matriz representativa do grafo da Figura 3.1. 19
Tabela 3.2 – Lista representativa do grafo da Figura 3.1. 19
Tabela 3.3 – Grandezas do modelo da fábrica para a estimação do número óptimo de AGVs. 21
Tabela 3.4 – Operações de carga/descarga permitidas por cada tipo de nós existentes num layout de AGVs. 25
Tabela 3.5 – Tipos de ramos existentes num layout de AGVs. 26
Tabela 3.6 – Sintaxe do ficheiro de configuração do layout. 26
Tabela 4.1 – Estruturas particulares de RP. 55
Tabela 6.1 – Ferramentas de suporte à modelização com RP. 87
Tabela 7.1 – Estados de movimentação possíveis de um AGV. 104
Tabela 7.2 – Estados de alocação de um AGV. 104
Tabela 7.3 – Estados de carga de um AGV. 105
Tabela 7.4 – Estados possíveis de uma tarefa. 105
Tabela 7.5 – Tipos de tarefas existentes. 106
Tabela 7.6 – Mensagens que o SGM pode enviar ao SGAGV. 107
Tabela 7.7 – Mensagens que o SGAGV pode enviar ao SGM. 108
Tabela 7.8 – Erros pelos quais o SGM pode ser notificado pelo SGAGV, através da mensagem (CONFIRMACAO, INSUCESSO, <erro>). 108
Tabela 7.9 – Razões possíveis para o cancelamento de uma tarefa por parte do SGAGV. 109
Tabela 7.10 – Mensagens que o SGAGV pode enviar ao SPAGV. 109
Tabela 7.11 – Valores dos parâmetros 1 e 2 de uma mensagem do tipo MOVIMENTO. 110
Tabela 7.12 – Valores possíveis para o parâmetro 1 de uma mensagem do tipo ESTADO_INTR. 110
Tabela 7.13 – Tipos de mensagens que o SPAGV pode enviar ao SGAGV. 110
Tabela 7.14 – Sintaxe de uma linha do ficheiro de configuração da posição dos pontos de carga/descarga em relação aos respectivos nós do layout. 115
Tabela 8.1 – Sintaxe de uma linha do ficheiro de configuração de nós de introdução e de tarefas de posicionamento. 121
Tabela 9.1 – Definição do caminho tecnológico das paletes do sistema de fabrico modelizado. 193
Tabela 9.2 – Ligação entre o objecto da classe SGM e o objecto da classe SISTEMA_MANUFACTURA. 225
Tabela 9.3 – Ligação entre o objecto da classe SGM e o objecto da classe SGAGV. 226
Tabela 9.4 – Ligação entre o objecto da classe SGAGV e o objecto da classe SISTEMA_MANUFACTURA. 226
Tabela 9.5 – Ligação entre o objecto da classe SGAGV e o objecto da classe SPAGV. 226
Tabela 9.6 – Ligação entre o objecto da classe SPAGV e o objecto da classe SISTEMA_MANUFACTURA. 226
Tabela 10.1 – Regras de afectação utilizadas nas simulações. 229
Tabela 10.2 – Valores dos parâmetros do SGM utilizados nas simulações. 229
Tabela 10.3 – Valores dos parâmetros do SGAGV utilizados nas simulações. 230
Tabela 10.4 – Valores dos parâmetros do SGAGV utilizados nas simulações. 230
Tabela 10.5 – Valores dos parâmetros da fábrica utilizados nas simulações. 230
Tabela 10.6 – Valores dos parâmetros do CRMP utilizados nas simulações. 231
Tabela 10.7 – Valores dos parâmetros dos armazéns automáticos utilizados nas simulações. 231
Tabela 10.8 – Valores dos parâmetros dos centros de maquinação, postos de trabalho manuais, célula de montagem e centro de controle de qualidade utilizados nas simulações. 232
Tabela 10.9 – Valores dos parâmetros do cais de expedição de produto acabado utilizados nas simulações. 232
Tabela 10.10 – Contagens de tarefas. 234
Tabela 10.11 – Estatísticas relacionadas com acções de controlo realizadas. 235
Tabela 10.12 – Estatísticas relacionadas com reafectações dinâmicas de tarefas. 235
Tabela 10.13 – Estatísticas relacionadas com erros ocorridos nos AGVs. 236
Tabela 10.14 – Estatísticas relacionadas com a resolução de conflitos ocorridos na gestão de tráfego. 236
Tabela 10.15 – Estatísticas relacionadas com o controlo dos movimentos dos AGVs e com o desempenho da gestão de tráfego. 238
Tabela 10.16 – Tempos característicos da permanência de uma tarefa nos vários estados possíveis. 240
Tabela 10.17 – Classificação de estratégias de escalonamento com base no tempo médio de atendimento de uma tarefa. 241

Tabela 10.18 – Classificação de estratégias de escalonamento com base no tempo médio de execução do passo 1 de uma tarefa. 242
Tabela 10.19 – Tempos totais de execução de uma tarefa. 242
Tabela 10.20 – Taxa de utilização dos AGVs. 244
Tabela 10.21 – Tempos característicos da permanência dos AGVs em cada estado. 246
Tabela 10.22 – Estatísticas dos movimentos efectuados pelos AGVs. 248
Tabela 10.23 – Número de paletes do CRMP. 249
Tabela 10.24 – Número de paletes dos armazéns. 250
Tabela 10.25 – Número de paletes dos centros de maquinação, postos de trabalho manuais, célula de montagem e centro de controle de qualidade. 251
Tabela 10.26 – Número de paletes do CEPA. 251
Tabela 10.27 – Comprimento médio das filas dos pontos de descarga. 252
Tabela 10.28 – Comprimento médio das filas dos pontos de carga. 253
Tabela 10.29 – WIP médio e total de paletes de produto acabado produzidas. 253

Lista de Acrónimos

AGV – Veículo Guiado Automaticamente (da expressão inglesa *Automatic Guided Vehicle*).

AGVs – plural de AGV.

FMS –Célula Flexível de Fabrico (da expressão inglesa *Flexible Automation System*).

SAGVs – Sistema(s) de AGVs.

CIM – Fabrico Integrado por Computador (da expressão inglesa *Computer Integrated Manufacturing*).

JIT – Mesmo-a-Tempo (da expressão inglesa *Just-In-Time*).

TQM – Gestão de Qualidade Total (da expressão inglesa *Total Quality Management*).

WIP – Em-Curso-de-Fabrico (da expressão inglesa *Work-In-Process*).

RP – Rede(s) de Petri.

SGAGVs – Sistema de Gestão de AGVs.

GPS – Sistema de Posicionamento Global (da expressão inglesa *Global Positioning System*).

SPAGV – Sistema de Posicionamento de AGVs.

SGAGV – Sistema de Gestão de AGVs.

SGM – Sistema Geral de Movimentação.

VMP – Veículo Mais Próximo (regra de despacho de AGVs).

VMU – Veículo Menos Utilizado (regra de despacho de AGVs).

TMCV – Tempo Mais Curto de Viagem (regra de despacho de AGVs).

MTFS – Máximo Tamanho da Fila de Saída (regra de despacho de AGVs).

MELFS – Mínimo Espaço Livre na Fila de Saída (regra de despacho de AGVs).

PCPSM – Primeiro a Chegar Primeiro a ser Servido Modificada (regra de despacho de AGVs).

RPC – Rede(s) de Petri coloridas.

CPN – Sigla inglesa utilizada na literatura para designar RPC (*coloured Petri nets*).

FDP – Função de densidade de probabilidade.

FIFO – Sigla inglesa que significa *first-in-first-out*, habitualmente utilizada para designar uma política de gestão de filas de espera, em que a primeira entidade a dar entrada, é a primeira a sair.

LIFO – Sigla inglesa que significa *last-in-first-out*, habitualmente utilizada para designar uma política de gestão de pilhas, em que a última entidade a dar entrada, é a primeira a sair.

CRMP – Centro de recepção de matérias-primas.

AAMP – Armazém automático de matérias-primas.

CMx – Centro de maquinação 'x'.

CM – Célula de montagem.

PMx – Posto de trabalho manual 'x'.

CQ – Centro de controle de qualidade.

AAPA – Armazém automático de produto acabado.

CEPA – Centro de expedição de produto acabado.

TU – Taxa de utilização de um AGV.

Glossário

JIT – Paradigma organizacional que procura eliminar a existência de *stocks* nos sistemas de fabrico, defendendo a recepção da matéria-prima ou dos componentes requeridos pelas operações de fabrico "Mesmo-a-Tempo", ou seja, exactamente no momento em que forem necessários.

TQM – Paradigma de Gestão da Qualidade Total que procura evitar tarefas de controlo de qualidade, à custa da garantia da Qualidade Total em todas as fases do processo produtivo.

WIP – Designação utilizada para referir a quantidade de peças em processamento ou à espera de serem processadas em *buffers*, no interior de um sistema de fabrico.

Tempo de *Setup* – Tempo necessário à preparação de uma máquina para a produção de outro tipo de peças (tempo de *download* do programa de uma máquina de controlo numérico ou de um robô, tempo necessário à troca de ferramenta de uma máquina, etc).

Time-to-Market – Tempo de lançamento de um novo produto no mercado, compreendendo a especificação, a concepção (ou engenharia do produto), o plano de produção (ou engenharia do processo) e a comercialização ou *marketing* do produto.

Layout – disposição física dos equipamentos de produção de uma fábrica (das máquinas e estações de trabalho) no espaço físico disponível, ou planificação das pistas ou trajectórias de um sistema de AGVs. Este termo é por vezes utilizado para referir o espaço físico respectivo (e.g. espaço reservado à passagem dos AGVs).

Lean-Production – paradigma de gestão industrial que tem por objectivo a minimização dos desperdícios resultantes de tempos mortos (sub-utilização do parque produtivo de uma fábrica), matéria-prima residual desaproveitada, custos de armazenamento, etc.

Deadlock – termo habitualmente utilizado para designar uma situação de bloqueio em que as entidades intervenientes (AGVs, processos a correr num computador, etc) estão permanentemente bloqueados, como resultado de uma alocação errada de recursos, em que cada entidade detém um subconjunto dos recursos necessários e está à espera pela libertação de recursos afectados a outras entidades do mesmo grupo.

Job Shop – termo inglês habitualmente utilizado na literatura para designar empresas que fabricam uma grande variedade de produtos e em pequenos lotes (ou quantidades).

Buffer – designação habitualmente utilizada na literatura sobre sistemas de fabrico para designar um armazém temporário de peças em processamento. É utilizado para evitar paragens de máquinas quando os tempos de processamento de máquinas ou células de fabrico consecutivas na cadeia de produção forem diferentes.

AGV filoguiado – AGV cujo sistema de navegação se baseia no seguimento do campo magnético criado por anéis de corrente implantados no solo do *layout* de trabalho.

Nó físico de um *layout* de AGVs – entidade utilizada no âmbito dos SAGVs para representar um espaço físico do *layout* onde em cada momento pode haver no máximo um AGV.

Nó fictício de um *layout* de AGVs – entidade utilizada na definição do *layout* de um SAGVs, que apesar de ter uma representação comum à dos nós físicos, não representa uma possível posição dos AGVs. São utilizados para gerir áreas partilhadas, tais como cruzamentos e trajectórias vizinhas onde exista a possibilidade de colisão entre AGVs. A sua função é análoga à função dos semáforos nos sistemas operativos multi-tarefa, onde estes são utilizados para partilhar recursos entre os vários processos a correr em paralelo.

Roll-back – designação utilizada na linguagem informática para designar um mecanismo que permite abortar a acção de um programa e colocá-lo no estado imediatamente anterior ao início da execução da acção em causa (e.g. alteração de informação numa base de dados). Serve para colocar o sistema num estado coerente, quando ocorram erros durante a execução de uma acção, cuja interrupção a meio, poderia deixar o sistema num estado incoerente.

Ranking – classificação ordenada segundo um determinado critério (e.g. distância de um trajecto no caso da ordenação de trajectos).

Espaço de estados – designação por vezes utilizada para designar o grafo de marcações de uma RP, ou seja, os estados de uma RP, atingíveis a partir de uma determinada marcação inicial.

Espaço de estados condensado – designação por vezes utilizada para designar a árvore de cobertura de uma RP, ou seja, o grafo de marcações sem ciclos.

Token – termo inglês habitualmente utilizado para designar uma marca de uma RP.

Debug – fase de desenvolvimento de *software*, em que se procura validar um programa e, eventualmente, corrigir erros, utilizando técnicas de corrida dos programas passo-a-passo, ou com a definição de *breakpoints*.

Breakpoint – designação utilizada em linguagem informática para referir uma condição de paragem da execução de um programa, quando este corre em modo *debug*.

Flag – designação inglesa habitualmente utilizada para referir uma variável booleana, que só pode tomar o valor 0 ou 1.

Palete – designação utilizada para referir uma unidade de matéria-prima, de em-curso-de-fabrico ou de produto acabado, manuseada e transportada por um AGV.

Stocador – designação habitualmente utilizada pelos fabricantes de armazéns automáticos para referir o veículo que é responsável pelo manuseamento das paletes no interior do armazém; nos armazéns de grandes dimensões, este veículo é habitualmente um corpo móvel, dotado de uma plataforma com um sistema de garfos telescópicos, para a realização de cargas e de descargas, e que se move ao longo do armazém sobre carris.

1. Introdução

Neste capítulo introdutório é caracterizado o conceito actual de integração de empresa, procurando-se enquadrar a importância de um bom desempenho ao nível do Sistema de AGVs na obtenção de um desempenho global do sistema de fabrico satisfatório. São apontados os problemas que são objecto de estudo desta dissertação. São também apresentados os objectivos do trabalho e a metodologia seguida durante a sua realização. No final do capítulo é feita uma breve descrição dos assuntos abordados nos capítulos subsequentes.

1.1 O papel dos Sistema de AGVs na integração dos sistemas de fabrico flexíveis

Na actualidade, a utilização de Sistemas de Fabrico Flexíveis (da sigla inglesa FMS – *Flexible Manufacturing System*) começa a generalizar-se, em consequência da importância que adquiriram na evolução tecnológica das empresas, como um meio para aumentarem a sua flexibilidade e para se adequarem à maior dinâmica e às maiores exigências do mercado actual [1].

Muitos dos progressos tecnológicos ao nível da automação dos sistemas de fabrico têm-se centrado em níveis inferiores (próximos ou muito próximos do *hardware*), resultando no aparecimento de máquinas mais sofisticadas, com menores tempos de maquinação, e menores tempos de *setup*. Devido a estes desenvolvimentos tecnológicos, o planeamento da produção começou a dar menos importância ao tempo de execução das operações de produção, passando a valorizar mais a eficiência com que os materiais são transportados entre estações de trabalho e entre estas e os armazéns [2].

Nos modernos ambientes de fabrico, este transporte é habitualmente assegurado por um sistema automático de manuseamento e de transporte de materiais, baseado em AGVs, que passará a ser designado daqui em diante por Sistema de AGVs (SAGVs).

1.1.1 Evolução do conceito de automatização industrial em direcção à integração empresarial

Na década de 80, a tendência em Automação Industrial consistia em aumentar a utilização de Sistemas de Fabrico, procurando aproveitar o potencial que estes oferecem no que respeita à flexibilidade e à maior capacidade de responder às exigências de um mercado cada vez mais severo e competitivo [3].

Nessa altura multiplicaram-se os cenários denominados por ilhas de automação. Estas consistiam em processos isolados de fabrico ou de gestão altamente automatizados e informatizados num ambiente heterogéneo, implicando a existência de muitas tarefas manuais de fabrico e de processamento da informação que colmatassem essas falhas de comunicação e de compatibilidade entre as várias ilhas de automação, que eram portanto sistemas fechados.

Este foi o primeiro estádio do conceito de Fabrico Integrado por Computador (da sigla inglesa CIM – *Computer Integrated Manufacturing*). Apesar de se terem alcançado melhorias significativas de produtividade e de qualidade, como resultado da progressiva substituição de operações manuais por operações parcialmente ou totalmente automatizadas, no final da década de 80 havia a percepção das limitações inerentes ao esforço de automatização que vinha sendo levado a cabo. Era altura de reflectir sobre as causa dessas insuficiências e de reformular o conceito de inovação tecnológica das empresas.

Ao longo da década de 90, as exigências do mercado têm evoluído no sentido da redução do tempo de lançamento de novos produtos (conhecido na literatura por *time-to-market*), e da maior diversidade nas características e soluções dos produtos oferecidos. Os próprios produtos tendem a ter ciclos de vida cada vez mais curtos, ou seja, o ritmo de inovação é cada vez mais acelerado.

No actual cenário macro-económico exigente, a competitividade das empresas está directamente relacionada com a sua capacidade de inovação. Esta inovação ao nível das soluções e dos produtos oferecidos implica necessariamente a inovação tecnológica das empresas, que lhes permitam produzir uma grande diversidade de produtos, à custa de uma planta fabril flexível, caracterizada pela facilidade de redefinição do processo produtivo ao nível do *layout* e do equipamento.

Essa inovação tecnológica vai muito para além da concepção de células de produção isoladas com um elevado grau de automatização e com uma elevado desempenho, porque estas potencialidades tecnológicas não se traduzirão numa melhoria global do sistema de fabrico, se não for possível integrar esforços e garantir a articulação entre os vários recursos de fabrico, nomeadamente máquinas, sistemas de transporte, sistemas de montagem e operários, bem como integrar o sistema de fabrico com o sistema de gestão.

O conceito de CIM pode ser definido como a integração do fluxo de materiais com o fluxo de informação, permitindo gerir o fluxo de materiais e os recursos de fabrico que processam os materiais [3]. Actualmente, o conceito de CIM dá maior importância a aspectos organizacionais, habitualmente denominados por factores humanos, que no passado eram relegados para segundo plano, mas que agora são reconhecidos como tendo um papel preponderante no funcionamento global de uma empresa industrial.

Tomou-se consciência de que o factor humano é o elemento diferenciador em ambientes globais em cada vez mais rápida mutação, onde a capacidade criativa, de melhoria contínua e de aprendizagem se sobrepõe às soluções meramente tecnológicas, facilmente transaccionáveis.

Assim, o conceito actual de CIM consiste na integração do fluxo de materiais inerente ao processo de fabrico e à logística intra-empresarial, com o fluxo de informação, que gere e controla tanto esse fluxo de materiais como o próprio processo de fabrico e a engenharia do produto e do processo, apoiada na integração da estrutura organizacional com o sistema de fabrico e o sistema de informação, que depende de aspectos organizacionais e de factores humanos.

A integração do fluxo de materiais e do fluxo de informação no seio de uma empresa consiste, por um lado, em implantar uma infra-estrutura de suporte à comunicação e à partilha de informação entre os recursos de produção e de gestão, baseada em redes de computadores e bases de dados – integração física. Por outro lado, é necessário garantir que os interlocutores intervenientes na comunicação o fazem de forma coerente e não ambígua,

utilizando protocolos e estruturas de informação comuns, ou seja, o mesmo contexto semântico – integração lógica.

A integração da estrutura organizacional com o sistema de fabrico e o sistema de informação passa pela adopção de novas metodologias emergentes, cujo objectivo é melhorar os diferentes aspectos de funcionamento de uma empresa. São exemplos dessas novas metodologias os conceitos Mesmo-a-Tempo (da sigla inglesa JIT – *Just-In-Time*) e Gestão da Qualidade Total (da sigla inglesa TQM – *Total-Quality-Management*). São exemplos de novos paradigmas organizacionais Grupos de Trabalho, Organizações Fractais, Produção Limpa (do inglês *lean-production*), Empresas Virtuais, etc.

1.1.2 A importância dos SAGVs no actual panorama de integração de sistemas de fabrico

A integração de uma empresa pode ser subdividida nas seguintes vertentes:

- o sistema de fabrico,
- o sistema de informação
- e a organização.

A utilização de SAGVs relaciona-se com a primeira vertente anteriormente enumerada, ou seja, a da integração do sistema de fabrico. A este nível, é dado ênfase a aspectos relacionados com a integração tecnológica e funcional de máquinas, equipamentos de fabrico, transportadores e AGVs, robôs e operadores humanos (responsáveis pelas operações de fabrico não automatizadas ou parcialmente automatizadas), que em conjunto são responsáveis pelas operações de fabrico e pela logística do em-curso-de-fabrico (habitualmente designado na literatura por WIP – *Work-In-Process*). Trata-se portanto da integração dos recursos que efectuam transformações sobre a matéria-prima e componentes, que são responsáveis pelo fluxo de materiais na fábrica e que produzem produto acabado ou componentes.

Um SAGVs interliga habitualmente os vários subsistemas de um sistema de produção, tendo portanto um papel bastante activo na gestão do fluxo de materiais no seio do sistema de fabrico. Um SAGVs inadequado pode tornar um sistema flexível de fabrico bastante inflexível, pois afecta irremediavelmente a eficiência do fluxo de materiais no interior da fábrica, merecendo por isso ser convenientemente estudado. Graças à flexibilidade de movimentos ou de manobras, e à adaptabilidade que caracterizam um SAGVs, a sua adequação e correcta inserção num sistema de fabrico flexível, pode torná-lo mais flexível e responsivo [2]. Para além disso, a sua utilização permite aumentar o grau de automatização do sistema de fabrico, pois diminui a intervenção humana no processo produtivo. Este aspecto assume particular importância em ambientes de fabrico hostis para o ser humano, como por exemplo na indústria química ou em locais na proximidade de equipamentos, cujo funcionamento se torne perigoso para os seres humanos (máquinas de soldadura por arco, balancés de grande porte, robôs potentes e de grande velocidade, etc).

Assim, os SAGVs assumem um papel preponderante em sistemas de fabrico onde o grau de automatização e o nível de integração sejam elevados. Nesse contexto é responsabilidade do SAGVs garantir a movimentação dentro dos prazos pretendidos da matéria-prima e dos componentes entre as estações de trabalho, e entre estas e os armazéns de matéria-prima e de produto acabado, procurando contribuir para uma boa produtividade

global do sistema de fabrico, através de um correcto escalonamento das operações de movimentação e de uma boa fiabilidade dos próprios AGVs. O objectivo essencial do SAGVs é, em última análise, contribuir para uma optimização do fluxo de materiais no interior da fábrica, merecendo por isso bastante atenção por parte de quem conceba e implemente um sistema flexível de fabrico.

1.1.3 Dificuldades inerentes à concepção de SAGVs

Na concepção de SAGVs são vários os problemas para os quais é necessário dar solução, procurando sempre garantir a satisfação dos requisitos colocados pelo sistema de fabrico onde se encontre inserido.

Nesta secção esses problemas são apenas enumerados, ficando adiada para um capítulo subsequente a sua descrição e análise detalhada e a apresentação de soluções para a sua resolução.

Os principais problemas ou dificuldades inerentes à concepção de SAGVs são [4]:

- a definição do *layout* das pistas ou trajectórias dos AGVs,
- a definição do número de veículos necessários,
- a gestão de tráfego e a política de reserva e de libertação de recursos (troços),
- a detecção e a resolução de *deadlocks*,
- o encaminhamento dos AGVs desde a origem até ao destino e o planeamento de trajectórias e
- as regras de despacho dos AGVs a utilizar.

1.2 Apresentação dos objectivos e da metodologia seguida na realização do trabalho

O trabalho desenvolvido, tem como principal objectivo apontar e, na medida do possível, resolver os problemas fundamentais que se levantam na concepção de SAGVs, que possam dar resposta aos requisitos colocados pelos actuais sistemas de fabrico flexíveis.

Para a maior parte desses problemas o tratamento matemático ou analítico revela-se inviável. A multiplicidade de AGVs, trajectos, lotes e sequências de processamento torna o comportamento do sistema muito complexo. Para além disso, o seu comportamento é condicionado em grande medida por informações de carácter estocástico, tais como: tempo de carga de uma peça num AGV, tempo de maquinação de uma peça numa máquina, taxa de avarias de uma máquina ou de um AGV, tempo de bloqueio de um AGV devido a condicionalismos de gestão de tráfego, etc.

Assim, é feita a validação e a avaliação das soluções encontradas, recorrendo-se à simulação em computador de cada caso em estudo [4]. Por essa razão, o trabalho desenvolvido dá particular ênfase à utilização de Redes de Petri (RP), como ferramenta de modelização e análise de sistemas discretos, de que os sistemas de fabrico são um bom exemplo. Embora não sendo o tema central deste trabalho, as RP são abordadas com algum detalhe, uma vez que foram uma ferramenta indispensável à prossecução do objectivo

principal, que é o estudo de SAGVs, com base em simulações em computador de um modelo baseado em RP.

De seguida será descrita a metodologia utilizada ao longo da realização do trabalho que serve de base à escrita desta dissertação.

Numa primeira análise, procurou-se dissecar os aspectos relevantes na concepção de SAGVs, estudando os resultados da investigação já realizada nesta área do conhecimento, com o objectivo de resolver os problemas enumerados em 1.1.3.

De seguida, definiu-se uma arquitectura de um Sistema de Gestão de AGVs (designado a partir de agora pela sigla SGAGVs), suficientemente geral e próxima da realidade, de forma a servir de plataforma para um modelo de um sistema de fabrico suportado num SAGVs.

Estabelecida a arquitectura do SGAGVs e utilizando como ferramenta de análise o pacote de *software Artifex*, que suporta Redes de Petri estocásticas e coloridas, construiu-se um modelo baseado numa RP que modelizasse o comportamento de um SGAGVs, com um nível de detalhe suficientemente apurado, de forma a permitir abarcar todos os problemas que se levantam na concepção de um SAGVs real. Construiu-se também um modelo de uma fábrica exemplo onde pudesse ser inserido o modelo do SGAGVs, permitindo assim analisar e simular o sistema flexível de fabrico suportado num SAGVs na sua globalidade.

Realizaram-se várias simulações com o mesmo sistema de fabrico, utilizando diferentes regras de despacho para os AGVs. Analisaram-se os resultados obtidos e procurou-se rentabilizar o esforço investido na realização das simulações, tirando conclusões baseadas na comparação do desempenho do sistema nos vários cenários simulados.

1.3 Roteiro da dissertação

Tendo sido feito o enquadramento do trabalho que é objecto desta dissertação, segue-se um breve roteiro pelos vários capítulos que se seguem a esta introdução.

O capítulo 2 faz uma análise sistemática de um SAGVs, caracterizando o AGV como elemento de automação, descrevendo e especificando uma arquitectura típica de um SAGVs e enumerando índices ou medidas quantitativas de avaliação do desempenho de um SAGVs. O capítulo 3 faz um estudo sistemático dos vários problemas inerentes ao funcionamento de SAGVs, definindo-os e fazendo uma análise comparativa das soluções actualmente disponíveis para a sua resolução. O capítulo 4 apresenta a teoria subjacente às Redes de Petri autónomas. O capítulo 5 apresenta as principais extensões às RP autónomas, que as tornam ferramentas úteis na modelização e simulação de sistemas de fabrico. O capítulo 6 apresenta algumas ferramentas de *software* de suporte à utilização das RP. O capítulo 7 apresenta a especificação de um modelo de um SAGVs. O capítulo 8 descreve detalhadamente a construção de um modelo da gestão de um SAGVs, baseado em RP estocásticas e coloridas, utilizando o *software Artifex*. O capítulo 9 descreve a construção de um modelo Artifex de um sistema de fabrico, suportado num SAGVs. O capítulo 10 descreve a metodologia e os objectivos das simulações efectuadas, com o modelo global do SAGVs construído no Artifex, apresentando e analisando os resultados obtidos. O capítulo 11 apresenta as principais conclusões resultantes do trabalho desenvolvido.

2. *Funcionalidades de um SAGVs e avaliação do seu desempenho*

Um SAGVs tem a seu a cargo a função de garantir um eficiente fluxo de materiais no interior do sistema de fabrico. Os modernos sistemas de fabrico têm de ser flexíveis e têm de permitir a reconfiguração dinâmica do sistema. O SAGVs é um componente chave para se atingirem os objectivos de um fabrico flexível, dinâmico e ágil [5]. Isto significa que se espera que o SAGVs possa fornecer os materiais requeridos à estação de trabalho apropriada, no tempo certo e na quantidade certa. Se estas condições não se verificarem o desempenho do sistema de fabrico na sua globalidade degradar-se-á , tornando-se menos eficiente, gerando menos lucro ou operando com custos mais elevados.

Os sistemas de fabrico actuais têm de ser flexíveis, pois estão sujeitos a uma grande dinâmica no que diz respeito à variedade de produtos produzidos e ao ciclo de vida dos mesmos. Assim, o SAGVs tem de ser suficientemente flexível para se adaptar às mutações sofridas pelo sistema de fabrico, nomeadamente novos planos de produção, novos caminhos tecnológicos para os materiais, diferentes cadências nos pontos de carga/descarga das estações de trabalho, etc. O SAGVs terá de contribuir para a flexibilidade do sistema de fabrico na sua globalidade, facilitando a redefinição do processo produtivo.

Este capítulo faz uma análise sistemática de um SAGVs, caracterizando o AGV como elemento de automação, especificando uma arquitectura típica de um SAGVs e enumerando índices ou medidas quantitativas de avaliação do desempenho de um SAGVs.

2.1 O AGV como dispositivo de automação

Um AGV é um Veículo Guiado Automaticamente (da sigla inglesa *Automatic Guided Vehicle*).

Trata-se de um dispositivo móvel utilizado no transporte automático de materiais em ambientes fabris, concebido para receber e executar instruções, seguir um caminho ou trajectória, e aceitar e distribuir materiais. As instruções para um AGV indicam para onde o veículo se deve dirigir, como deve chegar ao destino e o que deve fazer quando chegar ao destino [5].

Adicionalmente, o AGV é capaz de identificar a sua localização e de comunicar essa informação, tomando como referência as secções ou troços em que se encontram divididas as pistas implantadas no seu *layout* de trabalho, permitindo dessa forma o mapeamento global da posição dos elementos da frota do SAGVs.

A Figura 2.1 mostra um exemplo de um AGV de pequeno porte, equipado com dois tapetes de rolos para a carga/descarga de pequenas paletes, contendo peças metálicas processadas pela célula flexível de fabrico existente no CCP – Centro de CIM do Porto.

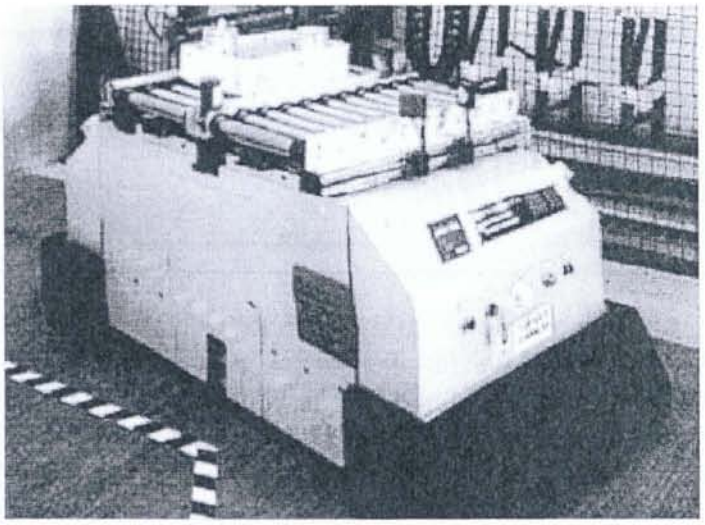


Figura 2.1 – O AGV do CCP – Centro de CIM do Porto – com fios de seguimento implantados no solo.

O AGV em causa foi produzido pela empresa portuguesa EFACEC, Automação e Robótica, S.A. É um AGV bidireccional, dotado de duas rodas direccionais (com um motor de direcção), uma à frente e outra atrás, tendo uma delas um motor responsável pela tracção do veículo. De ambos os lados, e mais ao centro do veículo, existem castores ("rodas loucas") que garantem o equilíbrio e a estabilidade do veículo.

A fonte de alimentação é uma bateria de tracção de 48 V de níquel-cádmio, estando preparada para permitir um funcionamento do veículo durante 24 horas por dia, pois suporta regimes de carga com uma corrente eléctrica 4 a 5 vezes superior à corrente nominal, ou seja, tempos de carga bastante reduzidos.

Os motores de direcção, de tracção e de accionamento dos tapetes de rolos são motores de corrente contínua, de imanes permanentes. A utilização habitual de motores de corrente contínua nos AGVs justifica-se pelo facto dos controladores de posição e de velocidade desses motores serem mais simples e mais baratos do que os controladores de motores de corrente alternada (de indução ou síncronos), e pelo facto da fonte de alimentação disponível ser uma fonte de tensão contínua.

O sistema de navegação baseia-se no seguimento do campo magnético criado por condutores implantados no solo e percorridos por uma corrente eléctrica sinusoidal. O campo magnético é detectado por duas antenas, uma colocada à frente e outra atrás. Este sistema de navegação é o mais comum nos AGVs actualmente comercializados. Esses AGVs são tecnicamente designados por AGVs filoguiados.

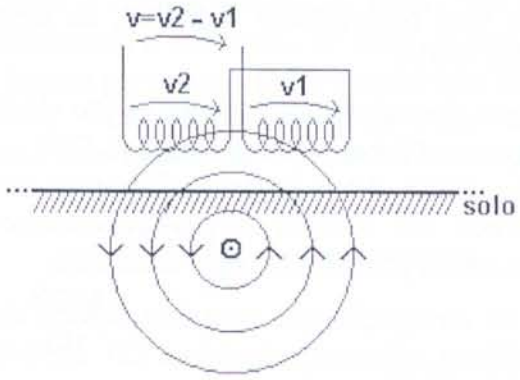


Figura 2.2 – Princípio de funcionamento dos AGVs filoguiados.

A Figura 2.2 ilustra o princípio de funcionamento do AGV filoguiado. A antena de seguimento é constituída por dois solenóides, colocados um de cada lado. O campo magnético sinusoidal criado pela corrente eléctrica sinusoidal que percorre o condutor embebido no solo induz uma força electromotriz (f.e.m.) também sinusoidal em cada solenóide, tanto mais intensa, quantas mais linhas de força atravessarem o seu núcleo.

A f.e.m. induzida no solenóide é máxima quando o seu ponto médio se encontrar alinhado com o condutor. Como os solenóides da antena estão associados em anti-série, geram uma tensão diferencial v, que se anula quando a antena se encontra centrada com a trajectória definida pelo condutor eléctrico existente no solo. Assim, o sistema de controlo de posição em malha fechada de um AGV filoguiado garante o seguimento das trajectórias, procurando anular em cada momento a tensão diferencial v.

Este sistema de navegação é bastante fiável. Para além disso, tem a vantagem de permitir a utilização dos condutores existentes no solo para a comunicação entre o sistema de gestão da frota dos AGVs, (computador central) e os AGVs. Para isso, é injectada uma portadora de alta frequência modulada pelo sinal eléctrico a transmitir aos AGVs. Nesta comunicação é utilizado habitualmente um protocolo *master-slave*, sendo o computador o *master* e o veículo o *slave*. Quando o *master* pretende comunicar com um determinado veículo, envia uma mensagem de *polling* a esse veículo. Após a resposta do veículo, estabelece-se a comunicação e o *master* começa a enviar-lhe informação.

Devido a dificuldades técnicas (ambientes industriais bastante poluentes do ponto de vista electromagnético), ou porque o sistema de navegação é diferente do filoguiado, muitas vezes não é utilizado esse canal de comunicação. Nesses casos são habitualmente utilizados emissores e receptores de rádio na comunicação entre o computador central e os veículo da frota de AGVs.

Um SAGVs filoguiado tem a desvantagem de não permitir a reconfiguração do *layout* de trabalho dos AGVs, o que o torna um sistema muito pouco flexível e inadequado a indústrias do tipo *job shop*, onde existe a necessidade de reconfigurar frequentemente o *layout* da fábrica.

No sentido de resolver esta limitação dos sistemas filoguiados, foram concebidos outros sistemas mais flexíveis. Contudo estes são quase sempre menos fiáveis, devido à menor robustez inerente ao princípio de funcionamento utilizado ou à maior sofisticação das tecnologias utilizadas.

São exemplos desses sistemas de navegação os sistemas baseados na colocação de uma fita reflectora ou com propriedades químicas, ao longo das pistas que definem as trajectórias do *layout* dos AGVs.

Mais recentemente foram objecto de investigação sistemas de navegação baseados no Sistema de Posicionamento Global (da sigla inglesa GPS – *General Positioning System*), utilizando comunicação satélite. Estes sistemas afiguram-se interessantes do ponto de vista da flexibilidade que oferece um sistema de posicionamento global, contudo apresentam limitações relacionadas com a precisão do posicionamento.

Outro sistema de navegação que tem sido objecto de investigação são os sistemas baseados em visão artificial, utilizando câmaras CCD. Devido à sofisticação das tecnologias envolvidas (reconhecimento de padrões, processamento de imagens ou de grandes volumes de informação, etc), são sistemas cuja possibilidade de utilização é ainda muito limitada.

Um outro sistema de navegação que começa a ser competitivo no mercado é o sistema baseado num emissor de raios *laser*. Neste sistema o AGV é equipado com um emissor de luz *laser* que efectua periodicamente um varrimento de 180° do espaço físico envolvente. Em cada ponto do *layout*, o veículo ao efectuar o varrimento detecta a reflexão de luz vinda de três pontos na vizinhança, nos quais existem pedaços de material reflector. Através dessa triangulação e da intensidade da luz reflectida e recebida pelo veículo, este consegue determinar a sua posição. A reconfiguração do *layout* de um SAGVs do tipo *laser* consiste na alteração do posicionamento das placas reflectoras colocadas nas paredes ou noutros obstáculos envolventes. Muito provavelmente, a médio ou longo prazo este sistema de navegação irá ocupar o lugar dos actuais sistemas filoguiados, que actualmente ainda são os mais utilizados.

Um AGV é habitualmente dotado de dispositivos de segurança, tais como pára-choques, sensores ultra-sónicos, lâmpadas de aviso e botões de emergência. Estes têm por finalidade proteger o AGV e as pessoas ou equipamentos existentes nas proximidades da sua área de trabalho.

2.2 Arquitectura de um SAGVs em 3 camadas

A Figura 2.3 mostra um diagrama de blocos de uma arquitectura de 3 camadas de um SAGVs. Esta arquitectura pressupõe que os AGVs são do tipo filoguiado. Este pressuposto estará implícito deste ponto em diante e nos capítulos subsequentes, sempre que o texto refira um SAGVs. As 3 camadas numeradas entre parêntesis são:

- SPAGV – Sistema de Posicionamento de AGVs (camada 1);
- SGAGV – Sistema de Gestão de AGVs (camada 2);
- SGM – Sistema Geral de Movimentação (camada 3).

O bloco "Fábrica" representa o sistema de fabrico em que o SAGVs se encontra inserido.

As setas indicadas na figura ilustram o sentido das trocas de informação que podem ocorrer entre os vários blocos do sistema. Junto a cada seta são enumeradas as mensagens mais habituais trocadas entre os blocos respectivos, no sentido indicado pela seta.

Esta arquitectura serviu de base à realização do trabalho que será analisado em capítulos mais à frente. Como o tema central do trabalho é o SAGVs propriamente dito,

analisou-se com menos detalhe o comportamento do bloco "Fábrica", por uma questão de simplificação. Por isso, desprezou-se a existência de erros durante as operações de carga ou de descarga de AGVs, que fossem provocados pelos dispositivos físicos existentes nos pontos de carga/descarga da fábrica. Uma análise mais complexa do comportamento desses dispositivos seria inútil, pois o tema central do trabalho é o estudo do SAGVs.

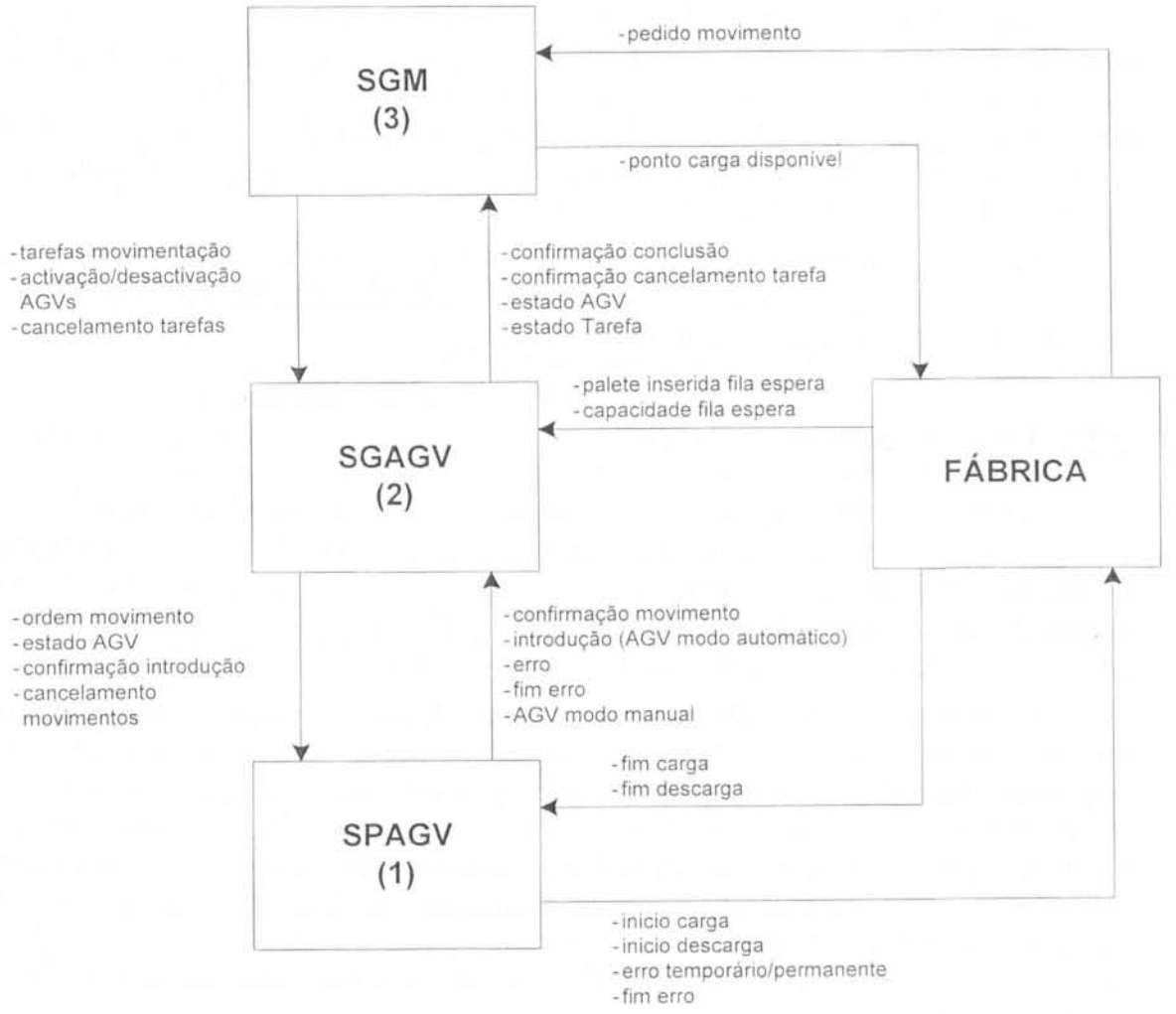


Figura 2.3 – Diagrama de blocos da arquitectura de 3 camadas de um SAGVs.

2.2.1 Especificação do SPAGV

A camada designada por SPAGV é a de mais baixo nível, ou seja, a que interage directamente com os AGVs. A este nível, as ordens relativas a movimentos elementares, emitidas pela camada superior, são mapeadas em instruções da linguagem de programação do AGV. Essas instruções são comunicadas ao veículo remotamente, utilizando o canal de comunicação existente entre o computador onde se encontra a correr a aplicação informática, que implementa o SPAGV, e os AGVs.

O SPAGV recebe informação relativa ao estado de cada AGV (diagnósticos de erros, o troço ou a posição onde se encontra, o estado de execução de um movimento, etc), informando a camada superior.

É também o SPAGV que gere a introdução de AGVs no sistema. A expressão "introdução do AGV" pretende designar a passagem do AGV do modo manual para o modo automático.

Um AGV em modo manual é como se não existisse para o resto do sistema de gestão de AGVs. Se estiver no estado activo, é autorizada a sua introdução por parte do SPAGV. Se estiver no estado inactivo, não é autorizada a sua introdução. Um AGV em modo automático pode receber ordens do sistema de AGVs para a execução de movimentos elementares (deslocar-se através de um troço, efectuar uma carga ou efectuar uma descarga), sendo conhecida a sua posição e o espaço físico que ocupa.

Cabe também ao SPAGV sincronizar o AGV com o outro dispositivo da fábrica interveniente numa operação de carga ou de descarga de materiais no AGV (tapete de rolos, mesa de transferência, tapete rolante, etc).

2.2.2 Especificação do SGAGV

A camada designada por SGAGV é a mais complexa de todas, pois tem a seu cargo as tarefas de processamento mais pesadas na gestão do SAGVs. Recebe da camada superior tarefas de movimentação e ordens de activação ou de desactivação dos AGVs, bem como ordens de cancelamento de tarefas pendentes. Notifica a camada superior do estado dos AGVs, do estado das tarefas pendentes e do cancelamento de tarefas.

Com base na informação relativa às tarefas de movimentação existentes em fila de espera ou em execução e na informação relativa ao estado dos AGVs existentes, o SGAGV é responsável pela correcta afectação das tarefas pendentes aos AGVs, procurando optimizar um critério ou um conjunto de critérios escolhidos para o controlo do sistema. Exemplos de critérios são a minimização do custo de execução das tarefas pendentes, expresso em tempo, distância percorrida ou energia consumida, a minimização do tempo de atendimento ou de permanência em fila de espera das tarefas, a minimização do comprimento das filas de espera nos pontos de carga da fábrica, etc. Esta função será designada daqui em diante por escalonamento.

O SGAGV também é responsável pela escolha do melhor trajecto para um AGV se deslocar entre dois pontos da fábrica. Esta função será designada daqui em diante por encaminhamento (do termo inglês *routing*).

Com base no trajecto escolhido para um determinado AGV se deslocar entre dois pontos do *layout*, o SGAGV decompõe esse movimento em movimentos mais elementares (movimentos através de troços), que resultam em ordens elementares enviadas à camada inferior, ou seja, ao SPAGV. O despacho dessas ordens de movimento elementares é condicionado pela gestão de tráfego, que consiste em evitar que os AGVs possam chocar quando num dado momento necessitem de ocupar o mesmo espaço físico para executar os seus próximos movimentos elementares. Esta função será designada daqui em diante por gestão de tráfego.

Cabe também ao SGAGV resolver situações de bloqueio de AGVs anómalas, devidas a situações indesejáveis tais como: *deadlocks*; um AGV livre ocupando um espaço físico

requerido por um ou mais AGVs a executarem tarefas, tendo o SGAGV de mover o AGV livre para um espaço vizinho que permita aos outros AGVs prosseguirem a execução das tarefas; um AGV num estado erróneo, ocupando um espaço físico requerido por um ou mais AGVs a executarem tarefas, tendo o SGAGV de replanear os trajectos destes AGVs, de forma a tentar desbloqueá-los. Esta função será designada daqui em diante por resolução de conflitos.

A informação recebida da fábrica sobre o estado das filas torna-se indispensável ao funcionamento do SGAGV, no caso dos critérios de optimização utilizados pelo escalonador dependerem do comprimento das filas de espera nos pontos de carga da fábrica. Como a arquitectura pretende ser o mais genérica possível, esse tipo de informação é representada na Figura 2.3 por uma seta entre os blocos Fábrica e SGAGV.

2.2.3 Especificação do SGM

A camada designada por SGM é a de mais alto nível. Cabe ao SGM estabelecer a interface entre o planeamento de produção da fábrica que gera pedidos de movimentação e o SAGVs. Por isso, é responsável pelo envio de ordens de movimentação e de cancelamento de tarefas pendentes à camada inferior, pelo acompanhamento do estado das tarefas pendentes e dos AGVs, permitindo a sua monitorização, e pelo *setup* do sistema (activação e desactivação de AGVs)

Esta camada pode ainda realizar o escalonamento de tarefas especiais – posicionamentos e tarefas só de carga ou só de descarga – fazendo um *bypass* ao escalonador da camada inferior. Dada a complexidade do escalonador existente no SGAGV, é difícil ou até impossível garantir que ele se comportará de forma satisfatória face a todas as exigências colocadas pelos planos de produção que se pretendam executar, nomeadamente garantir tempos de atendimento muito curtos para tarefas com carácter urgente. O escalonador do SGAGV é configurado para responder satisfatoriamente nos cenários mais prováveis da fábrica onde o SAGVs se encontre inserido. Isto justifica a importância da existência da possibilidade do SGM ter algum poder de escalonamento em cenários menos habituais, onde o escalonador do SGAGV não garanta os requisitos pretendidos.

2.3 Medidas quantitativas do desempenho de um SAGVs

Da análise efectuada sobre as funcionalidades de um SAGVs é fácil perceber que o seu comportamento é deveras complexo. Levando em linha de conta que o SAGVs não é um sistema isolado e que se encontra embebido no seio de um sistema de fabrico, o seu comportamento é em grande medida condicionado pelo comportamento global do sistema de fabrico, tornado a sua análise ainda mais complexa e assustadora.

Perante a complexidade do sistema, a avaliação do desempenho, pela via analítica, de qualquer SAGVs relativamente simples torna-se inviável. Assim, a avaliação do desempenho de um SAGVs durante a fase da sua concepção normalmente só pode ser feita recorrendo à realização de simulações do sistema de fabrico em que esteja inserido [4].

Em face dos resultados das simulações, é necessário extrair desses resultados medidas quantitativas suficientemente condensadas e objectivas, para ser possível obter conclusões relativas ao assunto que seja objecto de análise. O comportamento e o desempenho de um

sistema de fabrico, e do SAGVs em particular, pode ser analisado segundo vários pontos de vista.

Concentrando a análise nas tarefas de movimentação realizadas pelo SAGVs num determinado intervalo de tempo, pode-se falar nos seguintes índices de desempenho:

- tempo médio de permanência de uma tarefa em fila de espera;
- tempo médio de execução de uma tarefa;
- número de tarefas executadas;
- tempo médio gasto por um AGV na execução de cada fase da tarefa (viagem até ao ponto de carga, carga, viagem até ao ponto de descarga e descarga);
- ...

Concentrando a análise nos AGVs durante um determinado intervalo de tempo, pode-se falar nos seguintes índices de desempenho (valores médios para um AGV):

- taxa de utilização;
- tempo gasto em cada um dos estados possíveis;
- número de erros;
- distância percorrida carregado e descarregado;
- número de arranques carregado e descarregado;
- número de operações de carga;
- número de operações de descarga;
- ...

Concentrando a análise no comportamento de uma estação de trabalho da fábrica durante um determinado intervalo de tempo, pode-se falar nos seguintes índices de desempenho:

- número médio de paletes na fila de espera de entrada e de saída;
- comprimento máximo atingido nas filas de espera de entrada e de saída;
- número de paletes recebidas;
- número de paletes processadas;
- número de paletes distribuídas;
- tempo médio de permanência de uma palete na fila de entrada e de saída;
- valor médio do em-curso-de-fabrico (WIP);
- ...

Concentrando a análise no comportamento do controlador do SAGVs durante um determinado intervalo de tempo, pode-se falar nos seguintes índices de desempenho:

- número de chamadas ao escalonador;
- número de conflitos com AGVs parados resolvidos;
- número de conflitos com AGVs no estado erro resolvidos;
- número de *deadlocks* resolvidos;

- ...

Todas estas medidas quantitativas são relevantes na análise do desempenho de um sistema de AGVs, porque qualquer alteração numa das duas funcionalidades reflectir-se-á em todas elas, obviamente mais numas do que noutras.

Ao contrário do que acontece noutros sistemas de engenharia (e.g. circuitos electrónicos, controladores, aparelhos de alta fidelidade, etc) em que existem critérios de avaliação do desempenho universalmente aceites (diagramas de Bode, largura de banda, factor de distorção harmónica, relação sinal/ruído, etc), nos sistemas de fabrico, e particularmente nos SAGVs, não existe um conjunto de critérios de avaliação aplicável em todos os casos (uma métrica universalmente aceite). Neste tipo de sistemas é inevitável uma análise caso a caso, e fica ao critério do engenheiro de sistemas de fabrico a escolha criteriosa das medidas quantitativas e qualitativas que melhor classifiquem o desempenho do sistema em estudo.

3. *Concepção de um SAGVs*

A concepção de um SAGVs requer a tomada de decisão sobre as melhores estratégias para a resolução dos vários problemas associados ao seu funcionamento (ver 1.1.3). Com base nos requisitos impostos pelo sistema de fabrico, do qual o SAGVs será parte integrante, é necessário escolher as soluções que se adaptem melhor às exigências existentes. Porque não existe um SAGVs ideal, que satisfaça as necessidades de todos os tipos de aplicações, é necessário pensar numa solução de compromisso, dando prioridade a aspectos do seu funcionamento que mais influenciem o desempenho final do sistema, sob pena de ver penalizado o seu desempenho noutros aspectos que não sejam tão relevantes para o sistema de fabrico em causa.

Este capítulo faz um estudo sistemático dos vários problemas inerentes ao funcionamento de SAGVs, definindo-os e fazendo uma análise comparativa das soluções actualmente disponíveis para a sua resolução. O conhecimento das soluções existentes para cada um dos problemas é crucial para se conseguir decidir sobre a melhor configuração do SAGVs requerido por uma determinada aplicação.

Como foi referido em 2.2, ao longo do texto pressupõe-se que os SAGVs referidos são do tipo filoguiado, uma vez que são os mais utilizados actualmente. No entanto, a maior parte dos aspectos estudados são comuns a todos os tipos de sistemas, ou pelo menos, facilmente extensíveis a SAGVs com outros tipo de sistemas de navegação.

3.1 Layout das trajectórias

Um problema que se levanta logo à partida, prende-se com a definição do *layout* das trajectórias ou das pistas dos AGVs. No caso de AGVs filoguiados, trata-se de definir o plano de implantação dos rasgos, onde serão instalados os *loops* de corrente, utilizados no seguimento das trajectórias.

O projecto do *layout* das trajectórias é um aspecto de projecto extremamente importante, porque influencia significativamente o desempenho do SAGVs, merecendo por isso muita atenção na fase do projecto do sistema [4]. O *layout* tem um impacto directo na eficiência das decisões que digam respeito ao encaminhamento e ao escalonamento dos AGVs [17].

O projecto é condicionado pelo *layout* da fábrica, pois tem de assegurar que todos os pontos de carga/descarga fazem parte da rede de trajectórias a definir, e de garantir a conectividade entre todos esses pontos. No desenho das trajectórias é necessário manipular informação geométrica do *layout* da fábrica, para se impedir que os AGVs possam chocar com obstáculos, ao efectuarem alguma das trajectórias desenhadas, e para se fazer uma utilização conveniente e racional do espaço físico disponível.

Um aspecto que melhora fortemente a robustez e a tolerância a falhas do SAGVs é a existência de caminhos alternativos. Nesse sentido, deve-se ter especial atenção em áreas do *layout* onde se preveja a existência de uma densidade de tráfego elevada, e perto dos pontos de carga/descarga mais visitados (com maiores cadências). Estas zonas são aquelas onde a

não existência de caminhos alternativos pode tornar o sistema altamente ineficiente, ou até incapaz de satisfazer pedidos de movimentação. A localização dessas zonas terá de ser baseada na avaliação dos caminhos tecnológicos mais prováveis para as peças movimentadas pelos AGVs, e na análise do fluxo de materiais no interior da fábrica.

A criação de caminhos alternativos, permite evitar que a ocorrência de uma falha num AGV, e o consequente bloqueio do espaço físico por si ocupado, possa bloquear permanentemente o acesso a um ponto de carga/descarga ou a uma área do *layout*. A criação de caminhos alternativos em áreas de grande tráfego, permite descongestionar essas áreas e diminuir a probabilidade de ocorrência de *deadlocks*.

Para além destes aspectos funcionais, é necessário garantir uma boa optimização das trajectórias escolhidas. Esta visa tornar o tráfego da rede de AGVs o mais fluido possível, contribuindo de forma positiva para o desempenho do SAGVs. Nesse sentido, deverão ser tomados os seguintes cuidados:

- evitar que pontos de carga/descarga sejam simultaneamente pontos de passagem no acesso a outros pontos de carga/descarga, de modo a minimizar os tempos de espera dos AGVs, durante a execução de trajectos para aceder a esses pontos;
- evitar que a execução de uma trajectória por um AGV seja condicionada pela execução de uma trajectória vizinha por outro AGV, para se evitar o choque entre os veículos.

Existem basicamente dois tipos de *layouts*: unidireccionais e bidireccionais [4].

O *layout* unidireccional é geralmente preferido face ao bidireccional, porque facilita muito o controlo do SAGVs, por tornar menos frequente a ocorrência de conflitos entre AGVs e a ocorrência de situações de congestionamento.

O *layout* bididireccional tem a vantagem de requerer menos veículos e de permitir um melhoramento do desempenho do sistema face ao *layout* unidireccional [6], porque aumenta muito a flexibilidade de movimentos dos AGVs. Para além disso, permite normalmente uma configuração mais compacta para os sistemas flexíveis de fabrico, a nível do espaço físico ocupado, já que com menos pistas consegue servir todos os pontos do *layout*.

Estudos realizados no passado, onde se comparou o desempenho de instalações de AGVs unidireccionais e bidireccionais, demonstraram através de simulação que os sistemas bidireccionais dominam os sistemas unidireccionais, nomeadamente na taxa de utilização dos veículos [7].

3.1.1 Modelização de um layout de trajectórias

Nesta secção será apresentado um modelo para o *layout* de trajectórias de um SAGVs.

Nos algoritmos de Investigação Operacional relativos a problemas de redes (determinação do caminho mais curto, determinação da árvore de valor mínimo, determinação do fluxo máximo), as redes são habitualmente modelizadas por um grafo.

Um grafo G=(N, A) é constituído por dois conjuntos finitos: um conjunto N de nós e um conjunto A de arcos, que interligam os nós. Os arcos são direccionais, porque definem um sentido do nó origem para o nó destino. É possível haver dois arcos ligando os mesmos dois nós, desde que tenham sentidos opostos.

Dividindo as linhas, que definem o *layout* de um SAGVs, em segmentos, pode-se modelizar o *layout* à custa de um grafo como o da Figura 3.1.

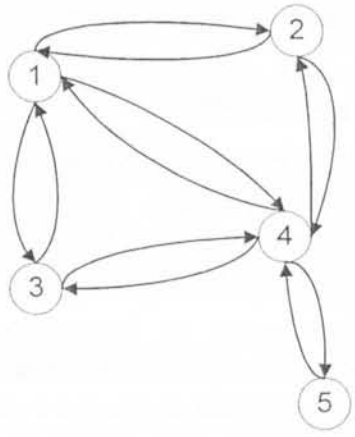


Figura 3.1 – Exemplo de um grafo representativo de um layout de trajectórias.

Cada segmento é representado por um nó. Assim, um nó representa uma posição no *layout*, à qual está associada uma área de ocupação, onde em cada momento só pode estar no máximo um AGV.

Os arcos representam a conectividade existente entre os nós da rede, ou seja, entre os segmentos do *layout*. Os arcos que saem de um nó representam os segmentos para os quais um AGV posicionado nesse nó se pode dirigir. Os arcos que se dirigem para o nó, representam os segmentos donde um AGV posicionado nesse nó pode ser proveniente.

Um grafo como o da Figura 3.1 pode ter duas representações computacionais: pode ser representado por uma matriz (Tabela 3.1) ou por uma lista (Tabela 3.2).

Tabela 3.1 – Matriz representativa do grafo da Figura 3.1.

		Para o nó				
Do		1	2	3	4	5
nó	1	0	1	1	0	0
	2	1	0	0	1	0
	3	1	0	0	1	0
	4	0	1	1	0	1
	5	0	0	0	1	0

Tabela 3.2 – Lista representativa do grafo da Figura 3.1.

Nós	Próximos Nós
1	2,3,4
2	1,4
3	1,4
4	1,2,3,5
5	4

A representação de um grafo através de uma matriz não é eficiente. As listas têm a vantagem de consumirem muito menos espaço de memória e de serem de mais fácil manipulação. Para além disso, permitem outras formas de manipulação (ordenação, classificação, etc). Por exemplo, ao seguir-se um trajecto, pode-se percorrer facilmente a lista em ambos os sentidos, em vez de se ter de procurar numa grande matriz um único 1.

Geralmente, as listas são a representação mais conveniente quando os sistemas apresentam as seguintes características [9]:

- Requisitos de armazenamento de informação imprevisíveis. O espaço de memória exacto da informação requerida por um programa depende da informação particular a ser processada. Num SAGVs não se sabe à partida se o grafo representativo de um *layout* terá uma centena ou milhar de nós.
- Manipulação da informação de forma intensiva. Num SAGVs é necessário percorrer a lista, classificá-la, procurar um nó, apagar ou mudar um elemento, centenas de vezes por segundo.

3.2 Número mínimo de veículos necessários

Na fase de concepção de um SAGVs é necessário determinar o número óptimo de veículos requeridos pelo sistema. Este número depende de diversos factores, nomeadamente o tempo de viagem, o tempo livre, o tempo de carga/descarga, o tempo de carga de baterias, a velocidade média dos AGVs e também o *layout* e os tempo de processamento dos produtos [4].

Num sistema de fabrico constituído por vários centros de maquinação, desempenhando funções diversas, uma peça típica visita muitos centros antes da sua produção ser completada. Quando uma peça está a ser processada, as necessidades de movimentação por parte do SAGVs são faseadas no tempo. À medida que o número de peças diferentes produzidas pela fábrica aumenta, o problema de determinação do número de veículos necessários torna-se intratável, e tem de ser analisado através de simulações do sistema num computador.

A estimação do número de veículos necessários, não entrando com factores de ordem temporal, reduz a complexidade do problema. Tal estimativa fornece um bom ponto de partida para a análise do problema através da simulação em computador [10].

Para se determinar o número de veículos requeridos, é necessário ter acesso a informação relativa aos lotes a serem processados. O tráfego no SAGVs é influenciado pelos seguintes factores: lotes que são fabricados simultaneamente, tamanho dos lotes, tempos de processamento e taxa de chegada. Devido à versatilidade das máquinas, os lotes podem ser processados em mais do que uma sequência. Isto permite o encaminhamento alternativo dos lotes em caso de falha numa máquina, ou por razões de balanceamento da carga de trabalho das várias máquinas. A probabilidade dos lotes usarem encaminhamentos alternativos tem de ser conhecida. Também é necessário conhecer os seguintes parâmetros do sistema de fabrico: o número de máquinas, o seu *layout*, a capacidade dos *buffers* local e central, o *layout* das pistas dos AGVs e o tempo de viagem entre os vários pontos de carga/descarga.

Em [10] são deduzidas expressões matemáticas que, com base na informação enumerada anteriormente, permitem estimar o número de veículos necessários.

O sistema de fabrico é modelizado através das grandezas indicada na Tabela 3.3.

Tabela 3.3 – Grandezas do modelo da fábrica para a estimação do número óptimo de AGVs.

n	número de peças processadas simultaneamente
v_i	tamanho do lote i
$NSEQ_i$	número de diferentes sequências pelas quais o lote i pode ser processado
S_{ij}	probabilidade do lote i ser processado segundo a sequência j
$NMAC_{ij}$	número de máquinas que o lote i tem de visitar para completar o processamento segundo a sequência j
PT_{ijk}	tempo de processamento do lote i, na máquina k, segundo a sequência j
t_{kl}	tempo de viagem entre duas máquinas k e l

Admitindo que a máquina m+1 é uma estação de carga/descarga, por onde entram e saiem todas as peças processadas na fábrica, o fluxo f_{kl}, entre duas quaisquer máquinas k e l, devido a um fluxo unitário na estação de carga/descarga, é dado pela Equação 3.1.

$$f_{kl} = \sum_{i=1}^{n} v_i \sum_{j=1}^{NSEQ_i} S_{ij} \partial_{kl} \qquad \forall k,l = 1,\ldots,m+1 \wedge k \neq l \tag{3.1}$$

O parâmetro δ_{kl} é definido da seguinte forma:

$$\partial_{kl} = \begin{cases} 1 & \text{se a maquina l for visitada imediatamente depois de} \\ & \text{ser visitada a maquina k na sequencia j do lote i} \\ 0 & \text{senao} \end{cases}$$

As propriedades da matriz de fluxos são representadas pelas Equações 3.2 a 3.5.

$$f_{kk} = 0 \qquad \forall k = 1,\ldots,m+1 \tag{3.2}$$

$$\sum_{k=1}^{m} f_{m+1,k} = 1 \tag{3.3}$$

$$\sum_{l=1}^{m} f_{l,m+1} = 1 \tag{3.4}$$

$$\sum_{l=1}^{m+1} f_{kl} = \sum_{l=1}^{m+1} f_{lk} \qquad \forall k = 1,...,m+1 \tag{3.5}$$

A Equação 3.2 mostra que não existe fluxo de uma máquina para ela própria. A Equação 3.3 assegura que os lotes só entram na fábrica através da estação de carga/descarga, e a Equação 3.4 assegura que os lotes só saem da fábrica através da estação de carga/descarga. A Equação 3.5 mostra a conservação do fluxo em todas as estações de trabalho.

A matriz de probabilidades de transição entre máquinas $P=[p_{kl}]$ é obtida a partir da matriz de fluxos F, através da sua normalização, de forma que a soma de qualquer linha seja 1.

Considerando um período de 8 h (480 min.), o número de cargas nl_k a serem removidas de qualquer máquina k para outras localizações, é dada pela Equação 3.6, onde μ_k é a taxa média de processamento de lotes da máquina k. Esta grandeza pode ser determinada pela Equação 3.7, onde a expressão entre parênteses rectos representa a probabilidade do lote i visitar a máquina k na sequência j.

$$nl_k = 480 / \mu_k \tag{3.6}$$

$$\mu_k = \sum_{i=1}^{n} \sum_{j=1}^{NSEQ_i} \sum_{l=1}^{NMAC_{ij}} \left[\frac{v_i S_{ij} \gamma_{k(ij)}}{\sum_{p=1}^{n} \sum_{q=1}^{NSEQ_p} v_p S_{pq} \gamma_{k(pq)}} \right] \times PT_{ijl} \qquad \forall k = 1,...,m \tag{3.7}$$

A taxa média de entrada dos lotes no sistema através da estação de carga/descarga pode ser calculada a partir do tempo médio que os lotes passam no interior do sistema. A soma dos produtos dos tempos totais de processamento dos lotes e das respectivas probabilidades é o tempo médio que passam no interior da fábrica.

Um lote que termine o processamento na máquina k, é transportado para outra máquina l com uma probabilidade p_{kl} e requer t_{kl} unidades de tempo de AGV. Para além disso, o AGV espera l_k unidades de tempo para carregar na máquina k e u_l unidades de tempo para descarregar na máquina l. Devido à capacidade limitada de armazenamento em frente às máquinas, os lotes são por vezes encaminhados para um *buffer* central (representado pelo índice b). Se α representar a proporção do tempo que isto ocorre, o total de tempo de AGV (TR) é dado pela Equação 3.8.

$$\begin{aligned} TR = {} & \alpha \left[\sum_{k=1}^{m+1} nl_k \times \left(\sum_{l=1}^{m+1} p_{kl} \left(t_{kb} + t_{bl} + l_k + u_b + l_b + u_l \right) \right) \right] + \\ & + (1-\alpha) \times \left[\sum_{k=1}^{m+1} nl_k \times \left(\sum_{l=1}^{m+1} p_{kl} \left(t_{kl} + l_k + u_l \right) \right) \right] \end{aligned} \tag{3.8}$$

Se TA representar o tempo total disponível por AGV, durante um período de 8 h, int(TR/TA) é o número de AGVs requeridos pelo sistema, onde int(x) representa o mais pequeno inteiro maior ou igual a x [10].

3.3 Gestão de tráfego

Se um SAGVs contiver apenas um AGV, a gestão de tráfego é simples. Quando há mais veículos a circularem no sistema, os problemas de interferência entre veículos e de colisões têm de ser evitados. Isto normalmente é feito criando zonas de exclusividade, para evitar colisões, e ramais em locais apropriados do *layout*, para resolver situações de conflito.

3.3.1 Possível modelo de operação para o movimento de um AGV

Tomando como referência o modelo do *layout* apresentado em 3.1.1, o modelo de operação para o movimento de um AGV aqui apresentado assume que em cada nó do *layout* existe um controlador para monitorar o seu estado, isto é, o seu grau de congestionamento [11].

O movimento de um veículo v até um nó n é atómico, isto é, pode efectuar-se na totalidade ou não se pode efectuar. Uma vez que cada nó pode admitir um número limitado de veículos, alguns deles poderão não conseguir deslocar-se até n, se for atingido esse limite. Basicamente, há duas abordagens possíveis para gerir situações de conflito como estas:

- Escalonamento – antes de v começar a executar o seu trajecto, é decidido quando e por que nós o veículo irá passar;
- Resolução dinâmica – cada veículo v tem de pedir para se deslocar até um nó n, sempre que necessite de o fazer; se o pedido for aceite, o veículo pode movimentar-se até n.

No método de resolução dinâmica, o pedido pode ser aceite ou rejeitado, dependendo do estado do nó e da estratégia de reserva e de libertação de nós. Para apresentar as estratégias possíveis, será assumido que cada nó pode admitir mais do que um veículo ao mesmo tempo, tornando assim a análise o mais geral possível. A sua capacidade será representada por cap(n) e o número de veículos, que um nó num dado momento possui, será representado por nv(n). Assim, a preposição $nv(n) \leq cap(n)$ é sempre verdadeira.

O esquema de bloqueio de um nó só pode ser o seguinte: se $cap(n)-nv(n) \geq 0$, nv(n) é incrementado e n é bloqueado, senão o pedido de bloqueio é rejeitado.

Para o esquema de libertação de nós poderão ser consideradas duas possibilidades: um esquema fechado ou um esquema aberto. No primeiro, só quando o veículo chega ao destino é que todos os nós são libertados. O principal objectivo é assegurar que um veículo pode ser capaz de voltar à posição inicial através do mesmo caminho. No esquema aberto, o veículo liberta n logo que chegue ao próximo nó do trajecto. Neste caso, o objectivo é bloquear o menor número possível de nós. Contudo não assegura que o veículo possa voltar à posição inicial através do mesmo caminho, sem requerer de novo o bloqueio dos nós.

O modelo utilizado neste trabalho é um modelo baseado no esquema de resolução dinâmica, com esquema de libertação de nós aberto e capacidade unitária para todos os nós. A resolução dinâmica foi preferida, porque o método de escalonamento tem a grande desvantagem de ser computacionalmente mais pesado, pois exige que se actualize o

escalonamento, sempre que o tempo gasto pelo veículo no movimento entre dois nós seja diferente do tempo inicialmente previsto. Se a escolha tivesse recaído sobre o esquema de libertação fechado, muitos dos nós estariam indisponíveis para a maior parte dos AGVs. Para além disso, a desvantagem de não haver possibilidade de se voltar atrás pelo mesmo caminho não é muito importante em *layouts* bidireccionais.

3.3.2 Regras de controlo utilizadas na gestão de tráfego

A gestão de tráfego tem de gerir a frota de AGVs, tomando como base as tarefas atribuídas, os trajectos escolhidos e o esquema de bloqueio e de libertação de nós utilizado. Na discussão que se segue, assume-se que as tarefas já estão atribuídas aos AGVs e que os trajectos já foram determinados. Estes dois assuntos serão tratados em secções posteriores deste capítulo.

Do ponto de vista do AGV, os nós são recursos partilhados no encaminhamento através do *layout*. Assim, é necessário definir um mecanismo que permita os movimentos dos AGVs de forma eficiente. As principais regras estabelecidas são:

- para um movimento através de um determinado arco, a gestão de tráfego bloqueia todos os nós físicos ocupados pelo AGV durante o movimento (incluindo o nó destino), bem como todos os outros nós que não existam fisicamente, mas que representem uma área partilhada que necessite ser gerida (e.g. cruzamento);
- decorre da regra anterior, que o movimento através de um determinado arco não é permitido, se algum dos nós a bloquear já estiver bloqueado por outro AGV (reservado para outro AGV);
- quando um AGV chega ao nó destino, liberta os nós anteriormente bloqueados, incluindo o nó origem, deixando reservados apenas aqueles que representem o espaço físico ocupado pelo veículo na sua nova posição (nó destino); permite assim que possam ocorrer outros movimentos de AGVs em direcção à sua posição anterior, ou seja, em direcção ao nó origem;
- para garantir um movimento sem paragens desnecessárias do veículo, é necessário ter, se possível, um número de operações elementares (movimentos) em avanço; se isto não acontecer, a ordem para o movimento seguinte só é enviada quando o AGV completar o movimento actual, implicando portanto a paragem do veículo antes do início do próximo movimento; o tempo de paragem depende do atraso na comunicação da ordem para o movimento seguinte.

3.3.3 Informação relativa ao layout utilizada na gestão de tráfego

No desenvolvimento de um SGAGVs, deve-se garantir que o *software* em causa seja não dependente de um *layout* em particular, sendo para isso necessário permitir que a informação sobre o *layout* (nós, arcos ou ramos e nós bloqueados) possa ser configurada através de um ficheiro de configuração.

De acordo com a discussão efectuada em 3.1.1, o programa de gestão do SAGVs guarda a informação contida no ficheiro numa estrutura hierárquica de 3 níveis (nó, ramo e bloqueio), baseada em listas ligadas. Essa estrutura consiste numa lista ligada de nós, em que cada elemento guarda informação do próprio nó, incluindo um apontador para uma lista de

ramos (designação utilizada ao longo do trabalho para referir um arco), que têm como origem esse nó. Cada elemento da lista de ramos contém a informação associada ao ramo, incluindo um apontador para o nó destino e um apontador para uma lista de nós a bloquear, antes do início do movimento de um AGV através do ramo.

De seguida, é apresentada a estrutura do ficheiro de configuração do *layout* utilizada na realização do trabalho [11], que inclui a informação utilizada pela gestão de tráfego, a saber: os nós bloqueados por cada ramo e a lista de ocupação de nós associada a cada nó (normalmente esta lista é vazia).

Um nó tem a seguinte informação associada:

- id_no – identificador único do nó;
- tipo_no – inteiro que representa o tipo de nó em causa, podendo tomar valores entre 0 e 3 (ver Tabela 3.4); este parâmetro identifica quais as operações de carga/descarga que podem ocorrer no nó;
- uma lista de ocupação de nós – por defeito, quando um AGV termina o seu movimento e chega a um nó, liberta todos os nós bloqueados, excepto o nó correspondente à sua posição actual; contudo, pode haver casos, em que para além de continuar a bloquear esse nó, seja necessário manter bloqueados outros nós vizinhos; quando isto acontece, os nós adicionalmente ocupados pelo AGV são indicados nesta lista;
- uma lista de ramos com origem no nó.

Tabela 3.4 – Operações de carga/descarga permitidas por cada tipo de nós existentes num layout de AGVs.

Tipo	Parâmetro "tipo_no"	Operações de Carga/Descarga permitidas
TRANSITO	0	nenhuma
CARGA_DESCARGA	1	operações de carga e de descarga
CARGA	2	só operações de carga
DESCARGA	3	só operações de descarga

Um ramo tem a seguinte informação associada:

- no_dest – identificador no nó destino;
- recta – distância em linha recta associada ao movimento;
- curva – distância em curva associada ao movimento;
- tras – distância em marcha-atrás associada ao movimento;
- peso_aux – penalização auxiliar do custo associado ao movimento;
- tipo – inteiro que representa o tipo de ramo em causa, podendo tomar valores entre 0 e 2 (ver Tabela 3.5); este parâmetro identifica em que fases de um trajecto o ramo pode ser utilizado;
- uma lista de bloqueios necessários à execução do movimento.

Tabela 3.5 – Tipos de ramos existentes num layout de AGVs.

Tipo	Parâmetro "tipo"	Descrição
TERMINAL	0	só pode ser o último ramo de um trajecto
PASSAGEM	1	não pode ser o último ramo de um trajecto
TERM_PASSAGEM	2	ambas as funcionalidades

Um elemento da lista de bloqueios de um ramo tem a seguinte informação:

- id_no – identificador do nó bloqueado;
- sentido – sentido com que um AGV ocupa o nó bloqueado, relativamente a um referencial colocado no nó, e orientado segundo o sentido arbitrado como o sentido positivo; esta informação só tem utilidade relativamente ao nó destino de um trajecto, servindo para determinar a orientação final do AGV; a Figura 3.2 ilustra como é determinado este parâmetro, comparando a orientação do veículo com a orientação do referencial colocado no nó destino.

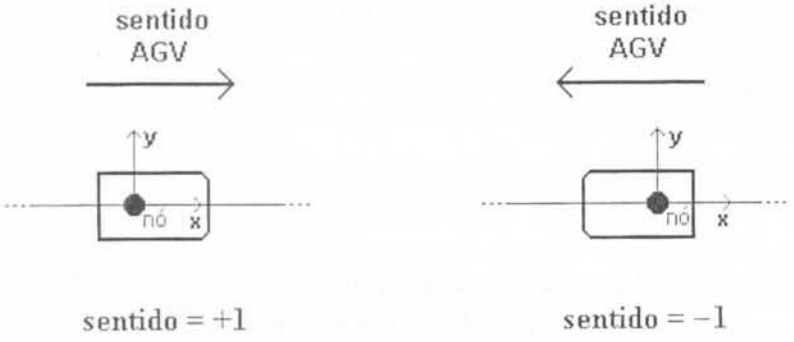


Figura 3.2 – Determinação do sentido do bloqueio de um nó do layout.

A sintaxe adoptada para o ficheiro de configuração do *layout* (ver Tabela 3.6) inclui mais campos do que aqueles que foram enumerados anteriormente, porque se trata da mesma sintaxe utilizada em SGAGVs reais, concebidos pela empresa portuguesa EFACEC, Automação e Robótica, S.A.. A adopção de uma sintaxe comum à dos sistemas da EFACEC, teve por objectivo possibilitar a eventual utilização das instalações de AGVs já implementadas, como casos de estudo no âmbito da realização deste trabalho de investigação. Os campos não definidos anteriormente devem ser considerados irrelevantes no âmbito deste trabalho, sendo por isso normalmente preenchidos com o valor por defeito 0.

Tabela 3.6 – Sintaxe do ficheiro de configuração do layout.

Tipo de Linha	Sintaxe
Nó	N:<id_no>,<yi>,<xi>,<yf>,<xf>,<tipo_no>, <zona>,<zona_op>,<dir>,<alt_max>,<corr>
Ocupação de Nós	O:<n1>,...,<nn>
Ramo	R:<no_dest>,<recta>,<curva>,<tras>,<peso_aux>,<pivot>,<tipo>, <opidi>,<opidim>,<opidif>,<alt_seg>
Lista de Bloqueios	B:(<id_no>,<sentido>),...,(< >,< >)

Como se pode constatar, a sintaxe prevê a existência de linhas do tipo N para os nós, do tipo O para as listas de ocupação de nós, do tipo R para os ramos e do tipo B para as listas de bloqueios dos ramos. Na definição de um nó, a linha do tipo O é opcional, quando a lista de ocupação de nós é vazia. As linhas do tipo R são opcionais e só existem se houver ramos com origem no nó em causa. Após a definição de um nó, por cada linha do tipo R tem de existir obrigatoriamente uma linha do tipo B (é sempre necessário pelo menos bloquear o nó destino do ramo). Um nó pode ter um número arbitrário de pares de linhas (R, B).

Segue-se um exemplo de aplicação da sintaxe anteriormente descrita, apresentando-se o ficheiro de configuração do *layout* representado pela Figura 3.3.

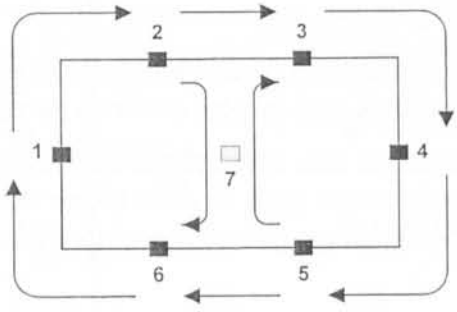


Figura 3.3 – Exemplo de um layout muito simples.

O nó 7 não é um nó físico, isto é, não representa uma posição possível para um AGV. Estes nós costumam ser designados por nós fictícios e são muitas vezes utilizados para gerir áreas partilhadas [11].

Se um AGV quisesse movimentar-se do nó 5 para o nó 3, ambos os nós estariam bloqueados até que o movimento terminasse. Por outro lado, se um AGV quisesse movimentar-se do nó 2 para o nó 6, também ambos os nós estariam bloqueados até que o movimento terminasse. Sem a existência do nó 7, nada poderia evitar que ambos os movimentos referidos se realizassem em simultâneo, e os AGVs colidiriam. Uma solução possível para o problema seria bloquear os nós 2, 3, 5 e 6, sempre que fosse necessário executar um daqueles movimentos. Mas esta não seria uma boa solução, porque iria limitar seriamente o número de AGVs possível no sistema, uma vez que não poderia haver nenhum AGV em qualquer um daqueles 4 nós, sempre que um outro AGV executasse um dos movimentos referidos. A melhor solução é bloquear o nó fictício 7 em ambos os movimentos, permitindo evitar a colisão dos AGVs e não limitando o número de AGVs no sistema.

O ficheiro de configuração do *layout* da Figura 3.3 é o seguinte:

```
N:1,0,0,0,0,1,0,0,0,0,0
R:2,2000,1000,0,0,0,1,0,0,0,0
B:(2,1)

N:2,0,0,0,0,0,0,0,0,0,0
R:3,2000,0,0,0,0,1,0,0,0,0
B:(3,1)
R:6,2000,2000,0,0,0,1,0,0,0,0
B:(6,1),(7,1)

N:3,0,0,0,0,0,0,0,0,0,0
R:4,2000,1000,0,0,0,1,0,0,0,0
B:(4,1)
```

```
N:4,0,0,0,0,1,0,0,0,0,0
R:5,2000,1000,0,0,0,1,0,0,0,0
B:(5,1)
N:5,0,0,0,0,0,0,0,0,0,0
R:6,2000,0,0,0,0,1,0,0,0,0
B:(6,1)
R:3,2000,2000,0,0,0,1,0,0,0,0
B:(3,1),(7,1)

N:6,0,0,0,0,0,0,0,0,0,0
R:1,2000,1000,0,0,0,2,0,0,0,0
B:(1,1)

N:7,0,0,0,0,0,0,0,0,0,0
```

Como se pode constatar, o nó fictício é definido à custa de uma única linha do tipo N, uma vez que não existem ramos com origem nesse nó (nem com destino nesse nó).

Os nós fictícios também têm aplicação noutros casos em que há recursos a partilhar, como por exemplo um cruzamento onde só pode passar um AGV de cada vez [11]. A Figura 3.4 representa um situação típica em que é necessário utilizar nós fictícios para gerir o tráfego nos cruzamentos.

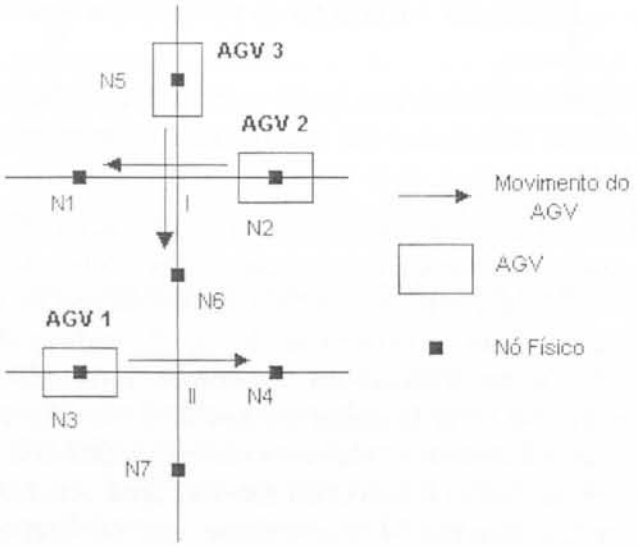


Figura 3.4 – Layout com áreas partilhadas.

No exemplo anterior, os recursos partilhados são os cruzamentos I e II. De facto, utilizando a simples teoria de bloqueio de nós, nada pode evitar os movimentos simultâneos de AGVs indicados pelas setas. Se esses movimentos ocorrerem simultaneamente, dá-se a colisão entre veículos.

Mais uma vez, a maneira mais óbvia de evitar essa situação seria bloquear alguns nós extra. Se o AGV 2 bloqueasse o nó N6, então o AGV 3 não poderia iniciar o seu movimento, antes do outro AGV chegar ao seu destino (N1). O mesmo aconteceria nos movimentos cruzados dos AGVs 1 e 3, em que o AGV 1 teria de bloquear também o nó N6. Todavia, esta solução seria ineficiente, uma vez que os AGVs 1 e 3 não poderiam executar os seus movimentos em simultâneo, quando não há qualquer restrição física para que isso aconteça.

A solução mais eficiente é utilizar nós fictícios representativos dos recursos partilhados (cruzamentos). Como é mostrado na Figura 3.5, antes de qualquer movimento que implique o atravessamento de um cruzamento, o nó fictício correspondente tem de ser bloqueado, evitando-se de forma eficiente a ocorrência de uma colisão entre AGVs.

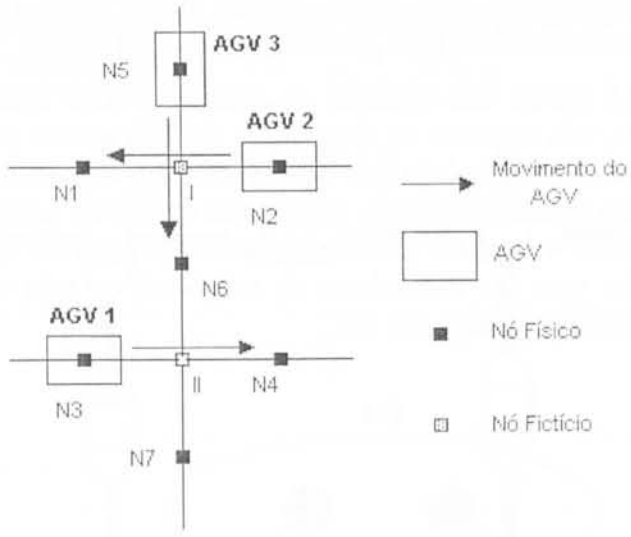


Figura 3.5 – Solução para o problema do layout com áreas partilhadas.

3.3.4 Deadlocks

Ao longo da última década, a investigação na área de sistemas de fabrico tem dado especial atenção a métodos de concepção, escalonamento, controlo e análise de desempenho de sistemas flexíveis de fabrico (FMS). A flexibilidade destes sistemas está estritamente relacionada com a rede de transporte programável que interliga as estações de trabalho, e com o controlador que monitora o progresso do fabrico dos vários lotes, e que coordena esse progresso. O fluxo concorrente de múltiplos lotes num sistema flexível de fabrico, que competem pela alocação de um conjunto finito de recursos partilhados, pode levar à ocorrência de situações clássicas de *deadlock*.[12].

3.3.4.1 Definição e identificação de um deadlock

Um *deadlock* é uma situação em que um grupo de processos, que concorrem pela utilização de um conjunto finito de recursos, ficam permanentemente bloqueados, como resultado de cada processo possuir o controlo de um subconjunto dos recursos por si requeridos, e de estar à espera da libertação dos restantes recursos, controlados por outros processos do mesmo grupo [11].

O fenómeno de ocorrência de *deadlocks* tem sido estudado intensivamente no contexto da ciência dos computadores e dos sistemas operativos. Segundo [12], as condições necessárias para a ocorrência de um *deadlock* são:

- Exclusão mútua – os recursos partilhados são utilizados de forma exclusiva;

- Reserva e espera (*Hold and Wait*) – cada processo continua a bloquear os recursos que já lhe estão atribuídos, até conseguir obter os outros recursos requeridos;
- Não preempção – os recursos reservados por um processo só serão libertados como resultado da acção voluntária desse processo;
- Espera circular (*Circular waiting*) – os processos participantes na situação de *deadlock* formam uma cadeia circular, de tal forma que cada processo bloqueia recursos necessários pelo processo seguinte da cadeia.

A Figura 3.6 ilustra uma situação de *deadlock*. Neste exemplo há duas unidades de recursos do tipo R1, três unidades de recursos do tipo R2 e cinco processos concorrentes P1 a P5, tendo os recursos disponíveis já sido atribuídos. P3 e P4 poderão eventualmente terminar, mas P1, P2 e P5 não podem prosseguir, porque não é possível satisfazer qualquer dos pedidos de reserva de recursos requeridos por esses processos, a não ser que um deles seja colocado num estado anterior (por um mecanismo de *roll-back*) ou seja terminado, para libertar os recursos actualmente controlados por ele.

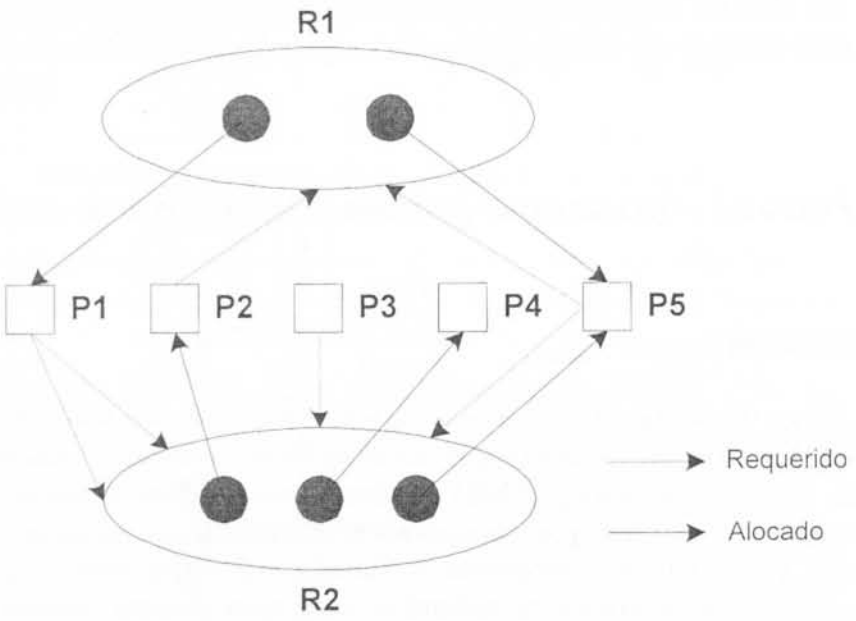


Figura 3.6 – Grafo de estados representativo de uma situação de deadlock.

Um *deadlock* é habitualmente detectado através de um grafo de estados do sistema [14] como o da Figura 3.6. Um grafo de estados é um grafo dirigido em que os vértices representam processos e recursos, e em que os arcos representam alocações de recursos e recursos requeridos.

Existe um *deadlock* quando há um ou mais ciclos no grafo reduzido de um grafo de estados [14]. A redução de um grafo de estados é um procedimento iterativo, que elimina todos os processos que não bloqueiam quaisquer recursos (e.g. P3 na Figura 3.6), ou que não tenham requisitado quaisquer recursos (e.g. P4 na Figura 3.6), e todos os arcos ligados a processos eliminados, alocando depois os recursos libertados a outros processos, que tenham requisitado os mesmos tipos de recursos.

Nos sistemas de fabrico que incluam SAGVs, os *deadlocks* podem ocorrer a vários níveis do sistema de controlo [14].

Podem ocorrer *deadlocks* ao nível da gestão de tráfego quando, por exemplo, todos os veículos do sistema ficam bloqueados por causa de AGVs parados nos cruzamentos (ver Figura 3.7). Este *deadlock* é resolvido, removendo um dos veículos de um dos cruzamentos. Pode ser prevenido, implementando uma simples política de gestão de tráfego, que impeça a paragem de AGVs nos cruzamentos.

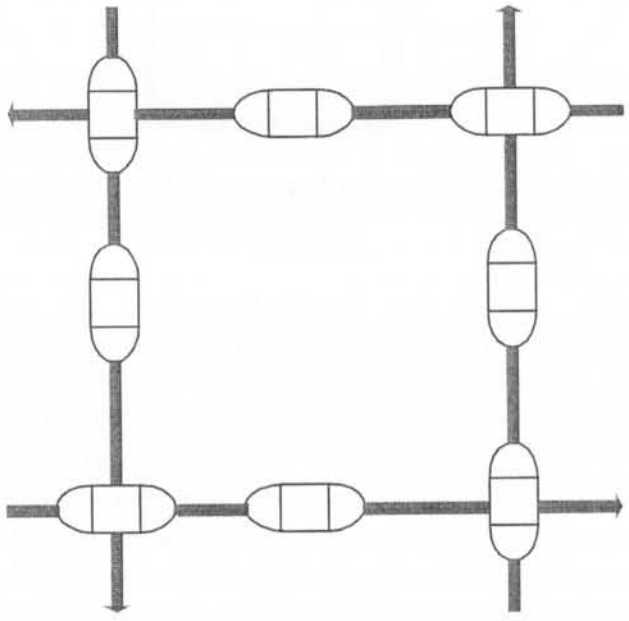


Figura 3.7 – Deadlock causado por uma gestão de tráfego inconveniente.

Outro tipo de *deadlock* que pode ocorrer nos sistemas de fabrico situa-se ao nível do planeamento de trajectórias. Na situação ilustrada pela Figura 3.8, o veículo Vb tem de ser encaminhado para o nó D1, para descarregar aí a carga que acabou de receber no nó P2, e o veículo Va tem de ser encaminhado para o nó P2, para efectuar aí uma carga. Se as decisões de planeamento de trajectórias forem feitas de tal forma que um veículo só possa sair do nó actual, quando o seu nó destino estiver livre, então forma-se uma situação de *deadlock*. Esta situação pode ser resolvida com uma estratégia de encaminhamento nó a nó, em que o planeamento de trajectórias decide só o próximo nó para onde o veículo deve ser movimentado, baseado no estado dos nós vizinhos e na informação disponível sobre o sistema global, ou atribuindo uma prioridade elevada ao veículo carregado, de tal forma que o veículo vazio possa ser enviado temporariamente para outro nó. Alternativamente, essa situação pode ser prevenida ao nível das decisões de despacho dos veículos, de tal forma que haja sempre no máximo um veículo a ser encaminhado.

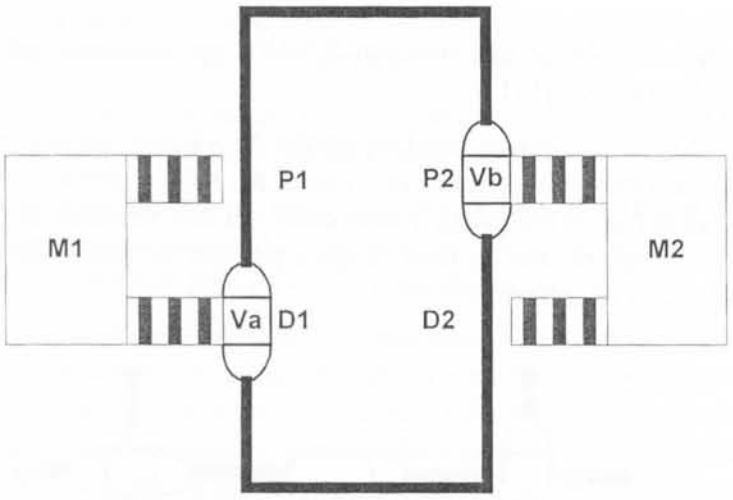


Figura 3.8 – Deadlock causado por um planeamento de trajectórias descuidado.

Também pode ocorrer um *deadlock* quando os veículos são despachados descuidadamente num sistema sobrecarregado, tal como ilustra a Figura 3.9. Nesta situação, a operação da máquina está bloqueada porque o *buffer* de saída está cheio, o veículo Va está à espera para descarregar no *buffer* de entrada, que também está cheio, e o veículo Vb, encaminhado para o *buffer* de saída, está bloqueado pelo veículo Va. Esta situação de bloqueio propaga-se gradualmente ao resto do sistema e pode levar à ocorrência de um *deadlock*, a não ser que alguma acção seja tomada para remediar a situação. Se o SAGVs for unidireccional, um empilhador manual terá de se dirigir ao *buffer* de saída, para desbloquear a operação de fabrico. Se o SAGVs for bidireccional, este procedimento pode ser automatizado através de um *buffer* central, utilizado para armazenar paletes bloqueadas nos veículos [16]. Em [14] é sugerido outro método para prevenir este tipo de *deadlock*. Esse método consiste em evitar que as paletes sejam despachadas para qualquer estação bloqueada, tendo em linha de conta o número de posições livres no *buffer* de entrada da sua próxima estação de trabalho, e as paletes que estejam em cada momento a serem transportadas para o mesmo *buffer*.

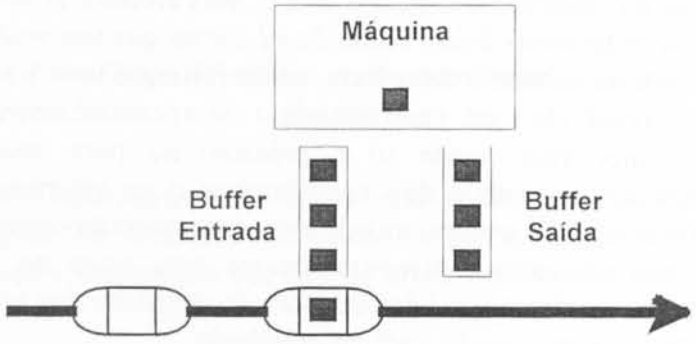


Figura 3.9 – Deadlock causado por más decisões no despacho dos AGVs.

O último procedimento pode provocar a ocorrência de outro tipo de *deadlock*, como mostra a Figura 3.10. Nesta situação, as paletes no *buffer* de saída das máquinas não podem ser transferidas para as próximas máquinas, porque os *buffers* de entrada dessas máquinas estão cheios. Este tipo de *deadlock* é puramente relacionado com o controlo do fluxo de

material, e poderá ocorrer num sistema de fabrico flexível, se houver um ou mais ciclos nos fluxos agregados de material. A forma mais comum de se resolver este tipo de situações passa por transferir a palete causadora do *deadlock* para um *buffer* central.

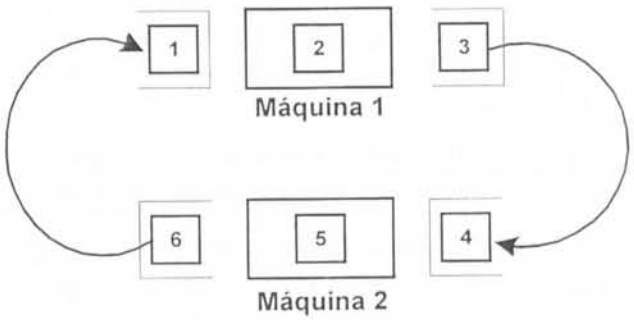


Figura 3.10 – Deadlock causado por um mau controlo do fluxo de material.

Em [13] é apresentado um modelo formal e são definidas condições necessárias e suficientes para a detecção de *deadlocks* em sistemas de fabrico, tomando como base as características do sistema em análise. Também segundo [13], o número total de possibilidades de *deadlocks* num sistema de fabrico com n máquinas (npd) pode ser determinado pela Equação 3.9, podendo ocorrer mais do que um *deadlock* em simultâneo.

$$npd = \sum_{i=2}^{n} C_i^n \tag{3.9}$$

Extrapolando esta teoria para o controlo do fluxo de materiais num sistema de fabrico [14], os recursos são o espaço nos *buffers* e nas estações de trabalho e os processos são componentes de fabrico ou peças em processamento. Na gestão de tráfego de um SAGVs, os recursos são os segmentos da rede de trajectórias do (nós do *layout* dos AGVs) e os processos são os veículos.

É fácil verificar que as condições necessárias para a ocorrência de um *deadlock*, anteriormente apontadas, se podem verificar num SAGVs [15]:

- Há exclusão mútua, uma vez que um nó do *layout* representa um recurso físico, que uma vez ocupado por um AGV, não pode ser ocupado por qualquer outro AGV;
- Pode ocorrer uma situação de reserva e espera, porque quando um AGV ocupa um nó, ele mantém esse recurso reservado para si e espera pela libertação de outros recursos (nós), para que possa efectuar o seu próximo movimento;
- Não há preempção porque, quando um nó é ocupado por um AGV, ele não pode ser ocupado por outro AGV, enquanto o primeiro não o libertar;
- Pode ocorrer uma situação de espera circular, se um grupo de AGVs for incapaz de fazer qualquer movimento, porque qualquer AGV necessita de bloquear um nó ocupado por outro AGV do mesmo grupo.

3.3.4.2 Apresentação e comparação das estratégias existentes para o tratamento dos deadlocks

As estratégias para tratar situações de *deadlock* podem ser classificadas em [14] [15]:

- prevenção,
- *evitação* (do inglês *avoidance*)
- e detecção e resolução.

Enquanto que as primeiras duas abordagens asseguram que os *deadlocks* nunca ocorrem, a última prevê a sua ocorrência, detecta-os e resolve-os.

No âmbito das técnicas de prevenção da ocorrência de *deadlocks*, a estratégia mais conservadora é designada por serialização [14]. Consiste em permitir em cada momento no máximo um processo no sistema. Como é óbvio, a utilização de recursos é muito baixa, degradando fortemente o desempenho do sistema.

A prevenção da ocorrência de *deadlocks* pode ser conseguida eliminando pelo menos uma das condições necessárias apontadas em 3.3.4.1. Assim, teoricamente haveria quatro possibilidades para prevenir a ocorrência de *deadlocks*.

Num SAGVs, a exclusão mútua e a não preempção não se podem evitar, porque são condições necessárias ao seu funcionamento. Associado a cada recurso, há um nó físico que não pode ser partilhado por mais do que um AGV ao mesmo tempo.

Para eliminar as condições de reserva e espera, é necessário forçar o processo a libertar os recursos previamente reservados, sempre que tentar alocar recursos não disponíveis. Há duas possibilidades para conseguir isso: (1) o AGV tenta reservar todos os recursos necessários, antes de iniciar a execução de um trajecto desde a sua posição actual até ao destino; (2) o AGV tenta reservar todos os recursos necessários à execução do seu próximo movimento, mas sempre que um recurso esteja indisponível, ele tem de libertar todos os recursos alocados.

A segunda possibilidade é obviamente não fazível, pela mesma razão que as duas primeiras condições necessárias não podem ser evitadas: não é possível libertar um nó ocupado por um AGV, enquanto ele ocupar o espaço físico associado, ou seja, enquanto não efectuar o seu próximo movimento e não sair desse nó. A primeira possibilidade, que é designada na literatura por alocação de uma só vez [14] (*one-shot allocation*), é perfeitamente fazível, mas causa situações de extrema ineficiência, especialmente à medida que o número de AGVs cresce ou o número de nós do *layout* decresce. Sendo uma tarefa atribuída a um AGV, a totalidade do trajecto desde a posição inicial do AGV até ao destino ser-lhe-á reservado, tornando indisponível para a realização de movimentos por parte de outros AGVs uma parte significativa do *layout*, até que o AGV em causa chegue ao destino.

Por último, para se eliminar a condição de espera circular, é necessário fazer uma ordenação linear dos diferentes tipos de recursos do sistema. Nesta abordagem, os recursos do sistema são divididos em diferentes classes. Os *deadlocks* são prevenidos fazendo com que todos os processos bloqueiem os seus recursos, de acordo com um sistema de crescimento estrito das classes de recursos [15]. Além disso, a alocação de todos os recursos de uma determinada classe tem de ser feita com um único pedido. A desvantagem desta abordagem é que a alocação dos recursos tem de ser feita de acordo com uma ordem previamente estabelecida, em vez de ser feita só quando os recursos são realmente necessários. Como consequência, alguns recursos são alocados demasiado cedo, prevenindo a ocorrência de

deadlocks, mas com o custo de decrescer seriamente a competição entre processos, e de degradar o desempenho global do sistema.

A evitação de *deadlocks* consiste em assegurar que as condições necessárias para a ocorrência dos *deadlocks* nunca sejam satisfeitas na sua totalidade. Para tal, todos os recursos são analisados antes da execução de qualquer bloqueio dos mesmos, sendo a reserva de recursos apenas autorizada quando não provoque a ocorrência de um *deadlock*.

Na literatura os algoritmos que implementam esta filosofia são conhecidos por algoritmos de Banker [14], porque a lógica é similar à de um banqueiro, quando este tem de decidir sob o empréstimo de dinheiro aos seus clientes.

Neste caso, não é requerida a reserva de todos os recursos de uma só vez. Como é óbvio, requer o conhecimento de futuros eventos do sistema em análise. Num SAGVs, não é possível implementar tal abordagem, devido à incerteza dos tempos envolvidos nos movimentos entre nós. Na realidade, isto não se deve só ao facto da velocidade dos AGVs variar significativamente, mas também porque os AGVs podem parar por qualquer razão, nomeadamente por causa de erros, obstáculos devidos a objectos ou pessoas que se atravessem no seu caminho, ou devido a intervenção humana.

Excluídas as duas primeiras abordagens pelas razões entretanto apontadas, para se ter um sistema fazível com elevada concorrência entre processos, de forma a possibilitar a obtenção de um sistema eficiente, é preferível aceitar que os *deadlocks* possam ocorrer, detectá-los e resolvê-los da melhor maneira possível.

3.4 Encaminhamento

A flexibilidade de um SAGVs é influenciada significativamente pela flexibilidade do encaminhamento dos veículos. Esta permite uma rápida recuperação face a avarias no sistema, desde que nesses casos as peças possam ser encaminhadas para uma máquina alternativa, ou se os lotes puderem ser processados segundo encaminhamentos alternativos (sequências diferentes). Sendo a flexibilidade do encaminhamento uma condição necessária para se realizar o encaminhamento alternativo das peças no seio de um sistema de fabrico, não é contudo condição suficiente [2].

Para se realizar o encaminhamento alternativo das peças, é necessário que o SAGVs o permita, através da adopção de sistemas de navegação de AGVs mais flexíveis (ver 2.1), ou da criação de caminhos alternativos no *layout* de trajectórias de um SAGVs filoguiado (ver 3.1). Se os caminhos forem fixos ou únicos, o encaminhamento numa situação de avaria até estações de trabalho que já não sejam atingíveis com os caminhos disponíveis, não é possível. Neste caso, a flexibilidade do encaminhamento fica comprometida.

3.4.1 Definição do problema

Quando num SAGVs é feita uma decisão de despacho, ou seja, uma decisão de afectação de um AGV a uma tarefa de movimentação, o controlador do sistema tem de fazer um planeamento da trajectória do veículo, desde a posição actual do veículo até ao ponto de carga da peça, e desde este ponto até ao ponto de descarga. O problema de planeamento do caminho de um AGV é designado por problema de encaminhamento.

Umn dos objectivos do encaminhamento de veículos é encontrar um caminho que permita aoo veículo viajar entre dois pontos do *layout* no menor tempo possível. Este tempo depende ddas distâncias, da velocidade dos veículos e dos atrasos sofridos ao longo do percurso, c devidos a decisões de gestão de tráfego. Outro objectivo é distribuir o tráfego o melhor posssível por todos os caminhos disponíveis, para se reduzirem os níveis de congestão de tráfego » no sistema [2].

3.4.2 Sobluções existentes para o problema de encaminhamento

Nurim SAGVs, o problema de encaminhamento tem de ser resolvido em tempo-real, sendo por r isso bastante importante o tempo de resposta do controlador. É necessário seleccionarar o melhor caminho o mais rápido possível. O tempo de resposta do controlador depende, ddentre vários aspectos, da simplicidade dos algoritmos utilizados.

O aalgoritmo de encaminhamento dos AGVs deve ser robusto, ou seja, deve ser capaz de se adaptotar a mudanças que possam ocorrer ao nível da topologia do *layout* e da intensidade de tráfego.).

A pprincipal tarefa do encaminhamento de um veículo é encontrar um caminho desde a sua posição actual até ao seu destino final. O escalonamento do veículo tem de assegurar um percurso isento de colisões ao longo do caminho seleccionado. As tarefas de encaminhamento e de escalonamento realizadas pelo controlador de um SAGVs estão intimamente relacionadas, havendo dois grandes tipos de abordagem ao problema [2]:

- planeamento completo do caminho – o controlador procura um caminho completo para o veículo, desde a sua posição actual até ao seu destino final, tendo o algoritmo de escalonamento de assegurar a não existência de colisões para a totalidade do caminho;
- planeamento incremental do caminho – o encaminhamento é realizado segmento a segmento, até que o veículo chegue ao seu destino, tendo o algoritmo de escalonamento de seleccionar o próximo segmento livre de colisões.

Os algoritmos de encaminhamento e de escalonamento de AGVs podem ser classificaddos em duas grandes categorias:

- não adaptativos ou estáticos – a escolha de um caminho é feita desde a posição actual do veículo até ao destino final, não sendo considerada a intensidade do tráfego actual na rede, nem a sua actual topologia;
- adaptativos – a decisão de encaminhamento e de escalonamento dos AGVs baseia-se no conhecimento do estado actual do sistema.

Os a algoritmos não adaptativos são fáceis de utilizar e têm tempos de resposta muito curtos, devvido à sua natureza estática. Contudo, não se adaptam a mudanças do estado da rede de AGVs, , podendo revelar-se ineficientes, se as condições de tráfego sofrerem grandes variações.

Os algoritmos adaptativos são mais complexos, têm tempos de resposta maiores, mas podem ser mais eficientes e mais robustos, desde que consigam acompanhar a dinâmica do SAGVs. Estes algoritmos dividem-se em três classes:

- Centralizados – a decisão de encaminhamento é feita num centro de controlo de encaminhamento, que baseado na informação global disponível sobre o estado de toda a rede de AGVs, determina um caminho para o AGV, desde a sua posição actual até ao destino final;
- Locais – o processo de decisão é descentralizado, em que cada AGV ao chegar a um nó selecciona o próximo nó para onde se deve dirigir; o próximo nó é seleccionado com base na distância e no tráfego existente entre o nó actual e todos os nós vizinhos, sendo escolhido aquele que possibilitar um menor tempo de viagem;
- Distribuídos – trata-se de uma combinação dos dois tipos anteriores; um veículo ao chegar a um nó determina o próximo nó, sendo esta decisão tomada com base na informação local e na informação global sobre o estado da rede.

Nos algoritmos centralizados, o centro de controlo de encaminhamento escolhe o melhor caminho para o AGV, mas estando o veículo a executar o trajecto, se por qualquer razão o AGV não cumprir os requisitos temporais inicialmente previstos pelo escalonador, o escalonamento efectuado torna-se inválido. Devido à quantidade de informação processada antes da tomada de decisão, esses algoritmos têm tempos de resposta indesejavelmente grandes, e tanto maiores quanto mais complexa for a rede, e quanto mais veículos tiver a frota de AGVs.

As técnicas de encaminhamento local são muito simples e têm tempos de resposta mais curtos. Podem também ser robustas, uma vez que a decisão tomada só afecta o movimento até ao próximo nó. A desvantagem dessas técnicas reside na escolha do próximo nó, que não garante a obtenção de um caminho optimizado, desde a posição inicial do veículo até ao destino final.

Os algoritmos distribuídos são robustos, porque o planeamento, tal como nas técnicas de encaminhamento locais, é apenas efectuado para o próximo nó. Contudo, a decisão é condicionada quer por informação local, quer por informação global do sistema. Tipicamente, estes algoritmos são mais simples do que os algoritmos centralizados, mas não podem garantir a optimização das soluções.

Apesar do reconhecimento da importância de um bom algoritmo de encaminhamento na obtenção de um bom desempenho global dos sistemas de fabrico, os algoritmos de encaminhamento locais e distribuídos ainda não foram suficientemente investigados, de forma a permitir a sua utilização nos SAGVs actualmente implementados. Por isso, a maior parte dos algoritmos de encaminhamento actualmente utilizados nos SAGVs são estáticos. Em [2] é apresentado o desenvolvimento de um algoritmo para o encaminhamento incremental de AGVs, sendo comparado o seu desempenho com as actuais técnicas de planeamento global do caminho, de natureza estática.

O encaminhamento de AGVs com janelas temporais (*time windows*) é uma generalização do problema clássico de encaminhamento, que consiste em respeitar os tempos de distribuição permitidos, ou seja, as janelas temporais definidas [18]. Neste tipo de problemas, o serviço prestado pelo AGV (servidor) pode começar dentro de uma janela temporal, definida pelo tempo mais cedo (*earliest time*) e pelo tempo mais tarde (*latest time*), quando o cliente permitir o início da prestação do serviço.

Em [19] é proposto um algoritmo de encaminhamento, baseado no algoritmo de Dijkstra do caminho mais curto, que mantém em cada nó uma lista de janelas temporais reservadas pelos escalonamentos de AGVs já efectuados, e uma lista de janelas temporais livres, disponíveis para os veículos a escalonar.

O problema abordado consiste em encontrar um caminho desde o ponto de partida do AGV até ao seu destino final, que permita completar o movimento no mais curto espaço de tempo, sem interferir com outros AGVs já escalonados, ou seja, livre de conflitos com outros AGVs. Trata-se de um algoritmo de encaminhamento que baseia as suas decisões no conhecimento do tráfego na rede de AGVs, modelizado através de janelas temporais.

Uma janela temporal reservada é definida pelo intervalo de tempo que decorre desde a ocupação de um nó por um AGV até à sua desocupação. Os restantes intervalos de tempo, em que o nó está disponível, são designados por janelas temporais livres. O algoritmo encaminha o AGV através da rede, ocupando as janelas temporais livres dos nós por onde o veículo necessite passar.

3.4.2.1 Algoritmo do caminho mais curto

Um dos algoritmos de encaminhamento mais utilizados nos SAGVs é o algoritmo do caminho mais curto, que é um dos problemas de redes estudados na área de Investigação Operacional. Este é o algoritmo que foi utilizado resolver o problema de encaminhamento nos SAGVs modelizados neste trabalho.

O algoritmo do caminho mais curto é um caso particular de um problema mais geral da Investigação Operacional, conhecido na literatura por "*K-th Shortest Path Problem*", em que se procura determinar um *ranking* de melhores caminhos entre dois pontos de uma rede, de K=1 até K=n (n≥1), sendo K a posição do respectivo caminho nesse *ranking* [20].

Apesar do problema de determinação do caminho mais curto ser um problema NP, cuja complexidade aumenta de forma exponencial com a dimensão do problema em causa, em [17] prova-se que limitando o número de nós utilizados a um número que obviamente depende do desempenho do computador utilizado, existem várias algoritmos para a resolução do problema num tempo satisfatório.

O algoritmo mais conhecido para encontrar o caminho mais curto baseia-se no princípio de optimização de Bellman [21]: se p:1→j é o caminho mais curto entre os nós 1 e j, sendo (i,jj) o último arco de p, então p_i:1→i é o caminho mais curto entre os nós 1 e i. Com base neste princípio, para o mesmo j podem-se obter vários caminhos 1→j, determinando caminhos mais curtos p_i para vários valores de i, e adicionando-lhes um arco (i,j). Estes caminhos 1→j têm pesos w(i)+wa(i,j), onde w(i) é o peso de p_i e wa(i,j) é o peso do arco (i,j). Determinado-se um i, cujo caminho permite obter o w(i)+wa(i,j) mínimo, então esse caminho adicionado ao arco (i,j) é o caminho mais curto entre os nós 1 e j. Note-se que um arco corresponde a um ramo do *layout* dos AGVs.

As equações de Bellman são as seguintes:

$$\begin{cases} w(1) = 0 \\ w(j) = \min\limits_{i \neq j}(w(i) + wa(i,j)) \end{cases} \tag{3.10}$$

Um dos algoritmos mais conhecidos, que se baseia nestas equações para encontrar o caminho mais curto, é o algoritmo de Dijkstra [22]. Este algoritmo é adequado a redes de AGVs com um número de nós não demasiado elevado, sendo nesse caso passível de ser processado num tempo razoavelmente baixo, e de ser utilizado na maior parte dos SAGVs implementados.

De seguida apresenta-se o algoritmo de Dijkstra, utilizando uma pseudo-linguagem.

```
início
    1 ← nó origem;
    adiciona 1 a S;
    w(n) = ∞, para qualquer nó n;
    w(1) = 0;
    Repete
          n ← um nó de S;
          remove n de S;
          para todos os arcos (n,i):
                se (w(n)+wa(n,i)<w(i)) então
                      w(i)=w(n)+wa(n,i);
                      pn(i)=n;
                      se (i ∉ S) então adiciona i a S;
     até que S=∅
fim
```

O conjunto S contem os nós a serem analisados. Cada nó n contém um peso w(n), que é o menor peso desde o nó origem até ao nó destino (nó n), e a indicação do nó anterior pn(n), correspondente a esse menor peso. Cada arco entre os nós n e i tem um peso wa(n,i).

No início, antes de qualquer caminho ser conhecido, os nós pn(n) ainda são indefinidos e os pesos w(n) têm um valor infinito (maior inteiro representável pelo programa de computador), excepto o nó origem, em que w(0)=0. O conjunto S contém apenas o nó origem (nó 1).

O procedimento começa a partir do nó origem, procurando um nó adjacente com um peso menor do que a soma dos pesos do nó origem e do arco respectivo. Este processamento é repetido para todos os nós existentes no conjunto S, até que S fique vazio.

Finalmente, para se obter o caminho mais curto, basta percorrer o caminho desde o nó destino até ao nó origem, de acordo com a informação indicada pelo campo pn(n) de cada nó.

O peso associado a cada arco pode ter vários significados, dependendo do critério de optimização utilizado. Exemplos de critérios de optimização são a minimização da distância percorrida, a minimização do tempo de viagem através do caminho, etc.

Considerando apenas pesos temporais, o tempo total de viagem encontrado pelo algoritmo do caminho mais curto pode não corresponder ao tempo de viagem real, devido à incerteza dos tempos de viagem dos AGVs entre os nós. Contudo, se os tempos forem estocásticos mas independentes, ou seja, se a distribuição dos tempos de viagem de um nó para outro não depender do tempo de viagem de outros veículos, o trajecto correspondente ao menor tempo esperado pode ser determinado pelo algoritmo de Dijkstra. [23].

3.5 Escalonamento

Na maioria da literatura relacionada com o escalonamento em SAGVs, ou se estabelece a comparação do desempenho de várias regras de despacho de AGVs, na reacção a

um escalonamento especificado, e num *layout* particular, ou se discute a sua importância na fase da concepção do sistema.

Obviamente, seria de esperar um aumento do desempenho do sistema de fabrico flexível resultante da coordenação de máquinas de fabrico e de AGVs, durante a fase do escalonamento das máquinas. Em [24] explora-se esta abordagem, formulando um modelo híbrido não linear e de programação inteira, desenvolvendo um procedimento iterativo para o escalonamento e testando-o numericamente em vários problemas exemplo. O problema formulado consiste em determinar os tempos de início e de fim de operação para cada lote, as viagens entre estações de trabalho e a afectação de veículos, de acordo com um critério de minimização designado por *makespan*. Este critério é um dos mais utilizados, dentre os critérios baseados em tempos de conclusão. Ele implica que o custo de um escalonamento depende do intervalo de tempo durante o qual o sistema é afectado a um conjunto de lotes.

Apesar do interesse desta abordagem, ela aumenta muito a complexidade do problema de escalonamento, porque resulta em tempos de viagem dependentes da sequência, para as viagens de AGVs descarregados entre viagens sucessivas dos AGVs carregados. Além disso, ainda é um assunto actualmente investigado, não sendo ainda muito segura a sua aplicação nos SAGVs reais.

Assim, o problema de escalonamento dos AGVs é tratado de forma independente do escalonamento do sistema de fabrico, através da aplicação de regras de despacho. Estas são heurísticas que permitem resolver o problema de afectação de um AGV a várias tarefas pendentes, ou de afectação de uma tarefa pendente a vários AGVs disponíveis, segundo determinados critérios definidos para o sistema em causa. Só assim o problema de escalonamento de AGVs se torna resolúvel em tempo real [4].

3.5.1 Definição do problema

Numa célula de fabrico constituída por vários centros de maquinação, desempenhando diferentes tipos de operações de fabrico, uma peça típica visita vários centros antes de serem satisfeitos os seus requisitos de produção nessa célula. Quando isso acontece, gera-se um problema de afectação de um AGV à tarefa, que corresponda ao transporte dessa peça até outra célula do sistema de fabrico, servida pelo SAGVs.

Quando um AGV completa uma tarefa de distribuição, o veículo é imediatamente afectado a outra tarefa, se houver alguma tarefa de transporte não atendida. Se não houver nenhuma tarefa pendente, o AGV permanece parado e livre, até que apareça uma tarefa de transporte e ele lhe seja afectado. Se no momento em que o AGV termina a execução de uma tarefa de transporte houver apenas uma tarefa pendente, o problema de afectação é trivial. Contudo, se houver várias tarefas pendentes esperando pela sua afectação a um AGV, então é preciso decidir qual a tarefa a afectar ao AGV que acaba de ficar disponível. Este problema de afectação pode aparecer noutra forma: quando há vários veículos disponíveis e surge uma tarefa, é necessário escolher um dos AGVs disponíveis, que esteja melhor colocado para executar essa nova tarefa.

Esta é a formulação do problema de afectação num SAGVs. Conforme se constata desta formulação, as decisões de despacho de AGVs dividem-se em duas categorias [16]:

- Despacho desencadeado por um centro de trabalho – consiste em seleccionar um veículo dentre um conjunto de veículos disponíveis, quando um centro de trabalho termina a sua operação e requer o transporte da peça que acaba de ser processada, desde esse centro até outro centro de trabalho;
- Despacho desencadeado por um veículo – consiste em seleccionar um centro de trabalho, dentre aqueles que requererem a execução de uma tarefa de transporte, para afectar o AGV, que acaba de ficar disponível, à sua tarefa.

3.5.2 Fases da execução de uma tarefa de transporte por um SAGVs

A execução de uma tarefa de transporte típica, por parte de um AGV, pode ser dividida nas seguintes fases:

- Passo 1 – movimento do AGV desde a sua posição inicial até ao ponto de carga da peça a transportar;
- Passo 2 – movimento de transferência de carga que carrega a peça a transportar no AGV;
- Passo 3 – movimento do AGV desde o ponto de carga da peça até ao ponto onde deve ser distribuída;
- Passo 4 – movimento de transferência de carga que descarrega a peça que acaba de ser transportada pelo AGV.

3.5.3 Objectivo de um escalonador e avaliação do seu desempenho

O objectivo do escalonamento dos AGVs é minimizar o tempo de atendimento das tarefas (tempo que as tarefas permanecem em fila de espera) e o tempo de execução do passo 1 das tarefas, ou seja, minimizar o tempo que os AGVs viajam descarregados. Quanto melhor for cumprido esse objectivo, maior será o número de tarefas executadas pelo SAGVs no mesmo intervalo de tempo e, portanto, melhor será o seu desempenho.

3.5.4 Descrição de regras de despacho habitualmente utilizadas nos SAGVs

São em grande quantidade as heurísticas conhecidas para o escalonamento em sistemas de fabrico. Em [25] são enumeradas cerca de 100 regras de despacho aplicáveis no âmbito dos sistemas de fabrico, resultantes de investigações no campo do sequenciamento e do escalonamento.

Essas regras de despacho podem ser classificadas em cinco categorias:

- Simples regras de prioridade – regras baseadas em informação relacionada com um único lote, tais como: prazo de entrega, tempo de processamento, número de operações que faltem realizar, etc;

- Combinação de simples regras de prioridade – regras que resultam da aplicação de mais do que uma regra de prioridade, consoante as diferentes circunstâncias encontradas;
- Índices de prioridade pesados – regras que resultam da combinação pesada de várias regras de prioridade diferentes;
- Regras de escalonamento heurísticas – regras que envolvem a consideração de factores mais complexos, tais como: carga antecipada de máquinas, o efeito de encaminhamentos alternativos, etc;
- Outras regras – regras estudadas para sistemas específicos, combinações de índices de prioridade baseados em funções matemáticas com parâmetros de lotes como argumentos, ou outras regras não contempladas nas classes anteriores.

Em [[16] é feita a caracterização de várias regras de despacho utilizáveis em SAGVs, para o despacho desencadeado por um centro de trabalho, e para o despacho desencadeado por um veículo. De seguida irão ser apresentadas as regras de despacho mais importantes, que foram utilizadas no estudo efectuado no âmbito deste trabalho, baseado na simulação em computador de um SAGVs.

3.5.4.1 Regras para o despacho desencadeado por um centro de trabalho

Estas regras são aplicáveis quando num sistema de fabrico não há tarefas pendentes e há um ou mais AGVs disponíveis, e um centro de trabalho requer a execução de uma nova tarefa de transporte por parte do SAGVs. É necessário afectar à nova tarefa o AGV que estiver melhor colocado para a executar, dentre os AGVs disponíveis. Cada uma das regras seguintes define um critério diferente a utilizar na escolha do AGV melhor colocado.

3.5.4.1.1 Regra do veículo mais próximo (VMP)

Esta regra é conhecida na literatura em inglês por *nearest vehicle rule*. Na aplicação desta regra, é calculada a distância desde o ponto de carga da nova tarefa até à posição actual de cada AGV disponível, sendo escolhido o AGV cuja distância seja menor.

A regra pode ser representada pela Equação 3.11, em que d_i é a distância do veículo i até ao ponto de carga da tarefa. Esta distância pode ser determinada pela Equação 3.12, em que J é o número de nós do caminho mais curto entre a posição do veículo i e o ponto de carga da tarefa. A distância d'_i representa a distância do veículo i até ao próximo nó do seu caminho mais curto, se estiver posicionado entre dois nós, valendo zero, se estiver posicionado num nó do trajecto. A distância $d(n_k,n_{k+1})$ representa a distância entre nós consecutivos do caminho mais curto.

$$d_j = min(d_i) \quad \text{para todos os AGVs disponiveis i} \tag{3.11}$$

$$d_i = d'_i + \sum_{k=1}^{J-1} d(n_k, n_{k+1}) \tag{3.12}$$

Em vez da regra ser baseada na distância mais curta, pode ser baseada no tempo de viagem mais curto. As expressões anteriores continuam a ser válidas, desde que se substituam as distâncias pelos tempos de viagem respectivos. É de notar que para uma rede de AGVs congestionada, o caminho da distância mais curta não é necessariamente igual ao caminho do tempo de viagem mais curto.

3.5.4.1.2 Regra do veículo há mais tempo disponível

Esta regra é conhecida na literatura em inglês por *longest idle vehicle rule*. A regra atribui a mais alta prioridade ao veículo que tenha permanecido disponível durante mais tempo, dentre os veículos que estão disponíveis. Isto é equivalente a despachar o veículo cuja diferença entre o tempo actual e o tempo em que o veículo tenha ficado disponível é máxima.

3.5.4.1.3 Regra do veículo menos utilizado (VMU)

Esta regra é conhecida na literatura em inglês por *least utilized vehicle rule*. A regra procura tirar vantagem do facto de na maior parte dos SAGVs haver acesso a estatísticas temporais sobre os AGVs, que permitem determinar a taxa de utilização de cada veículo. O veículo escolhido é aquele cuja taxa de utilização é menor, até ao instante em que a decisão de despacho tem de ser tomada.

Tal como a regra do veículo há mais tempo disponível, a regra do veículo menos utilizado permite balancear a carga de trabalho dos vários veículos de uma frota de AGVs.

3.5.4.2 Regras para o despacho desencadeado por um veículo

Estas regras são aplicáveis quando num sistema de fabrico não há tarefas pendentes e não há nenhum AGV disponível, e um AGV termina a execução de uma tarefa, ficando disponível para a execução de outra tarefa de transporte. É necessário decidir sobre qual a tarefa pendente a afectar a este AGV, que ficou entretanto disponível. Cada uma das regras seguintes define um critério diferente a utilizar nessa decisão.

3.5.4.2.1 Regra do tempo mais curto (ou da distância mais curta) de viagem (TMCV)

Esta regra é conhecida na literatura em inglês por *shortest travel time/distance rule*. O seu objectivo é minimizar o tempo durante o qual os veículos viajam descarregados, no acesso aos pontos de carga das tarefas que lhes sejam afectadas. A regra define que a tarefa a afectar ao AGV disponível é aquela cujo tempo ou distância da viagem, entre o ponto onde o AGV se encontra até ao seu ponto de carga, é menor.

A aplicação desta regra é na maior parte dos casos vantajosa. Contudo, a regra é muito sensível quanto ao *layout* da planta fabril e quanto à localização do equipamento na planta fabril.

Os veículos ficam disponíveis quando terminam a execução de uma tarefa de transporte. Assim, os pontos em que eles ficam disponíveis são pontos de descarga. Se por acaso um ponto de carga de um centro de trabalho não for o mais próximo de qualquer ponto de descarga, de acordo com esta regra, esse centro de trabalho nunca irá ser contemplado com uma decisão de despacho de uma das suas tarefas, a não ser que na decisão de despacho só

intervenham um único AGV e uma única tarefa. Neste caso o fluxo de saída de materiais do centro de trabalho em causa é reduzido drasticamente. Como continuam a ser efectuadas novas distribuições de materiais para esse centro de trabalho, a sua fila de saída aumenta até ao máximo da sua capacidade, provocando nessa altura o bloqueio das suas máquinas. O bloqueio das máquinas provoca o aumento da fila de entrada até ao limite da sua capacidade, uma vez que continuam a ser efectuadas aí distribuições de material. Quando isso acontecer todos os veículos afectados a tarefas, que impliquem a descarga de material nesse centro de trabalho, vão também bloquear. Desta forma, todo o sistema de fabrico pode acabar por bloquear, porque entretanto pode deixar de haver AGVs disponíveis para a satisfação de tarefas de movimentação de outros centros de trabalho.

3.5.4.2.2 Regra do máximo tamanho da fila de saída (MTFS)

Esta regra é conhecida na literatura em inglês por *maximum outgoing queue size*. Se q_k representar o número de peças existentes na fila de saída do ponto de descarga de um centro de trabalho k e se R_k é o número de peças aí existentes ainda não atribuídas a qualquer veículo, onde $1 \leq R_k \leq q_k$, será tomada a decisão de despachar o AGV disponível para o centro de trabalho i que verifique a Equação 3.13.

$$q_i = max(q_k) \quad \forall k: R_k \geq 1 \tag{3.13}$$

3.5.4.2.3 Regra do mínimo espaço livre na fila de saída (MELFS)

Esta regra é designada na literatura em inglês por *minimum remaining outgoing queue space rule*. Neste caso a regra de despacho é baseada num índice que é função da capacidade e do comprimento da fila de saída em cada centro de trabalho. Se Q_k for a capacidade da fila de saída no centro de trabalho k, S_k for o comprimento actual dessa fila e R_k o número de peças aí existentes, que ainda não tenham sido atribuídas a qualquer AGV, para cada centro de trabalho k em que $R_k \geq 1$, é calculado o índice C_k, definido pela Equação 3.14. A regra define o despacho do AGV disponível para o centro de trabalho i que verifique a Equação 3.15.

$$C_k = Q_k - S_k \tag{3.14}$$

$$C_i = min(C_k) \quad \forall k: R_k \geq 1 \tag{3.15}$$

O objectivo desta regra é reduzir a possibilidade de ocorrência do bloqueio de centros de trabalho, devido ao crescimento das suas filas de espera.

3.5.4.2.4 Regra do primeiro a chegar primeiro a ser servido modificada (PCPSM)

Esta regra é designada na literatura em inglês por *modified first come-first serve rule*. A regra procura atribuir veículos aos centros de trabalho por ordem cronológica da recepção pelo SAGVs das tarefas de transporte pendentes.

Quando um centro de trabalho lança uma nova tarefa e esta não pode ser imediatamente executada, é guardado o tempo em que a tarefa é lançada. Se houver o lançamento de novas tarefas por parte do mesmo centro de trabalho, antes da primeira tarefa pendente ser atribuída a um AGV, o seu tempo de lançamento não é guardado. Por outras palavras, nenhum centro de trabalho pode ter mais do que uma tarefa a competir pela utilização de AGVs.

Quando um AGV fica disponível, é afectado ao centro de trabalho cuja primeira tarefa pendente tenha o menor tempo de lançamento. Quando a tarefa pendente de um centro de trabalho é atendida, as necessidades de transporte do centro são avaliadas e actualizadas de uma de duas maneiras:

- se não houver mais nenhuma tarefa pendente, o centro de trabalho não necessita de mais nenhum AGV;
- se for requerido mais do que um AGV, é atribuído à nova tarefa pendente mais antiga o tempo em que o último AGV foi afectado a uma tarefa do centro de trabalho em causa, ou seja, o tempo em que foi atendida a última tarefa pendente.

Apesar desta regra não considerar directamente qualquer exaustão da capacidade das filas dos centros de trabalho, ela assegura a redução do tempo de atendimento de uma tarefa. O número de afectações feitas a um centro de trabalho está relacionado com a sua intensidade de realização de operações de fabrico.

3.5.5 Formulação matemática do problema de afectação de AGVs a tarefas

O problema de escalonamento num SAGVs é basicamente um problema de afectação de AGVs a tarefas ou de tarefas a AGVs. Para se ter uma optimização do número de tarefas de transporte executadas, para além de se necessitar de um algoritmo que determine o caminho mais curto (ver 3.4.2.1) é necessário um algoritmo que realize uma correcta afectação dos AGVs às tarefas.

A optimização da solução do problema de afectação passa pela avaliação do custo associado às soluções possíveis para o problema. Assim, dado um conjunto de tarefas pendentes e um conjunto de AGVs disponíveis, a resolução do problema de afectação consiste em determinar o custo associado a cada combinação AGV/tarefa, através da determinação do custo do respectivo trajecto mais curto (desde a posição actual do AGV até ao ponto de carga da tarefa), e em determinar qual o conjunto de afectações que permite obter o custo global mais baixo de execução do conjunto de tarefas em causa, de tal forma que cada AGV fique afectado a uma e só uma tarefa.

Se o número de AGVs for 1 ou se o número de tarefas for 1, a resolução deste problema equivale à aplicação da regra de despacho TMCV (ver 3.5.4.2.1) ou da regra de despacho VMP (ver 3.5.4.1.1), respectivamente.

A determinação do custo da afectação de uma tarefa a um AGV consiste em calcular o custo associado à execução do passo 1 da tarefa por parte do AGV (ver 3.5.2), que

corresponde ao cálculo do custo de execução do trajecto mais curto entre a posição actual do AGV e o ponto de carga da tarefa. Este custo depende do peso dos ramos do *layout*, que normalmente representa o tempo necessário para percorrer esse ramo.

O problema de optimização da afectação de um conjunto de tarefas T a um conjunto de AGVs V é um caso especial do problema de transportes estudado em Investigação Operacional [8] [21], em que cada armazém fornece 1 e cada centro de distribuição pede 1.

A formulação matemática deste problema de optimização é dada pelas Equações 3.16, em que C_{vt} representa o custo da afectação do veículo v à tarefa t (custo do caminho mais curto), NV o número de veículos e NT o número de tarefas.

$$\begin{cases} \sum_{v=1}^{NV}\sum_{t=1}^{NT} C_{vt}X_{vt} \\ \sum_{v=1}^{NV} X_{vt} = 1, \ \forall t \in T \\ \sum_{t=1}^{NT} X_{vt} = 1, \ \forall v \in V \\ X_{vt} = \begin{cases} 1 & \text{se v afectado a t} \\ 0 & \text{senao} \end{cases} \end{cases} \tag{3.16}$$

A matriz formada pelos coeficientes X_{vt} é designada por matriz de eficiência.

3.5.5.1 O algoritmo húngaro

Um algoritmo que permite obter de forma eficiente uma solução óptima para o problema de afectação formulado pelas Equações 3.16 é o chamado algoritmo húngaro [8]. Este método baseia-se no facto da adição de uma constante a todos os elementos de uma linha ou de uma coluna da matriz de eficiência não mudar a solução óptima do problema [8].

Assumindo que $C_{vt} \geq 0, \ \forall v \in V, \ \forall t \in T$, há um caso especial no qual se pode constatar facilmente que se chegou a uma solução óptima: quando se tem um custo total de afectação igual a zero. O algoritmo húngaro subtrai e adiciona valores a linhas ou colunas inteiras, de forma a obter-se um problema com a mesma solução óptima e um custo total igual a zero.

Uma possível implementação do algoritmo húngaro consiste nos seguintes passos:

1. Em cada linha da matriz, encontrar o menor elemento e subtrair esse elemento a todos os elementos dessa linha;
2. Cobrir com o menor número possível de traços horizontais e verticais (linhas ou colunas, respectivamente) todos os zeros da matriz;
3. Se o número de traços igualar o número de afectações (dimensão da matriz de eficiência), saltar para o passo 6, senão executar o passo 4;
4. Escolher o menor elemento da matriz, dentre o conjunto dos elementos que não estejam cobertos pelos traços determinados em 2;

5. Subtrair esse elemento a todos os elementos desse conjunto e adicioná-lo a elementos que pertençam a dois traços simultaneamente, ou seja, que se situem no cruzamento de um traço horizontal com um traço vertical;
6. Um AGV pode ser afectado a uma tarefa, se o seu valor na Matriz de Eficiência for zero.

Uma restrição importante deste método é o facto da matriz de eficiência ter de ser uma matriz quadrada. Por outras palavras, o número de AGVs a afectar tem de ser igual ao número de tarefas a afectar. Quando esta restrição não é respeitada pelo problema a resolver, é necessário converter esse problema num outro equivalente, cuja matriz de eficiência seja quadrada.

Para isso, criam-se AGVs fictícios ou tarefas fictícias, se NV<NT ou se NV>NT, respectivamente, de tal forma que se obtenha um problema em que NV=NT. Os custos a introduzir valem infinito (ou um valor muito grande) para a afectação de um AGV real a uma tarefa fictícia ou para a afectação de um AGV fictício a uma tarefa real, e valem zero para a afectação de um AGV fictício a uma tarefa fictícia. Este procedimento procura garantir que AGVs e tarefas fictícias sejam afectadas entre si.

O valor da dimensão da matriz n é condicionado pela capacidade de processamento do computador utilizado e pelo tempo de resposta máximo permitido na obtenção de uma solução do problema de afectação. Esse número n deve ser igual ou superior ao número de AGVs do sistema, para que não possam ficar AGVs livres desnecessariamente, quando houver tarefas suficientes para ocupar todos os veículos. As tarefas que não possam ser contempladas pelo algoritmo de afectação, ou que sejam afectadas a AGVs fictícios, ficam a aguardar em fila de espera. Da mesma forma, AGVs afectados a tarefas fictícias permanecem disponíveis.

3.5.5.2 Reafectação dinâmica

Uma das grandes vantagens da utilização de um algoritmo de Investigação Operacional, como o algoritmo húngaro, computacionalmente mais pesado do que as regras de despacho apresentadas em 3.5.4, na afectação de uma frota de AGVs a um conjunto de tarefas pendentes, é a possibilidade de implementar um mecanismo de reafectação dinâmica.

O mecanismo de reafectação dinâmica tem por objectivo optimizar ainda mais a afectação de AGVs às tarefas, à custa de um maior número de tarefas e de AGVs intervenientes no processo de escalonamento.

Sem esse mecanismo, quando o processo de escalonamento é chamado, apenas intervêm as tarefas em fila de espera e os AGVs parados e disponíveis. Com o mecanismo de reafectação dinâmica, também intervêm as tarefas (e os AGVs a elas afectados) que estejam no primeiro passo da sua execução, ou seja, em que o AGV ainda esteja a dirigir-se ao ponto de carga. Assim, estas tarefas podem perfeitamente ser afectadas de outra forma aos AGVs, se a nova afectação permitir uma melhor optimização do custo global de execução das tarefas, que o SAGVs tem em mãos.

O processo de escalonamento é chamado sempre que um AGV termine a execução de uma tarefa e fique disponível para a realização de uma outra tarefa, desde que haja pelo menos uma tarefa em fila de espera ou na primeira fase da sua execução. Pode acontecer que o AGV que acaba de ficar disponível esteja melhor colocado para executar uma tarefa actualmente em execução, do que o AGV que se encontra actualmente afectado à tarefa.

Neste tipo de situações, o mecanismo de reafectação dinâmica permite que o AGV, que acaba de ficar disponível, "pegue" numa tarefa que já estava afectada a outro AGV, resultando numa optimização importante do desempenho do escalonador.

Assim, à custa de um processamento mais pesado ao nível do processo de escalonamento, devido ao aumento da dimensão do problema de afectação, é possível obter um melhor desempenho do sistema.

4. *Teoria das Redes de Petri autónomas*

Os sistemas integrados por computador (CIM) são sistemas cujo comportamento evolui em função da ocorrência de eventos, sendo as suas actividades caracterizadas por comportamentos tais como: o sequencial, o paralelo, o concorrente, o sincronizado, a exclusão mútua, etc.

Estes sistemas fazem parte de uma classe designada por sistemas dinâmicos de eventos discretos. A relação entre eventos, a sua evolução ao longo do tempo e a sua ordem de ocorrência são de grande importância para caracterizar o comportamento desses sistemas [5]. Comparados com os sistemas modelizados através de equações diferenciais (e.g. processo de fabrico contínuo), os sistemas de fabrico flexíveis são caracterizados por mudanças qualitativas ou abruptas do seu estado (e.g. chegada de uma peça a uma máquina) condicionadas pela ocorrência de eventos, em vez de serem caracterizados por mudanças quantitativas como nos sistemas contínuos (e.g. volume de petróleo refinado numa refinaria).

As Redes de Petri (RP) são uma das ferramentas mais rigorosas e poderosas para modelizar os sistemas discretos. Seguindo a abordagem seguida em [26], este capítulo apresenta os princípios básicos das RP, tomando como referência as RP autónomas e os seus principais conceitos teóricos. A apresentação das extensões às RP autónomas, tais como, RP sincronizadas, RP temporizadas, RP estocásticas e RP coloridas, é feita no capítulo seguinte.

4.1 História do aparecimento das RP

Carl Adam Petri é um matemático alemão contemporâneo que definiu um modelo matemático de uso geral, para descrever as relações existentes entre condições e eventos.

Este trabalho de investigação foi levado a cabo entre 1960 e 1962. Desde então, tem havido um grande esforço de investigação sobre as RP, um pouco nos Estados Unidos, onde foi realizado um trabalho notável no MIT (Massachusetts Institute of Technology) nos primeiros anos da década de 70, e desde sempre um pouco por toda a Europa.

Os investigadores franceses contribuíram significativamente para o desenvolvimento das RP. Já nos anos de 1972-1973, alguns deles utilizavam este modelo para descrever controladores lógicos, ainda antes do aparecimento do Grafcet.

O Grafcet é um formalismo baseado nas RP, que é utilizado para descrever o comportamento sequencial de programadores lógicos, ou seja, sistemas com entradas e saídas e de carácter "tudo ou nada". Para descrever a ocorrência dos eventos, é utilizada a lógica booleana. Embora este formalismo tenha muitas afinidades com as RP sincronizadas, o seu interesse resulta do facto de terem sido desenvolvidas ferramentas de *software*, que partindo da descrição do comportamento de um programador lógico através de um modelo Grafcet, geram automaticamente o código do programa, que implementa o comportamento modelizado.

O modelo das RP é rico. Permite uma análise qualitativa e têm sido adaptadas a numerosas aplicações, através do estudo de extensões às RP autónomas, que possibilitam

descrições dos sistemas mais condensadas, e onde pode intervir o factor tempo: sincronizadas, temporizadas, estocásticas, coloridas, etc.

As RP apresentam duas características interessantes. Por um lado, permitem modelizar e visualizar tipos de comportamento tais como: paralelismo, concorrência, sincronização, partilha de recursos, etc. Por outro lado, são muitos os resultados teóricos que tornam possível o seu tratamento matemático. As suas propriedades foram e ainda são estudadas exaustivamente.

Os principais utilizadores das RP são os investigadores na área dos computadores e na área do controlo e automação industrial. Contudo, as RP são suficientemente gerais para permitirem modelizar fenómenos de variadíssimos tipos.

4.2 Conceitos básicos sobre RP

4.2.1 Definição de uma RP

Uma RP é um grafo bipartido, constituído por caminhos construídos à custa da alternância de posições e de transições, interligadas por arcos dirigidos.

4.2.2 Elementos constituintes de uma RP

Uma RP tem dois tipos de nós, designados por posições (tradução do termo inglês *places*) e por transições (tradução do termo inglês *transitions*), conforme ilustra a Figura 4.1. Uma posição é representada por um círculo, e uma transição é habitualmente representada por uma barra ou por uma caixa. As posições e as transições são interligadas por arcos. Estes são dirigidos, podendo ter o sentido de uma posição para uma transição ou de uma transição para uma posição.

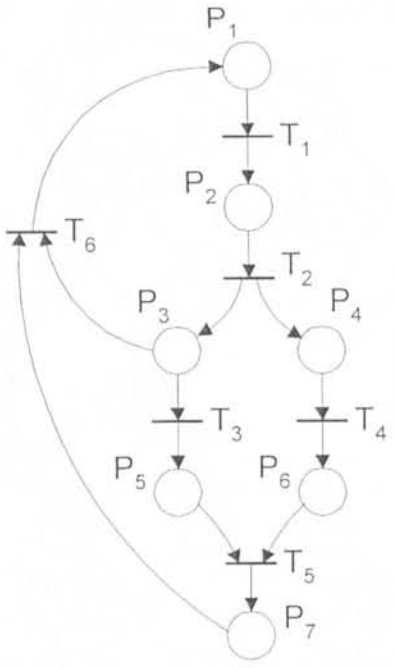


Figura 4.1 – Exemplo de uma RP.

A Figura 4.1 mostra uma RP que contém sete posições, seis transições e quinze arcos dirigidos. O conjunto das posições é $P=\{P_1, P_2, P_3, P_4, P_5, P_6, P_7\}$ e o conjunto das transições é $T=\{T_1, T_2, T_3, T_4, T_5, T_6\}$. A posição P_3 é uma posição de entrada da transição T_3, porque existe um arco dirigido de P_3 para T_3. A posição P_5 é uma posição de saída da transição T_3, porque existe um arco dirigido de T_3 para P_5. Da mesma forma, uma transição pode ser uma transição de entrada de uma posição ou uma transição de saída de uma posição.

4.2.3 Marcação de uma RP

Diz-se que a RP representada na Figura 4.1 é não marcada, porque não contém qualquer marca (habitualmente designada por *token* na literatura sobre RP) em qualquer uma das suas posições. A Figura 4.2 representa uma RP equivalente, mas marcada.

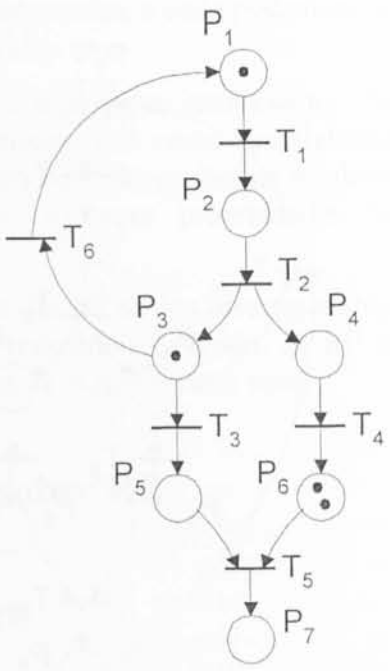


Figura 4.2 – Exemplo de uma RP marcada.

O número de marcas $M(P_i)$ existentes numa posição P_i designa-se por marcação da posição. Cada posição contém m_i marcas, sendo este sempre um número inteiro positivo ou igual a zero.. No exemplo da Figura 4.2, $m_1=m_3=1$, $m_6=2$ e $m_2=m_4=m_5=m_7=0$.

A marcação da rede M, é definida pelo vector das marcações das posições. No exemplo anterior, é o vector $M=(m_1, m_2, m_3, m_4, m_5, m_6, m_7)=(1, 0, 1, 0, 0, 2, 0)$.

A marcação da rede num dado momento define o estado da RP, ou mais precisamente, o estado do sistema descrito pela RP. A evolução do estado corresponde portanto à evolução da marcação, causada pelo disparo sucessivo de transições. Deste ponto em diante, se não for dito nada em contrário, quando se referir uma RP, assume-se que ela é marcada.

4.2.4 Mecanismo de disparo de uma transição

Uma transição só pode ser disparada se em cada uma das suas posições de entrada houver pelo menos uma marca. Neste caso, diz-se que a transição está sensibilizada (do termo inglês *enabled*). Uma transição fonte, sem qualquer posição de entrada, está portanto sempre sensibilizada.

A transição T_1 está sensibilizada nos casos a), b) e c) da Figura 4.3 (antes do disparo), porque nos três casos há pelo menos uma marca nas posições P_1 e P_2. No caso d), a transição não está sensibilizada, porque a posição P_2 não contém qualquer marca.

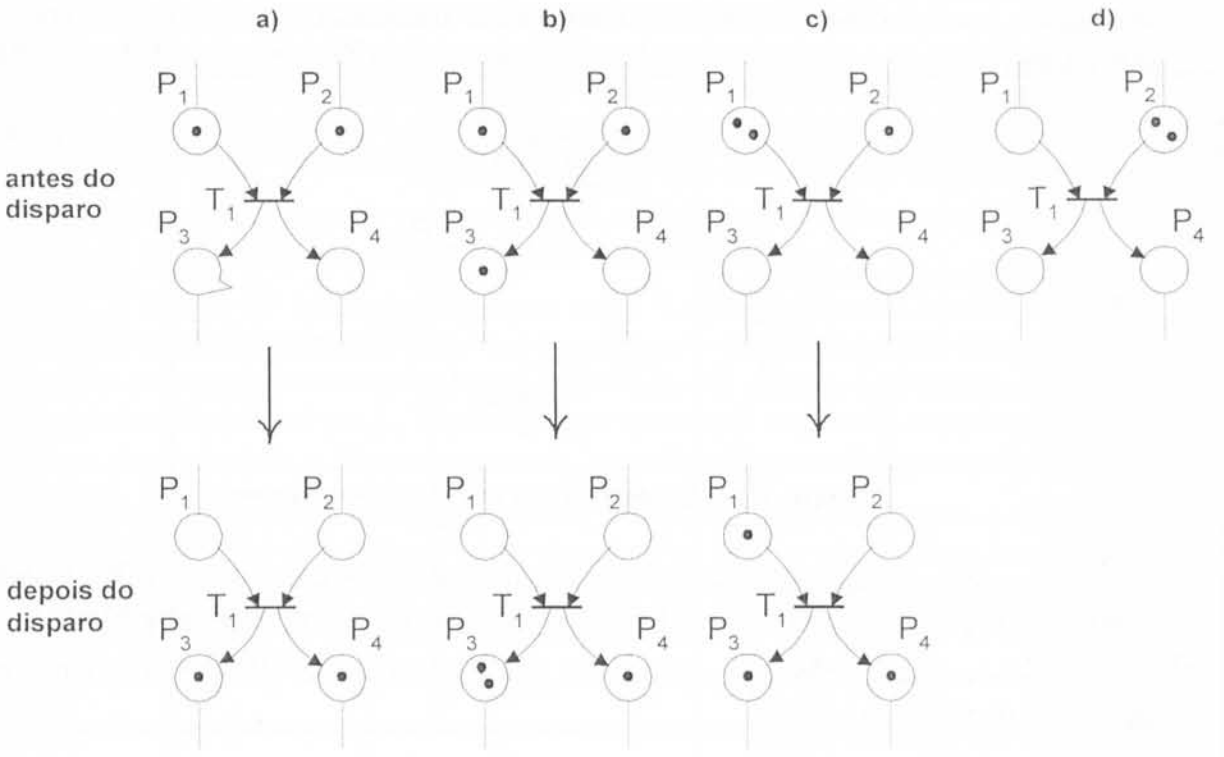


Figura 4.3 – Disparo de uma transição.

O disparo de uma transição sensibilizada consiste em retirar uma marca de cada uma das suas posições de entrada e em adicionar uma marca a cada uma das suas posições de saída. Este mecanismo é ilustrado pela Figura 4.3, que mostra a marcação das posições antes e depois do disparo da transição T_1. Note-se que no caso b) aí representado há duas marcas na posição P_3, depois do disparo da transição, porque essa posição já continha uma marca antes da ocorrência do disparo.

Quando uma transição está sensibilizada, isto não implica que ela será disparada imediatamente. Sabe-se apenas que existem condições para isso acontecer. Na Figura 4.2, há duas transições sensibilizadas: T_1 e T_3. Contudo, não se sabe quando é que essas transições serão disparadas. Sabe-se, no entanto, que a próxima evolução do estado da RP (da sua marcação) corresponderá ao disparo de T_1 ou de T_3.

O disparo de uma transição é atómico, ou seja, é indivisível. Apesar do factor tempo não estar presente nas RP autónomas, é útil considerar-se que o disparo de uma transição tem uma "duração nula", para evidenciar melhor este conceito de indivisibilidade.

4.3 RP autónomas e não autónomas

Nas RP apresentadas anteriormente, o comportamento de um sistema é descrito de forma autónoma, porque não se sabe quando é que uma transição sensibilizada será disparada, ou seja, não se conhecem os instantes de disparo das transições. Por esse facto, essas RP são denominadas RP autónomas.

A RP representada na Figura 4.4 descreve o ciclo de estados de um sistema, por exemplo de um motor, que pára, arranca, pára novamente, etc. No estado correspondente à

marcação indicada, o sistema está parado, e a única transição sensibilizada é a transição T_1. Mas aqui a transsição será disparada, quando ocorrer o evento externo correspondente, que para a transição T_1, é a ocorrência da ordem de arranque. Esta RP é uma rede não autónoma.

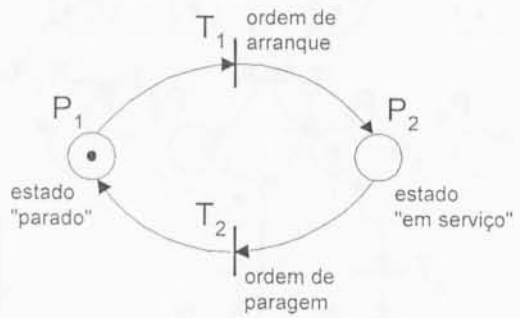


Figura 4.4 – Exemplo de uma RP não autónoma.

Uma RP não autónoma descreve o funcionamento de um sistema, cuja evolução é condicionada por: eventos externos (RP sincronizadas) ou pelo tempo (RP temporizadas).

Neste capítulo, assume-se que as RP são autónomas. As RP não autónomas serão abordadas no próximo capítulo.

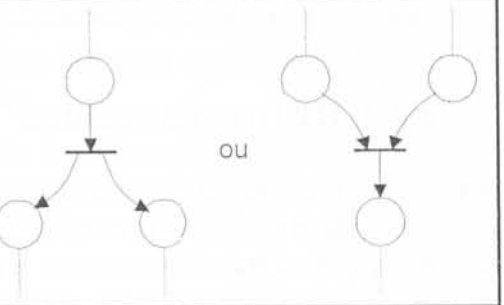
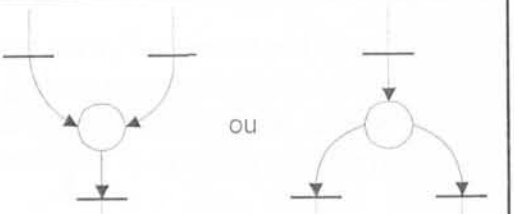
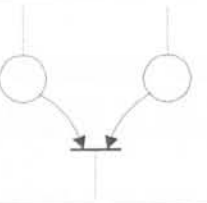
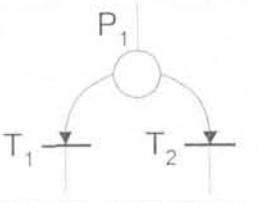
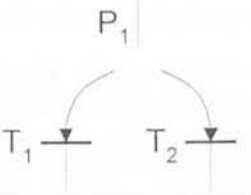
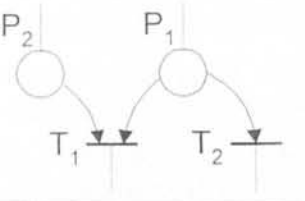
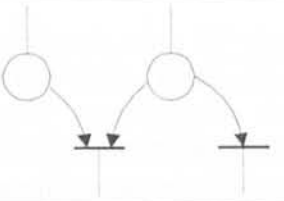
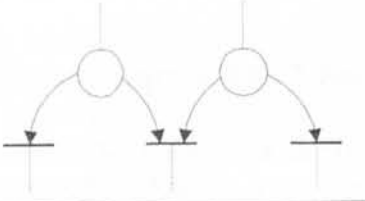
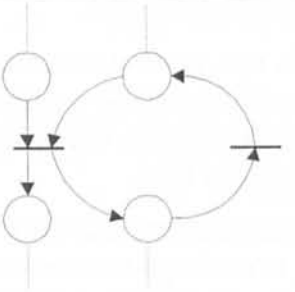
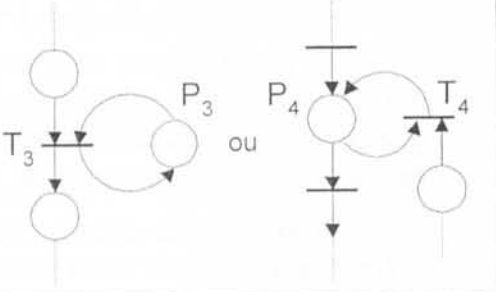
4.4 Estruturas particulares de RP

Algumas RP apresentam características estruturais particulares. Esta secção apresenta as estruturas partticulares indicadas na Tabela 4.1. Nessa tabela, também é apresentado um contra exemplo para cada estrutura particular.

4.4.1 Grafo de estados

Uma RP não marcada é um grafo de estados (da expressão inglesa *state graph*) se e só se todas as transições tiverem exactamente uma posição de entrada e uma posição de saída. É de notar, que marcando uma RP do tipo grafo de estados, o seu comportamento será equivalente a um grafo de estados clássico, se em cada momento tiver no máximo uma marca na sua marcação.

Tabela 4.1 – Estruturas particulares de RP.

Numa RP do tipo...	Pode-se encontrar...	Não se pode encontrar...
Grafo de estados		
Grafo de eventos		
Livre de conflitos		
Escolha livre		
Simples		
Pura		

4.4.2 Grafo dde eventos

Uma RP é um grafo de eventos (da expressão inglesa *event graph*) se e só se todas as posições tiveremm exactamente uma transição de entrada e uma transição de saída. Um grafo de estados marcacado também é por vezes denominado por grafo de transições. O grafo de estados é portantito a representação dual do grafo de estados.

4.4.3 RP livrre de conflitos

Uma RP livre de conflitos é aquela em que todas as posições têm no máximo uma transição de saídida. Um conflito estrutural corresponde portanto à existência de uma posição P_j, que tem no mínimo duas transições de saída, T_1, T_2, ... Este tipo de conflitos será representado ao longo do texto por $<P_1,\{T_1,T_2,...\}>$.

Se por exxemplo P_j tiver uma marca, poderá haver mais do que uma transição de saída sensibilizada, mmas apenas poderá ser disparada uma transição. Se houver várias transições candidatas ao cconsumo da marca em P_j, este conflito terá de ser resolvido segundo um determinado critétério (e.g. disparando aleatoriamente uma das transições sensibilizadas).

4.4.4 RP de eescolha livre

Neste tippo de RP existem duas variantes conhecidas por RP de escolha livre (da expressão inglessa *free-choice Petri net*) e por RP de escolha livre estendida (da expressão inglesa *extendedd free-choice Petri net*).

Uma RP de escolha livre é aquela em que para todos os conflitos $<P_1,\{T_1,T_2,...\}>$, nenhuma das traansições T_1, T_2, ... possui outra posição de entrada para além de P_1.

Uma RP de escolha livre estendida é uma rede tal, que para todos os conflitos $<P_1,\{T_1,T_2,...\}>$, todas as transições T_1, T_2, ... têm o mesmo conjunto de posições de entrada. Isto significa quue se T_1 tiver P_1 e P_2 como posições de entrada, então T_2 tem P_1 e P_2 como posições de entrarada, etc.

4.4.5 RP simples

Uma RP simples é aquela em que cada transição pode apenas ser afectada no máximo por um conflitito. Por outras palavras, se existir uma transição T_1 e dois conflitos $<P_1,\{T_1,T_2,...\}>$ e $<P_2,\{T_1,T_2,...\}>$, então a RP não é simples.

É de notatar que o conjunto das RP do tipo grafo de eventos está contido no conjunto das RP livres de conflitos, que por sua vez está contido no conjunto das RP de escolha livre, que por sua vez está contido no conjunto das RP simples. O conjunto das RP do tipo grafo de estados está contitido no conjunto das RP de escolha livre.

Num grafo de estados pode haver conflitos, mas não pode haver sincronização. Num grafo de eventos pode haver sincronizações, mas não pode haver conflitos.

4.4.6 RP puras

Um par constituído por uma posição P_i e por uma transição T_i é um auto-ciclo (do termo inglês *self-loop*), se P_i é simultaneamente uma posição de entrada e uma posição de saída de T_j. Uma RP pura é aquela em que não existem situações deste tipo.

Qualquer RP impura pode ser transformada numa RP pura equivalente, à custa da adição de mais transições e de mais posições à RP impura.

A Figura 4.5 ilustra um exemplo deste tipo de transformação. A transição T_1 foi substituída por duas transições T'_1 (transição de início) e T''_1 (transição de fim), e por uma posição, para a qual T'_1 é uma transição de entrada e T''_1 é uma transição de saída. Também foi adicionada uma posição P_0, marcada inicialmente com uma marca, cuja função é assegurar que as transições T'_1 e T''_1 são disparadas em alternância, ou seja, que T'_1 não pode ser disparada duas vezes consecutivas.

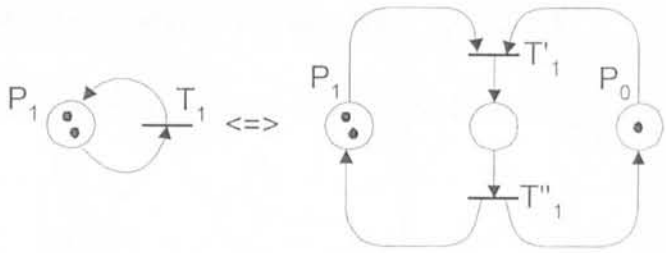


Figura 4.5 – Exemplo da transformação de uma RP impura numa RP pura.

4.5 Propriedades de uma RP

Nesta secção serão apresentadas algumas propriedades das RP autónomas. Para isso, vai-se passar a definir uma notação e alguns conceitos úteis nessa apresentação.

A Figura 4.6 representa uma RP com uma marcação inicial M_0. A marcação da RP, num dado momento, é um vector coluna, cuja i-ésima componente é a marcação da posição P_i. Para facilitar a escrita, as marcações serão escritas na forma transposta, ou seja, como vectores linha. Para distinguir melhor ambas as formas, são utilizados parênteses rectos para as matrizes não transpostas e parênteses curvos para as matrizes transpostas.

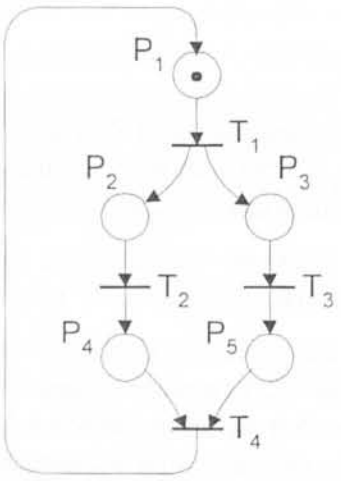


Figura 4.6 – RP com a sua marcação inicial.

Assim, a marcação inicial do exemplo anterior é representada por:

$$M_0 = (1,0,0,0,0) = [1,0,0,0,0]^T = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Para a marcação inicial M_0, a única transição sensibilizada é T_1. O disparo de T_1 resulta na marcação M_1=(0,1,1,0,0). Esta evolução da marcação da RP será daqui em diante representada com a seguinte notação: $M_0[T_1>M_1$.

Para a marcação M_1, há duas transições sensibilizadas: T_2 e T_3. Designando as marcações obtidas do disparo dessas transições por M_2 e M_3, respectivamente, tem-se: $M_1[T_2>M_3$=(0,0,1,1,0) e $M_1[T_3>M_3$=(0,1,0,0,1).

Para a marcação M_2, só a transição T_3 está sensibilizada. O seu disparo resulta na marcação M_4: $M_2[T_3>M_4$=(0,0,0,1,1).

Para a marcação M_3, só a transição T_2 está sensibilizada. O seu disparo resulta na marcação inicial M_0.

Com isto, já foram vistas todas as marcações possíveis da RP da Figura 4.6, partindo da marcação inicial M_0. O conjunto dessas marcações é $^*M_0$={M_0, M_1, M_2, M_3, M_4}. Estas marcações atingíveis desde M_0 estão representadas na Figura 4.7.

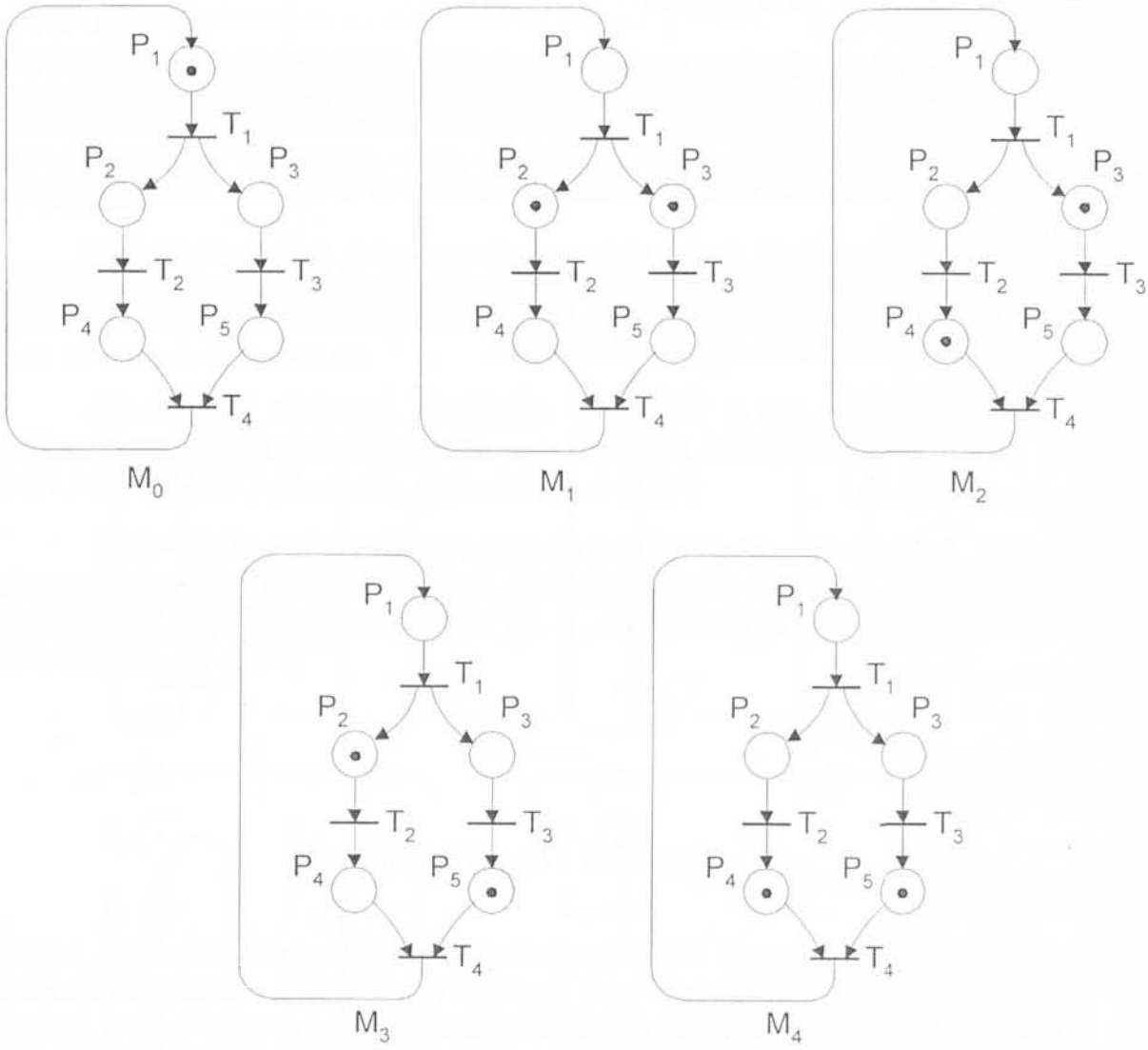


Figura 4.7 – Marcações atingíveis de uma RP.

Começando na marcação inicial M_0, T_1 e T_2 podem ser disparadas por esta ordem. Depois destes disparos, a marcação é M_2. Isto pode ser escrito como $M_0[T_1,T_2>M_2$. Se $S=T_1T_2$ representar uma sequência de disparo, então $M_0[S>M_2$. Isto significa que partindo da marcação M_0 se obtém a marcação M_2, através da sequência de disparo S.

Uma sequência de disparo é definida para uma dada marcação. É uma série de transições disparáveis em sequência, sem o disparo de outras transições. Por exemplo, a sequência T_1T_4 não é uma sequência de disparo para a marcação M_0, porque sendo disparada T_1, T_4 não fica sensibilizada e, portanto, não pode ser a primeira transição a ser disparada depois de T_1.

Diz-se que a marcação M_1 cobre a marcação M_2, se a marcação de cada posição P_i em M_1 é maior ou igual do que a sua marcação em M_2, para todas as posições P_i da RP. Diz-se que a marcação M_1 é maior que a marcação M_2, se existir na RP pelo menos uma posição P_i, cuja marcação em M_1 é maior do que a sua marcação em M_2.

4.5.1 RP limitada e RP segura

Uma posição P_i é limitada para uma marcação inicial M_0 se existir um inteiro k tal que, para todas as marcações atingíveis desde M_0, o número de marcas em P_i não excede k (diz-se que a posição P_i é k-limitada).

Uma RP é limitada para uma marcação inicial M_0, se todas as suas posições forem limitadas para M_0.

A Figura 4.8 apresenta uma RP limitada. Tem-se $M_0[T_1T_2>M_0$, etc. O conjunto de marcações atingíveis é $^*M_0=\{M_0, M_1, M_2, M_3\}$, e observa-se que as marcações de P_1 e de P_2 são limitadas. Assim, a RP é limitada (3-limitada).

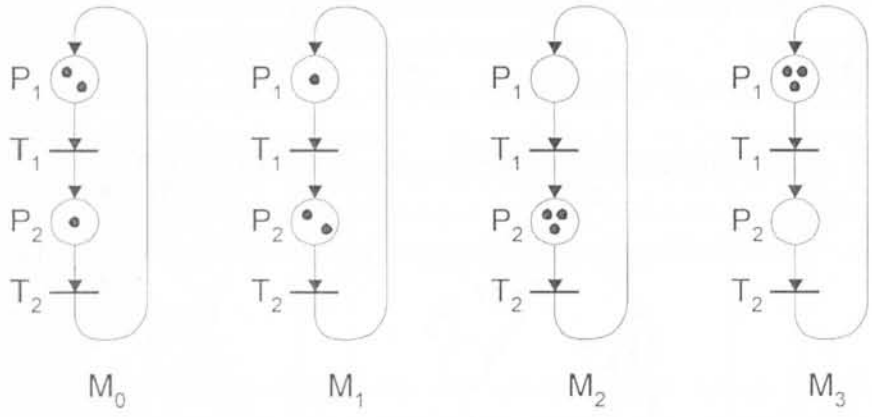


Figura 4.8 – Uma RP limitada.

A Figura 4.9 apresenta uma RP não limitada a partir de M_0. De facto, $M_0[T_1>M_1$, $M_1[T_1>M_2$, $M_2[T_1>M_3$, etc. Cada vez que a transição T_1 é disparada, é adicionada uma marca à posição P_2. As marcações atingíveis são infinitas em número, e a posição P_2 não é limitada.

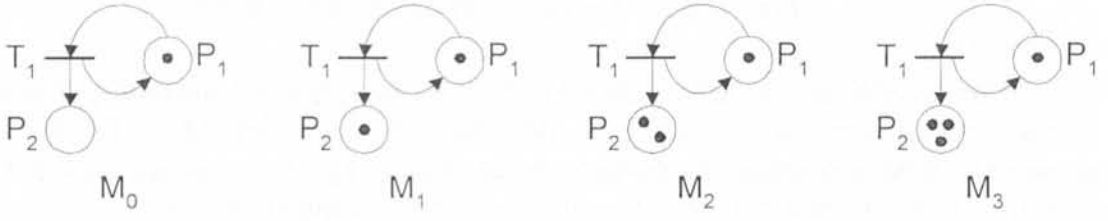


Figura 4.9 – Uma RP não limitada.

Uma RP é segura para uma marcação inicial M_0, se para todas as marcações atingíveis, cada posição contiver no máximo uma marca. Uma RP segura é obviamente uma RP limitada.

A RP da Figura 4.8 é limitada, mas não é segura. A RP da Figura 4.7 é segura, porque para qualquer marcação, há sempre zero ou uma marca em cada posição.

4.5.2 RP viva

A marcação de uma RP evolui através do disparo de transições. Quando determinadas transições não podem nunca mais ser disparadas, e quando a totalidade ou uma parte de uma RP não "funciona", este facto é um indicador de um problema de concepção do sistema modelizado.

Uma transição T_j é viva para uma marcação inicial M_0, se para todas as marcações atingíveis $M_i \in {}^*M_0$ existir uma sequência de disparo S a partir de M_i, que contém a transição T_j. Por outras palavras, a transição é viva se houver sempre possibilidade de a disparar, seja qual for o estado actual da RP.

Este conceito é ilustrado pela Figura 4.10. A RP a) tem duas marcações atingíveis. Para cada uma dessas marcações, existe uma sequência de disparo que contém T_1. De facto, $M_0[T_1>M_1$ e $M_1[T_2T_1>M_1$. Assim, a transição T_1 é viva.

A RP b) tem três marcações atingíveis a partir de M_0: ${}^*M_0=\{M_0, M_1, M_2\}$. Pode-se observar, que não há qualquer sequência de disparo para M_1 ou para M_2 que contenha T_1. Isto significa que se for atingida a marcação M_1, T_1 nunca ficará de novo sensibilizada. Por isso, esta transição não é viva.

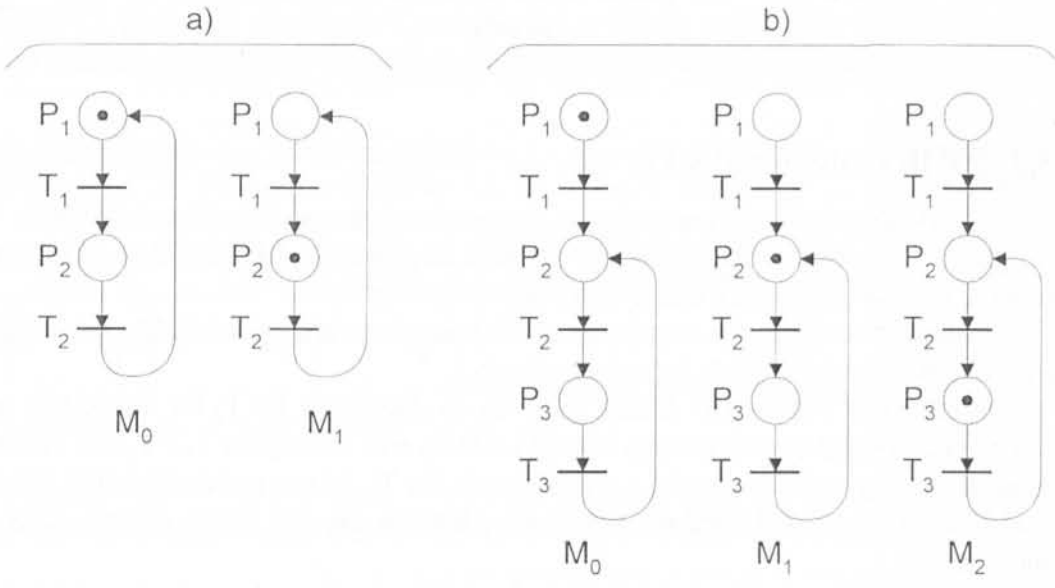


Figura 4.10 – Ilustração das propriedades transição viva e RP viva.

Uma RP é viva para uma marcação inicial M_0, se todas as suas transições forem vivas para M_0.

Uma transição T_j é quase-viva para uma marcação inicial M_0, se houver uma sequência de disparo para M_0, que contenha T_j. Uma RP é quase-viva, se todas as suas transições forem quase-vivas. Por outras palavras, uma transição é quase-viva se houver uma hipótese da transição poder ser disparada (ainda que isso só possa acontecer uma só vez). A transição T_1 da RP b) da Figura 4.10 será disparada uma só vez, antes de ficar permanentemente sem condições de ser disparada.

A RP a) da Figura 4.11 representa uma RP quase-viva. De facto, para cada transição existe uma sequência de disparo que a contenha, a partir da marcação inicial M_0 representada. $M_0[T_1>(0,1,0,0)$ e $M_0[T_2T_3T_4>(0,0,1,0)$. Contudo, existe o conflito $<P_1,\{T_1,T_2\}>$. Se T_1 for a primeira transição a ser disparada, as restantes transições nunca mais ficarão sensibilizadas.

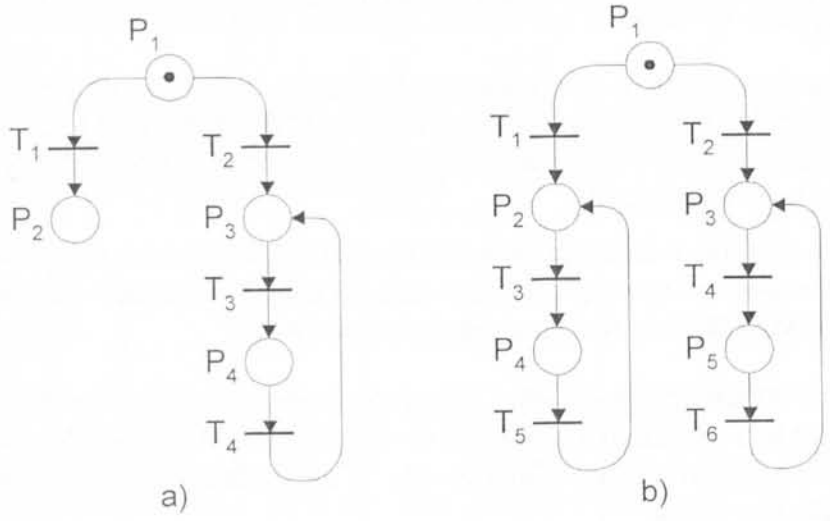


Figura 4.11 – RP quase-vivas: (a) Com um deadlock; (b) Livre de deadlocks.

4.5.3 RP livre de deadlocks

Um *deadlock* é uma marcação em que nenhuma transição está sensibilizada. Na RP a) da Figura 4.11, o disparo da transição T_1 resulta na marcação $M_1=(0,1,0,0)$. Esta constitui um *deadlock*, porque a marcação da RP não pode evoluir mais.

Uma RP é livre de *deadlocks* para uma marcação inicial M_0, se nenhuma das marcações atingíveis $M_1 \in {}^*M_0$ for um *deadlock*.

A RP b) da Figura 4.11 é uma RP livre de *deadlocks*. Se T_1 for a primeira transição a ser disparada, obtém-se a marcação $M_1=(0,1,0,0,0)$, e as transições T_3 e T_5 são vivas para esta marcação. Se a primeira transição a ser disparada for T_2, há um comportamento simétrico. Por outras palavras, uma RP livre de *deadlocks* é aquela em que existe sempre uma parte que "funciona".

As propriedades quase-viva e livre de *deadlocks* são independentes. Ambas as RP da Figura 4.11 são quase-vivas. Contudo, a RP a) tem um *deadlock* e a RP b) é livre de *deadlocks*.

As propriedades viva e livre de *deadlocks* dependem claramente da marcação inicial. Se a marcação inicial da RP a) da Figura 4.10 for zero (nenhuma marca em qualquer uma das suas posições), tem-se um estado de *deadlock*, porque nenhuma transição está sensibilizada, apesar desta RP ser viva para uma marcação inicial, que contenha pelo menos uma marca. Diz-se que uma RP não marcada é estruturalmente viva, se houver uma marcação inicial finita M_0, para a qual a RP marcada é viva. Por exemplo, as RP da Figura 4.11 não são estruturalmente vivas.

4.5.4 RP com conflitos efectivos e conflitos efectivos persistentes

O conceito de conflito estrutural de uma RP já foi apresentado em 4.4.3. Nesta secção, serão apresentados os conceitos de conflito efectivo e de persistência. Lembre-se que um conflito estrutural corresponde a um conjunto de pelo menos duas transições, que têm uma posição de entrada comum. Isto é representado por $K=<P_i,\{T_1,T_2,...\}>$.

Um conflito efectivo consiste na existência de um conflito estrutural K e de uma marcação M, tais que o número de marcas em P_i é mais pequeno que o número de transições de saída de P_i, que estão sensibilizadas por M. Um conflito efectivo é representado por $K^e=<P_i,\{T_1,T_2,...\},M>$. Este conceito é ilustrado pela Figura 4.12.

No caso a), tem-se o conflito estrutural $K_1=<P_1,\{T_1,T_2\}>$. A marcação sensibiliza T_1 e T_2, mas uma vez que só existe uma marca em P_1, há um conflito efectivo $K^e=<P_1,\{T_1,T_2,...\},M>$.

No caso b), a marcação actual apenas sensibiliza a transição T_1, não havendo por isso qualquer conflito efectivo (apesar de existir um conflito estrutural).

No caso c), ambas as transições estão sensibilizadas, mas há duas marcas em P_1. Neste caso, não há conflito efectivo, porque ambas as transições podem ser disparadas. Contudo, é de notar que após o disparo de uma das transições, haverá um conflito efectivo.

No caso d), apesar da existência de um conflito efectivo, a estrutura é particular. De facto, a marcação apenas torna possível o disparo de uma das transições. Mas, quer seja disparada T_1, quer seja disparada T_2, continuará a existir uma marca em P_1, mantendo-se o conflito efectivo. Este tipo de situação é denominado por conflito efectivo persistente.



Figura 4.12 – RP com conflitos.

Uma RP é isenta de conflitos efectivos para uma marcação inicial M_0, se para todas as marcações atingíveis $M_i \in {}^*M_0$ não houver conflitos efectivos.

Uma RP é persistente para uma marcação inicial M_0, se para todas as marcações atingíveis $M_i \in {}^*M_0$ for válida a seguinte propriedade: se T_j e T_k são transições sensibilizadas por uma marcação M_i, então T_jT_k é uma sequência de disparo a partir de M_i (assim como T_kT_j por simetria).

4.5.5 Invariantes de uma RP

Esta secção faz uma breve introdução ao conceito de invariante de uma RP e da sua utilidade na caracterização do comportamento da RP. Uma apresentação mais detalhada sobre

o assunto pode ser consultada em [26] e em muitas outras referências relativas a técnicas de análise sistemáticas das propriedades de RP.

Começando numa marcação inicial, a marcação de uma RP pode evoluir como consequência do disparo de transições. Se não houver *deadlocks*, o número de disparos possíveis é ilimitado. Contudo, nem todas as marcações podem ser atingidas, e nem todas as sequências de disparo podem ser disparadas. Os invariantes permitem detectar propriedades das marcações atingíveis, independentemente da evolução do estado da RP.

Os invariantes podem manifestar-se sob duas formas:

- Componente conservativa – um invariante de marcação (da expressão inglesa *marking invariant*) é obtido, se numa RP a soma das marcas existentes num subconjunto das suas posições for constante, para qualquer das marcações atingíveis;
- Componente repetitiva – resulta do facto de haver na evolução do estado de uma RP uma sequência de disparo repetitiva, constituída por um subconjunto das suas transições; esta sequência de disparo repetitiva pode levar a RP a passar periodicamente por um mesmo estado (marcação), ou a ver a marcação de um subconjunto das suas posições aumentada gradualmente; este tipo de invariante é habitualmente designado por invariante de disparo (da expressão inglesa *firing invariant*).

4.6 Técnicas de análise das propriedades de uma RP

Esta secção faz uma apresentação introdutória sobre as principais técnicas de análise das propriedades de RP. Essas técnicas permitem analisar uma RP de forma sistemática e classificá-la no que diz respeito às propriedades definidas em 4.5.

As técnicas conhecidas dividem-se em três grandes grupos:

- Desenho do grafo de marcações ou da árvore de cobertura,
- Métodos baseados em álgebra linear e
- Métodos de redução.

Apesar destas técnicas serem poderosas e sistemáticas, a sua aplicabilidade fica comprometida para RP de média ou grande dimensão, devido ao crescimento exponencial do número de marcações atingíveis, dada uma marcação inicial.

4.6.1 Grafo de marcações e árvore de cobertura

Na modelização de sistemas, por vezes é necessário apurar se um determinado estado pode ser atingido, partindo de um estado inicial. Este problema pode ser resolvido recorrendo à construção do grafo de marcações.

A construção do grafo de marcações consiste em analisar as marcações atingíveis para uma RP, dada uma marcação inicial. No início da secção 4.5, foi feita essa análise para a RP da Figura 4.6. Dessa análise foi possível concluir que as marcações atingíveis para aquela RP são as que estão ilustradas na Figura 4.7. Esta informação pode ser representada num grafo,

cujos nós são as marcações atingidas, e cujos arcos indicam a transição disparada que dá origem à evolução da marcação.

Por vezes, este grafo contém ciclos, que eventualmente podem ser infinitos. Para tornar a representação possível ou mais compacta, é habitual construir-se uma árvore de cobertura. Neste caso, esses ciclos são representados por um par de arcos ligando as marcações que fazem parte do ciclo.

Contudo, mesmo assim este problema pode-se tornar intratável, porque o grafo de cobertura pode ser muito grande, mesmo para RP razoavelmente pequenas. Este facto torna inviável a utilização desta técnica de análise em RP mais complexas, com um elevado número de estados. Em particular, para RP não limitadas, a construção do grafo de cobertura é impossível, devido ao número infinito de estados.

Por vezes, o grafo de marcações é designado por espaço de estados e a árvore de cobertura é designada por espaço de estados condensado.

4.6.2 Métodos baseados em álgebra linear

O comportamento de qualquer RP autónoma pode ser formalizado matematicamente, recorrendo à álgebra linear. Dessa forma, definindo um vector de posições, um vector de transições, uma matriz de incidência de entrada com a informação relativa às posições de entrada de cada transição, e uma matriz de incidência de saída com a informação relativa às posições de saída de cada transição, é possível obter uma representação formal da estrutura da RP. Com base na equação matricial da RP, é possível calcular a marcação resultante da execução de uma sequência de disparo de transições, para uma dada marcação inicial.

Esta metodologia é poderosa e elegante. Com base nela, foram desenvolvidas técnicas de análise para a determinação de invariantes de marcação e de invariantes de disparo. O seu formalismo matemático torna directa a realização de programas de computador, que implementem essas técnicas de análise de RP.

Mais uma vez, a aplicação destes métodos em ferramentas de análise de RP torna-se não fazível, quando o tamanho das RP aumenta.

4.6.3 Métodos de redução

Conforme foi referido em 4.6.1 e em 4.6.2, a aplicabilidade de técnicas sistemáticas de análise de RP é fortemente condicionada pelo tamanho das redes.

Numa tentativa de alargar a aplicabilidade desses métodos, foram desenvolvidos métodos de redução, que permitem transformar uma RP numa RP mais simples, mas mantendo determinadas propriedades da RP inicial.

Esses métodos dividem-se em dois grandes grupos:

- Métodos que mantêm as propriedades de RP viva e de RP limitada – são métodos que se baseiam na substituição de posições ou de transições;
- Métodos que mantêm os invariantes – são métodos eficientes que permitem obter todas as componentes conservativas da RP inicial.

4.7 Comportamentos dos sistemas discretos facilmente identificáveis através de RP

Uma das grandes vantagens das RP é a sua grande capacidade de representar graficamente certos comportamentos e de visualizar certos conceitos.

Esta secção tem por objectivo identificar estruturas encontradas frequentemente na modelização de sistemas com RP, que definem comportamentos particulares, tais como: sincronização, partilha de recursos ou exclusão mútua, memorização, leitura, escolha, sequência, concorrência, circuito ou auto-ciclo, etc.

O conhecimento destas estruturas é muito importante, particularmente na modelização de sistemas de fabrico, onde é possível encontrar todos aqueles tipos de comportamento [5].

4.7.1 Sincronização

A sincronização entre duas condições pode ser identificada, através da localização de duas ou mais posições, ligando a uma transição comum (ver Figura 4.13). Mesmo que uma posição receba uma marca, a transição tem de esperar que todas as outras posições de entrada recebam também uma marca, antes de poder ser disparada.

No contexto dos sistemas de fabrico, a chegada de uma peça a uma máquina e a disponibilidade da mesma, é um exemplo de duas condições que têm de se verificar para se gerar um evento do tipo "início de maquinação". A sincronização de várias actividades permite reduzir os tempos de espera e assim maximizar a taxa de produção.

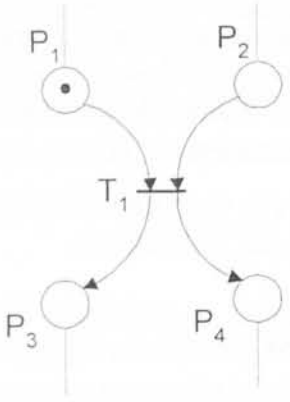


Figura 4.13 – Modelização de sincronização com uma RP.

4.7.2 Partilha de recursos ou exclusão mútua

A exclusão mútua é uma estrutura que permite modelizar a partilha de recursos entre vários processos ou máquinas. Por exemplo, um robô pode ser partilhado por duas máquinas para carregar ou descarregar. A Figura 4.14 representa um exemplo de exclusão mútua, em que dois processos paralelos partilham um mesmo recurso, cuja disponibilidade é representada pela existência de uma marca na posição P_1.

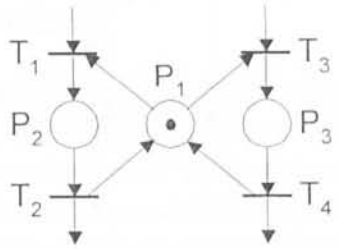


Figura 4.14 – Modelização da partilha de um recurso com uma RP.

4.7.3 Memorização

No caso a) da Figura 4.15, a posição P_1 memoriza o facto da transição T_1 ter sido disparada. A existência de uma marca em P_1 autoriza o futuro disparo da transição T_2, desde que as outras condições de disparo sejam também satisfeitas. Este tipo de estrutura permite memorizar a ocorrência de um evento.

Também é possível memorizar um número, através da presença de várias marcas numa posição (caso b) da Figura 4.15). Isto poderá significar, por exemplo, o número de peças existentes num *buffer* de uma sistema de fabrico.

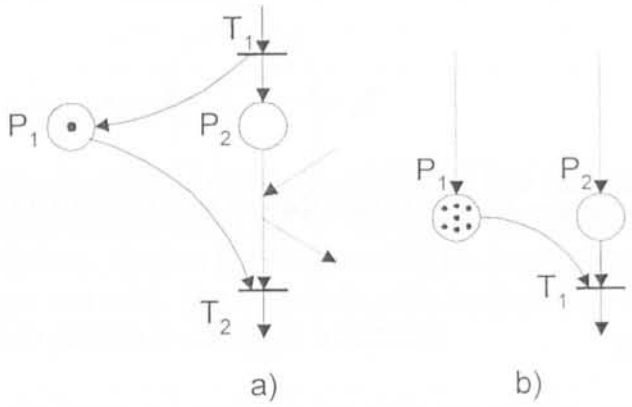


Figura 4.15 – Modelização da memorização com uma RP: a) De um disparo; b) De um número

4.7.4 Leitura

Na Figura 4.16, o disparo de T_1 é condicionado pela marcação de P_1, sem contudo modificar a sua marcação. Neste tipo de situações diz-se que é feita uma leitura.

Num sistema de fabrico dotado com um SAGVs, a chegada de uma peça à fila de saída de uma máquina pode ser representada por uma marca em P_1. Esta informação pode ser utilizada para desencadear a chamada de um AGV para transportar essa peça, sem contudo consumir essa marca. A marca só será consumida, quando o AGV chegar ao ponto de carga e carregar a peça.

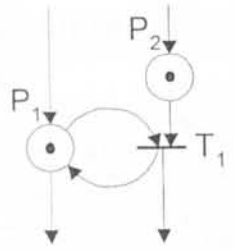


Figura 4.16 – Modelização da leitura numa RP.

4.7.5 Escolha

Quando há mais do que uma máquina disponível para processar uma peça, que acabe de chegar a uma célula de fabrico, é necessário escolher qual a máquina para onde a peça deve ser despachada.

No contexto das RP, uma posição com mais do que um arco de saída representa várias alternativas de evolução do estado do sistema (ver Figura 4.17). A escolha da transição a disparar tem de ser basear num critério, podendo ser aleatória ou basear-se na definição de prioridades para as transições em conflito.

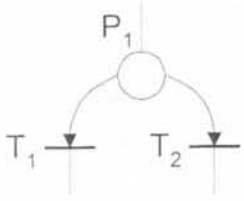


Figura 4.17 – Modelização de uma escolha numa RP.

4.7.6 Sequência

Uma estrutura comum em sistemas discretos é a sequência de eventos. Por exemplo, sequências *carrega_peça, processa_peça, descarrega_peça* são muito comuns nos sistemas de fabrico. Na modelização com RP pode ser utilizada uma série de posições e de transições, para modelizar sequências de eventos (ver Figura 4.18).

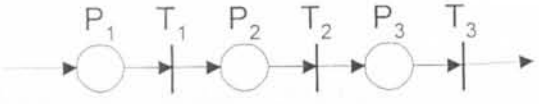


Figura 4.18 – Modelização de uma sequência de eventos numa RP.

4.7.7 Paralelismo e concorrência

Nos sistemas de fabrico modernos, há muitas operações que têm de ser executadas simultaneamente, para se optimizar a eficiência. A concorrência pura de dois eventos pode ser representada numa RP por duas transições independentes. Um caso muito comum é a existência de duas ou mais sequências, representando operações executadas em paralelo, com uma transição inicial e uma transição finais de sincronização (ver Figura 4.19).

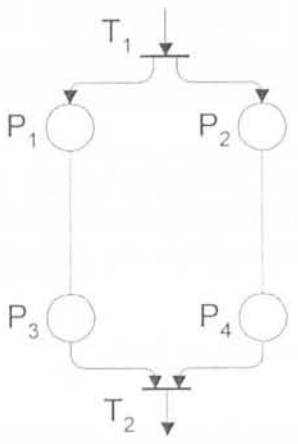


Figura 4.19 – Modelização de paralelismo e de concorrência numa RP.

4.7.8 Circuito ou auto-ciclo

Na prática, muitos sistemas ou os seus componentes têm um comportamento cíclico. A Figura 4.20 ilustra a representação com uma RP de um circuito simples (à esquerda) e de um ciclo (à direita). Um circuito simples pode representar de forma simplificada as actividades repetitivas realizadas por um robô. Um auto-ciclo pode representar a disponibilidade de uma máquina, ou o seu estado quando se encontrar a trabalhar.

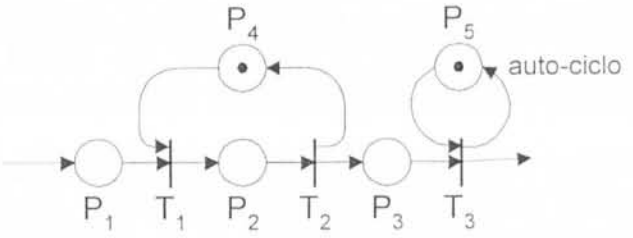


Figura 4.20 – RP com um circuito simples e um auto-ciclo.

5. *Redes de Petri não autónomas*

No capítulo anterior foram apresentados os conceitos básicos sobre as RP autónomas, que são vocacionadas para a análise qualitativa dos sistemas modelizados. Neste capítulo, serão apresentadas extensões das RP, que permitem não só descrever o que acontece, mas também quando acontece. Estas RP permitem modelizar os sistemas, sincronizando o disparo das transições com a ocorrência de eventos externos e/ou com a passagem do tempo. Por isso, são denominadas RP não autónomas.

Graças ao desenvolvimento destas extensões, que adicionaram novas regras de funcionamento, as RP passaram a permitir a construção de modelos mais ricos e compactos, e a tratar um maior número de aplicações [5].

5.1 RP sincronizadas

Numa RP autónoma sabe-se que uma transição pode ser disparada se estiver sensibilizada, mas não se sabe quando será disparada. Numa RP sincronizada, há um evento associado a cada transição. A transição será disparada se estiver sensibilizada e quando o evento associado ocorrer. Os eventos externos correspondem a mudanças de estado no mundo externo, por oposição aos eventos internos, que correspondem a mudanças de estado do sistema modelizado (mudanças na marcação da RP).

Os três pequenos exemplos da Figura 5.1 ilustram os conceitos essenciais de uma RP sincronizada.

No caso a), o evento externo E^3 está associado à transição T_1. Esta transição está sensibilizada e receptiva ao evento E^3, uma vez que existe uma marca em P_1, mas só poderá ser disparada quando ocorrer o evento E^3. Nas RP autónomas, uma transição sensibilizada também está em condições de ser disparada. Nas RP sincronizadas, não basta que esteja sensibilizada, sendo também necessário que ocorra o evento associado.

No caso b), a transição T_2 está receptiva ao evento E^1, porque está sensibilizada. É disparada quando E^1 ocorre. Por outro lado, a transição T_3, que tem o mesmo evento associado, não é disparada quando E^1 ocorre, porque não está sensibilizada, ou seja, não está receptiva ao evento E^1.

No caso c), a transição T_4 está receptiva ao evento E^2, porque está sensibilizada. É disparada quando E^2 ocorre, mas só é disparada uma vez, apesar de haver duas marcas em P_6 quando E^2 ocorre. Este tipo de comportamento das RP sincronizadas corresponde a uma opção tomada pelos investigadores que as criaram. Outra alternativa seria a transição T_4 poder ser disparada duas vezes consecutivas, aquando da ocorrência de E^2, dando lugar à passagem de duas marcas de P_6 para P_7. Contudo, esta abordagem é menos lógica do que a primeira, porque normalmente à ocorrência de um evento, corresponde apenas o desencadear de uma acção, e não o desencadear da mesma acção duas ou mais vezes consecutivas.

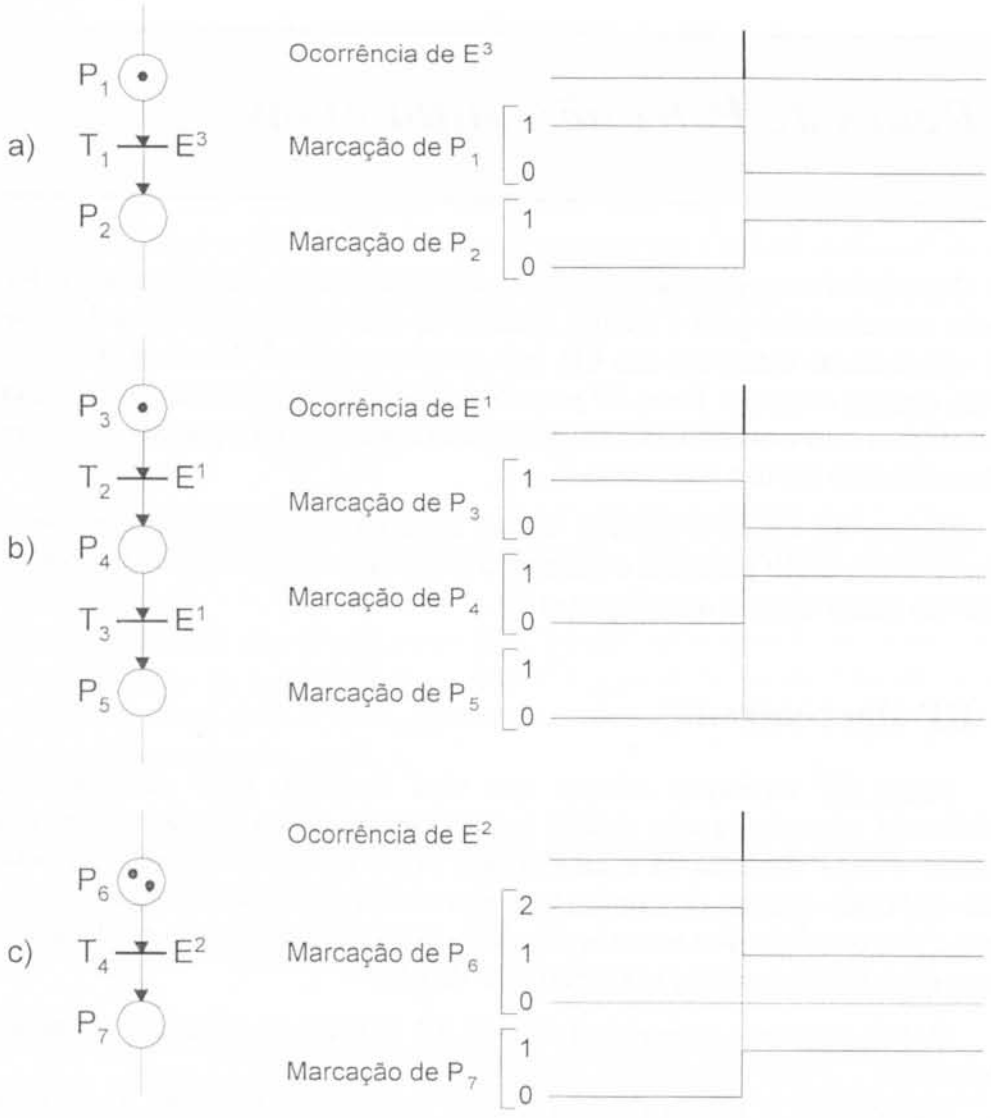


Figura 5.1 – Disparo de uma transição sincronizada.

5.1.1 Relação entre as propriedades de uma RP sincronizada e da RP autónoma equivalente sem sincronização

O conhecimento da relação existente entre as propriedades de uma RP sincronizada e da mesma RP sem sincronização é importante, porque a maioria dos métodos de análise das propriedades de RP foram desenvolvidos para as RP autónomas. O conhecimento dessas relações permite, sempre que possível, inferir as propriedades da RP sincronizada, a partir da análise da RP autónoma, utilizando aqueles métodos.

A condição da RP autónoma ser limitada para uma marcação inicial M_0 é suficiente mas não necessária para que a RP sincronizada seja limitada para a mesma marcação inicial.

A condição da RP autónoma ser viva para uma marcação inicial M_0 não é nem necessária nem suficiente para que a RP sincronizada seja viva para a mesma marcação inicial.

Se a RP autónoma for viva para uma marcação inicial M_0, então todas as RP totalmente sincronizadas, construídas a partir dessa rede, são também vivas para M_0.

5.2 RP temporizadas

Uma RP temporizada permite modelizar sistemas cujo funcionamento seja dependente do tempo. Por exemplo, permite modelizar uma operação, que tem associado um determinado intervalo de tempo não nulo entre o seu início e o seu fim.

As RP temporizadas tornam possível a avaliação do desempenho dos sistemas, havendo dois grandes métodos de modelização:

- RP P-temporizadas – os tempos do sistema são associados às posições;
- RP T-temporizadas – os tempos do sistema são associados às transições.

De seguida, será descrito o princípio de funcionamento de ambas as variantes.

5.2.1 RP P-temporizadas

Nas RP P-temporizadas (da expressão inglesa *P-timed PN*), é possível associar a cada posição P_i um tempo d_i (pode ser ou não igual a zero). Este tempo pode ser constante, ou variável no caso mais geral. As RP P-temporizadas foram definidas com tempos cujos valores são números racionais, para permitirem funcionamentos periódicos. Contudo, também é possível definir redes com tempos cujos valores sejam número reais.

5.2.1.1 Princípio de funcionamento de uma RP P-temporizada

Quando é depositada uma marca numa posição P_i, esta tem de permanecer nessa posição durante um tempo igual ou superior a d_i. Enquanto a marca não pode sair de P_i, diz-se que a marca está indisponível. Quando tiver passado um tempo igual a d_i, a marca fica disponível.

Este princípio de funcionamento é ilustrado pela Figura 5.2. Quando T_1 é disparada, é depositada uma marca na posição P_1. Essa marca permanece indisponível durante um tempo igual a d_1. Quando fica disponível, a transição T_2 é sensibilizada. Quando a transição T_2 é disparada (não necessariamente ao fim do tempo d_1), é depositada uma marca indisponível em P_2, etc.

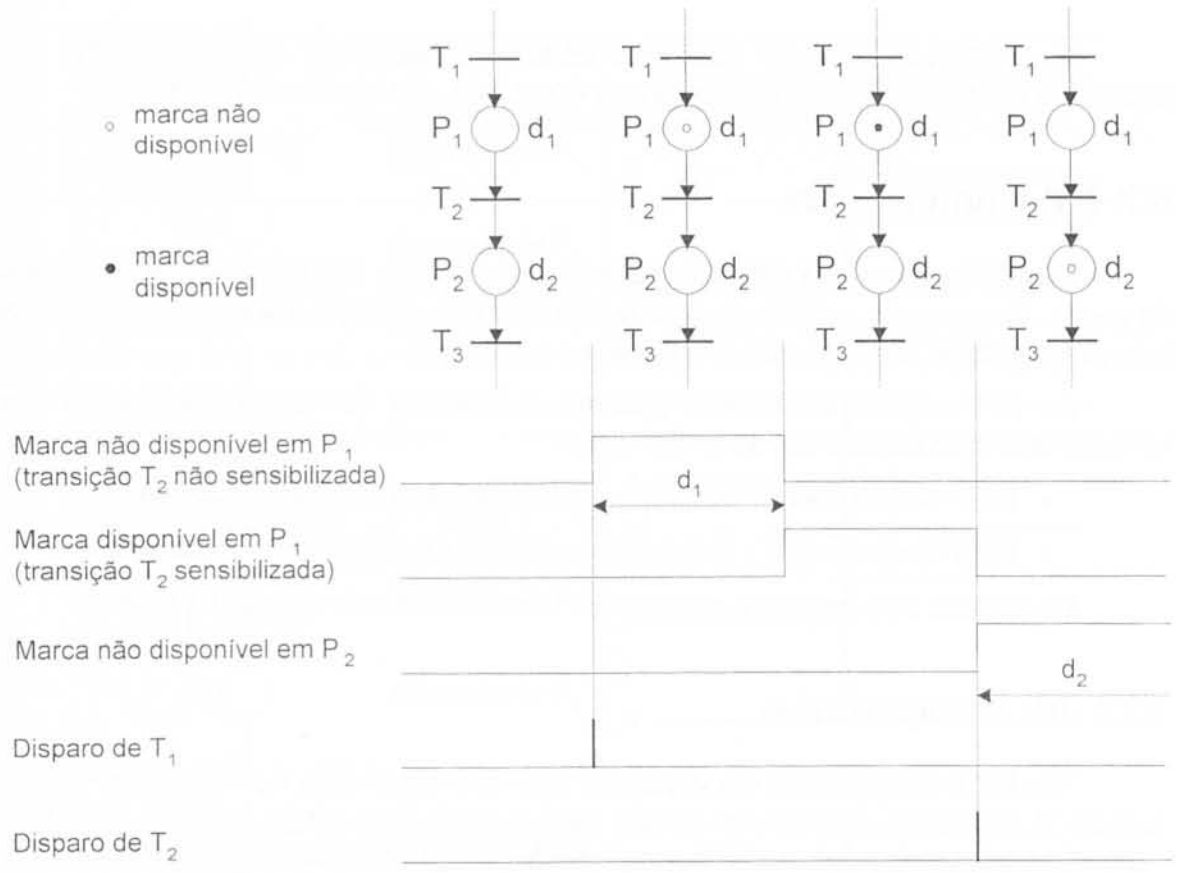


Figura 5.2 – Disponibilidade de uma marca numa RP P-temporizada.

5.2.1.2 Evolução da marcação de uma RP P-temporizada

No primeiro instante de funcionamento da RP, a marcação inicial M_0 é apenas constituída por marcas disponíveis. Em qualquer instante, a marcação M actual da RP é a soma de duas marcações M^a e M^u, sendo M^a a marcação apenas constituída por marcas disponíveis, e M^u a marcação apenas constituída por marcas indisponíveis.

Uma transição está sensibilizada para uma marcação $M=M^a+M^u$, se estiver sensibilizada para a marcação M^a. Quando uma transição é disparada, é retirada uma marca disponível de cada posição de entrada, e é depositada uma marca indisponível em cada posição de saída. Quando uma marca é depositada numa posição P_i, aquando do disparo da transição em t, essa marca permanecerá indisponível no intervalo de tempo $(t, t+d_i)$.

5.2.2 RP T-temporizadas

Nas RP T-temporizadas (da expressão inglesa *T-timed PN*), os tempos são associados às transições. Existem resultados teóricos que provam que os modelos P-temporizados e T-temporizados são equivalentes, sendo possível transitar de um para o outro (ver 5.2.3).

5.2.2.1 Princípio de funcionamento de uma RP T-temporizada

Neste tipo de redes, uma marca tem dois estados possíveis: pode estar reservada para o disparo de uma transição T_j, ou pode estar não reservada.

Este comportamento é ilustrado pelo exemplo da Figura 5.3. Quando a transição T_1 é disparada, é depositada uma marca na posição P_1, sensibilizando-se a transição T_2. Então, pode-se decidir disparar T_2 a qualquer momento. Quando é tomada essa decisão num instante t, a marca requerida pela realização do disparo é reservada, permanecendo nesse estado no intervalo de tempo $(t, t+d_j)$. No fim deste intervalo de tempo, a transição é efectivamente disparada. A marca reservada para o disparo de T_2 é removida de P_1, e é depositada uma marca não reservada em P_2, etc.

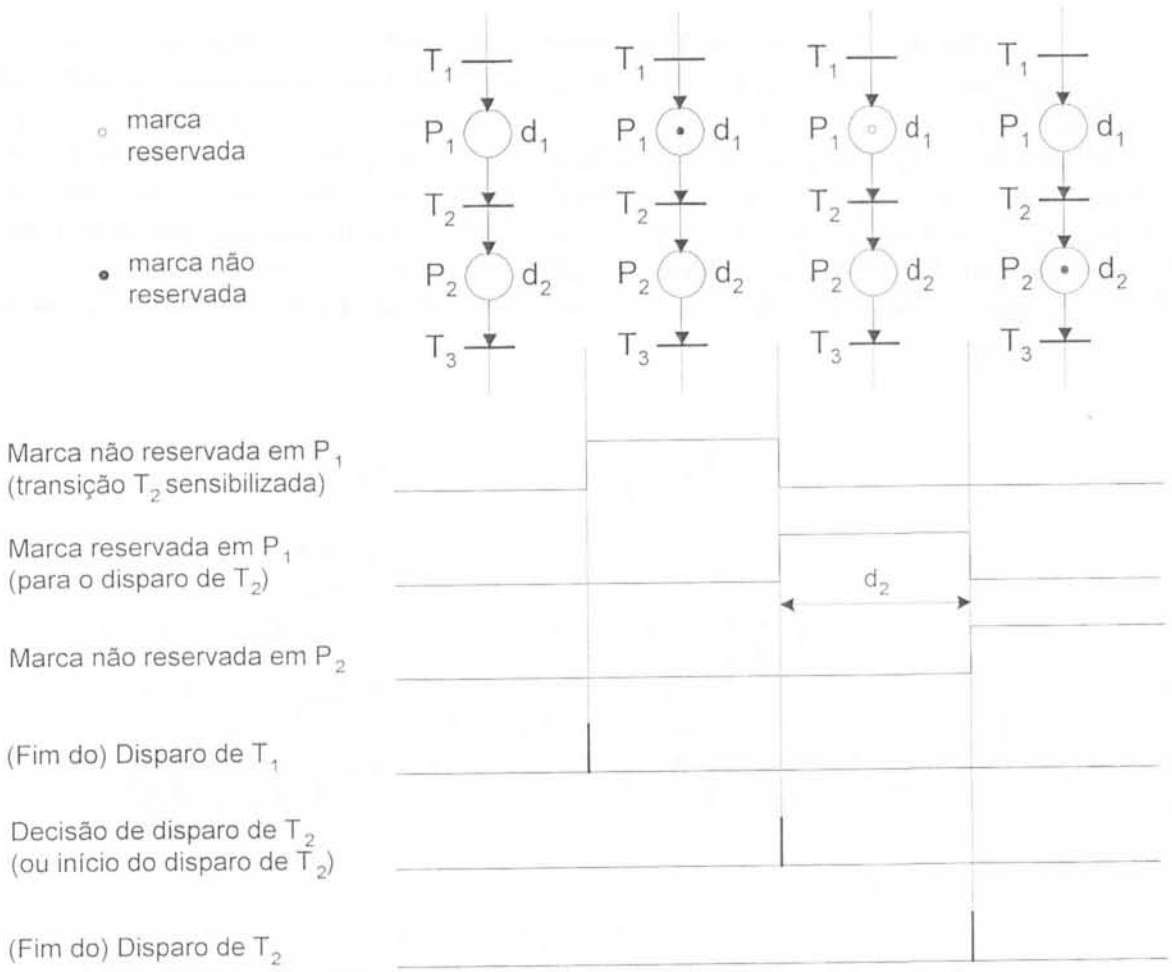


Figura 5.3 – Reserva de uma marca numa RP T-temporizada.

Seja t o instante em que é decidido o disparo da transição T_j e $t+d_j$ o instante em que o disparo é realmente efectuado em tempo nulo. O princípio de funcionamento da RP T-temporizada tem a vantagem de permitir considerar que o disparo de uma transição tem uma duração nula, o que é consistente com a definição de RP não autónoma, que considera o disparo de uma transição um evento indivisível (ver 4.2.4). Muitas vezes é utilizado um vocabulário mais sugestivo: t é o início do disparo, $t+d_j$ é o fim do disparo e, nesse intervalo de tempo, diz-se que as marcas reservadas "estão na transição".

5.2.2.2 Evolução da marcação de uma RP T-temporizada

Em qualquer instante, a marcação actual M é a soma de duas marcações M^r e M^n, em que M^r é a marcação apenas constituída por marcas reservadas, e M^n é a marcação constituída apenas por marcas não reservadas. Uma transição está sensibilizada para uma marcação $M=M^r+M^n$, se estiver sensibilizada para a marcação M^n.

A marcação de uma RP T-temporizada é feita de marcas reservadas e não reservadas. Uma transição T_j está sensibilizada, se houver marcas não reservadas nas suas posições de entrada em número suficiente para que isso aconteça. Só nesse caso, é que pode ser decidido o disparo da transição e ser efectuada a reserva das marcas necessárias.

5.2.3 Equivalência entre RP P-temporizadas e T-temporizadas

A Figura 5.4 ilustra a transformação de uma RP T-temporizada numa RP P-temporizada. A transição T_1 é substituída por uma sub-rede, constituída por duas transições T'_1 e T''_1 e por uma posição P'_1, de tal forma, que as posições de entrada de T'_1 são as posições de entrada de T_1 e as posições de saída de T''_1 são as posições de saída de T_1. O tempo associado à posição P'_1 é o mesmo tempo que estava associado à transição T_1. Este procedimento é seguido para todas as transições da RP T-temporizada, cujo tempo seja maior do que zero. Para todas as posições da RP T-temporizada inicial, $d_i=0$. A marcação inicial permanece inalterada para todas as posições da RP inicial e a marcação inicial das posições adicionadas é zero.

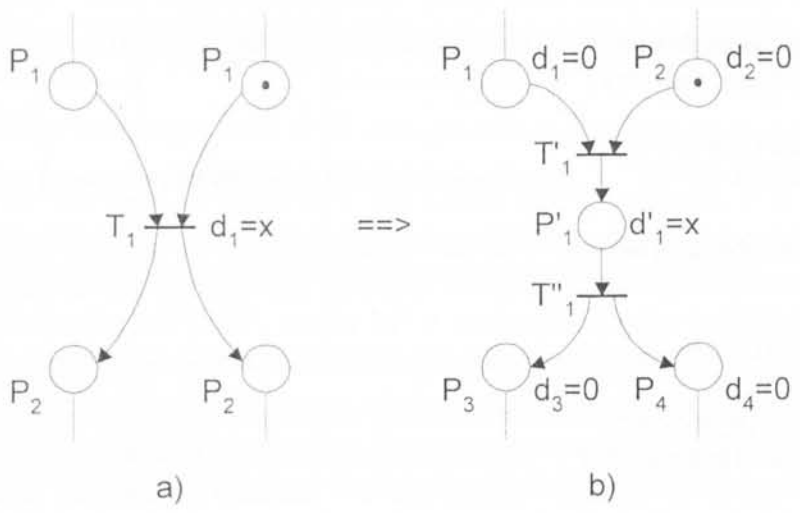


Figura 5.4 – Transformação de uma RP T-temporizada numa RP P-temporizada

A Figura 5.5 ilustra a transformação inversa de uma RP P-temporizada numa RP T-temporizada. A posição P_1 é substituída por uma sub-rede, constituída por duas posições P'_1 e P''_1 e por uma transição T'_1. As transições de entrada de P'_1 são as transições de entrada de P_1 e as transições de saída de P''_1 são as transições de saída de P_1. O tempo associado à transição adicionada é igual ao tempo que estava associado à posição P_1 na RP inicial. Para todas as transições T_j da RP inicial, $d_j=0$. A marcação inicial de P''_1 é a marcação inicial de P_1 e a marcação inicial de P'_1 é zero.

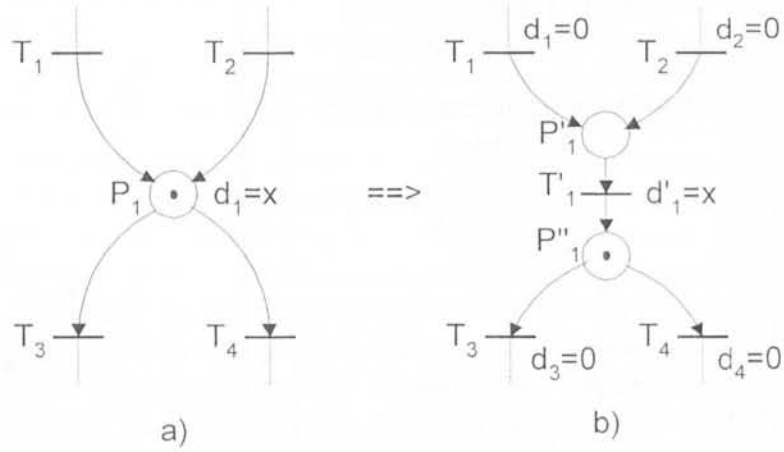


Figura 5.5 – Transformação de uma RP P-temporizada numa RP T-temporizada

De acordo com a definição de RP, que especifica um tempo de disparo nulo das transições, o modelo P-temporizado é mais lógico do que o modelo T-temporizado. Todavia, a utilização do segundo modelo é vantajosa no caso de cada operação, associada a uma posição com um tempo não nulo, ter apenas uma transição de entrada e uma transição de saída (ver Figura 5.4). Nestes casos, o modelo T-temporizado permite a obtenção de uma RP mais compacta.

5.3 RP interpretadas

Uma RP interpretada tem as seguintes características:

- É uma RP sincronizada.
- É uma RP P-temporizada.
- Tem uma parte de processamento de dados, cujo estado é definido por um conjunto de variáveis $V=\{V_1,V_2,...\}$. Este estado é modificado por operações $O=\{O_1,O_2,...\}$, que estão associadas às posições. O estado determina o valor das condições (denominadas predicados) $C=\{C_1,C_2,...\}$, que estão associadas às transições.

Na Figura 5.6 pode-se observar que o tempo d_i e a operação O_i estão associados com a posição P_i, e que o evento E_j e a condição C_j estão associados com a transição T_j.

Se a transição T_j está sensibilizada (se há pelo menos uma marca disponível em cada uma das posições de entrada) e se a condição C_i é verdadeira (se o predicado é verdadeiro), a transição T_j tem condições para ser disparada, o que ocorrerá aquando da ocorrência do evento E_j. Se for depositada uma marca na posição P_i no instante t, a operação O_i pode ser executada, e a marca permanece indisponível durante d_i.

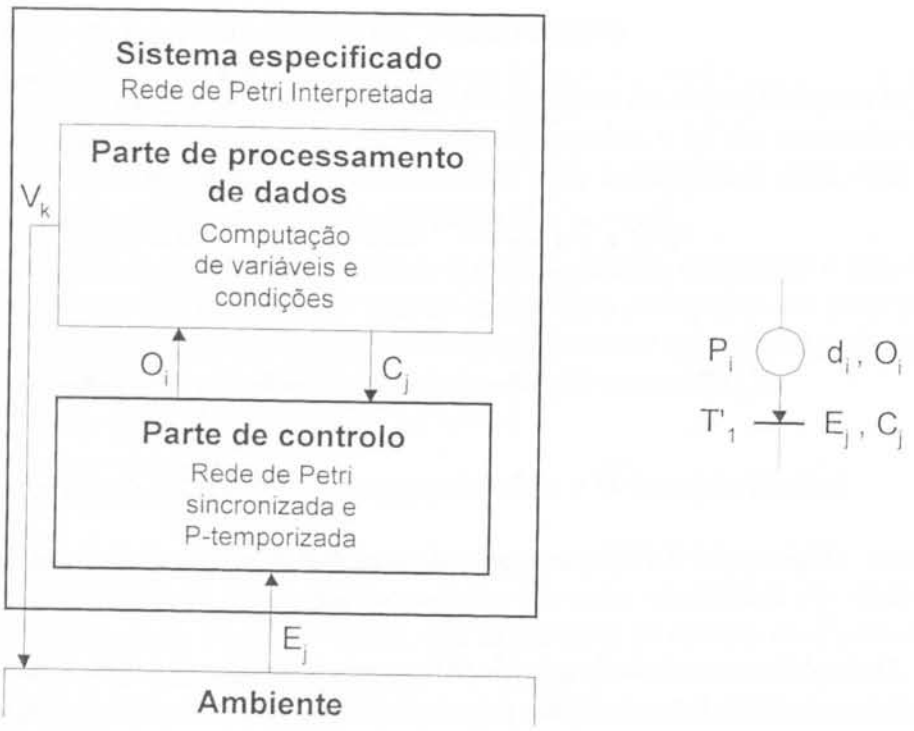


Figura 5.6 – RP interpretada.

5.4 RP estocásticas

Numa RP temporizada, o tempo associado a cada posição ou a cada transição é fixo. Estes modelos são adequados à análise de sistemas, nos quais as durações das operações são fixas. É o que se passa, por exemplo, em sistemas de produção onde os tempos de maquinação sofrem muito pequenas variações. Contudo, há fenómenos que não podem ser modelizados convenientemente com tempos constantes. É o caso, por exemplo, do tempo entre avarias de uma máquina, ou o tempo de execução de operações manuais. Estes tempos só podem ser modelizados à custa de variáveis aleatórias.

As RP estocásticas permitem associar ao tempo de disparo ou à taxa de disparo de uma transição uma variável aleatória. O caso mais frequente é a descrição dessas grandezas com base numa função de densidade de probabilidade exponencial.

Seja X uma variável aleatória, que segue uma função de densidade de probabilidade (FDP) exponencial, com uma taxa μ. A expressão da FDP exponencial é a Equação 5.1. O seu valor esperado é dado pela Equação 5.2 e a sua variância é dada pela Equação 5.3. A Equação 5.4 é a expressão da densidade de probabilidade acumulada.

$$h(t) = \mu \cdot e^{-\mu t} \tag{5.1}$$

$$E(X) = \frac{1}{\mu} \tag{5.2}$$

$$V(X) = \frac{1}{\mu^2} \tag{5.3}$$

$$H(t) = P(X \leq t) = 1 - e^{-\mu t} \tag{5.4}$$

A teoria subjacente às RP estocásticas resulta da combinação da teoria das RP autónomas e da teoria das filas de espera ou dos processos de Markov. Os resultados teóricos relativos a invariantes de marcação podem ser facilmente estendidos às RP estocásticas. O comportamento aleatório da RP estocástica é idêntico ao da cadeia de Markov equivalente, permitindo a aplicação dos métodos de uma cadeia de Markov na determinação das probabilidades dos estados da RP, em regime estacionário. Desta análise, podem ser obtidos índices de desempenho, tais como: a marcação média de uma posição em regime estacionário, a frequência média de disparo de uma transição, etc.

A Figura 5.7 representa do lado esquerdo uma estação de trabalho, constituída por dois servidores, um *buffer* de entrada e um *buffer* de saída. A estação produz peças, usando peças não processadas do *buffer* de entrada e depositando as peças processadas no *buffer* de saída.

O funcionamento do sistema pode ser descrito pela RP da Figura 5.7. Assume-se que o tempo de serviço de um servidor é uma variável aleatória, distribuída de acordo com uma lei exponencial, com um valor médio v_1. Usando este valor médio v_1, a taxa de disparo da transição T_1 é definida por $\mu_1=1/v_1$. Esta taxa de disparo significa que a probabilidade da transição T_1 disparar no intervalo (t, t+dt), sabendo que está sensibilizada no instante t, é μ_1.dt. A RP prevê a avaria de um servidor (transição T_2), sendo μ_2 a taxa de avarias e μ_3 a taxa de reparação de um servidor.

A marcação da RP significa que há um servidor avariado , um servidor a processar uma peça, três peças no *buffer* de entrada e uma peça acabada no *buffer* de saída.

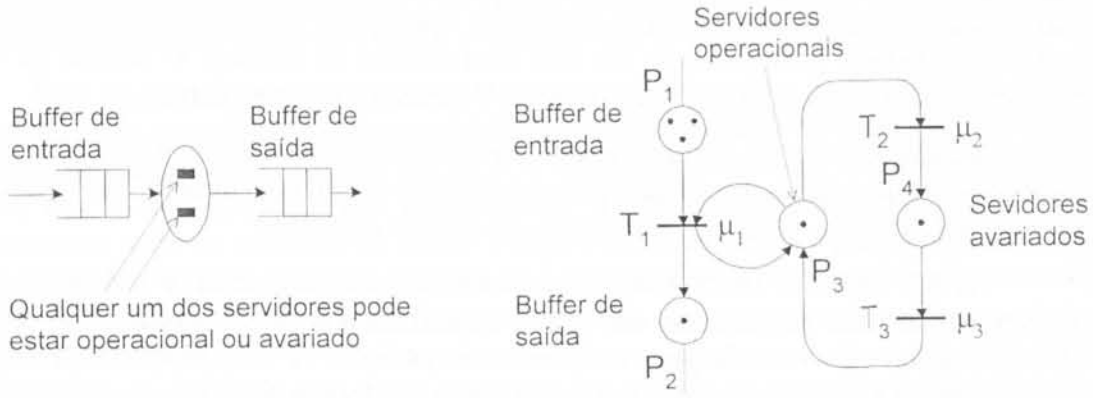


Figura 5.7 – Exemplo de uma RP estocástica.

5.5 RP coloridas

Os modelos baseados em RP têm provado serem adequados à descrição de sistemas, onde existam problemas de partilha de recursos, sincronização e concorrência.

É habitual encontrar nos sistemas entidades que tenham uma descrição comum. Estas têm de ser modelizadas pelo mesmo padrão de uma RP, que é repetido tantas quantas vezes o número de entidades idênticas. À medida que o número de padrões se torna grande, é fácil perceber que o modelo atinge um tamanho tal, que torna impossível a sua utilização.

Numa RP, a informação é baseada no estado (na marcação) das posições. A presença de uma marca numa posição pode modelizar, por exemplo, a disponibilidade de uma máquina. Quando essa posição não tem qualquer marca, isso significa que a máquina está ocupada. Várias marcas numa mesma posição podem representar um conjunto de peças idênticas armazenadas num *buffer*.

Para se enriquecer a informação associada a uma posição de uma RP, tem de se ser capaz de distinguir duas marcas existentes na mesma posição. Para isso, associa-se um identificador ou cor a cada marca, sendo a informação representada por um par (*posição, cor da marca*). A este novo tipo de RP dá-se o nome de RP colorida (CPN – *coloured Petri net*). Daqui em diante, as RP coloridas serão designadas através da sigla RPC. As RPC foram o primeiro tipo de RP de alto nível a ser definido, tendo como principais vantagens o aumento do nível de abstracção dos modelos e a possibilidade de se obterem modelos mais compactos e mais legíveis.

O formalismo original das RPC foi definido por Kurt Jensen – Professor do Departamento de Ciência dos Computadores, da Universidade de Aarhus, na Dinamarca – na sua tese de doutoramento, em 1980. O grupo CPN da Universidade de Aarhus é um dos centros mais importantes de desenvolvimento de RP de alto nível, de métodos de análise e de ferramentas de suporte às RPC. Em [27] são introduzidos os conceitos básicos da utilização das RPC na modelização de sistemas.

5.5.1 Apresentação intuitiva das RP coloridas

Numa RPC, cada transição pode ser disparada de várias maneiras, representadas pelas várias cores de disparo associadas à transição. A relação entre as cores de disparo e a marcação colorida em questão é definida por funções associadas aos arcos. A cor associada à marca pode ser um n-uplo, podendo transportar informação detalhada e complexa sobre uma dada entidade (e.g. parâmetros de um lote num sistema de fabrico). O disparo de transições pode dar origem ao desaparecimento de cores e à criação de novas cores.

A parte superior da Figura 5.8 representa dois sistemas idênticos. Em cada sistema, há um carrinho que se pode mover quer para a esquerda, quer para a direita. Logo que o carrinho chega a um extremo, inicia um movimento em sentido contrário.

As RP dos dois sistemas estão representadas na parte inferior da Figura 5.8. Cada posição destas redes representam um estado do sistema. Para cada carrinho, as posições P_l^b e P_l^y representam o movimento para a esquerda e as posições P_r^b e P_r^y representam o movimento para a direita. O estado inicial é o movimento para a esquerda de ambos os carrinhos.

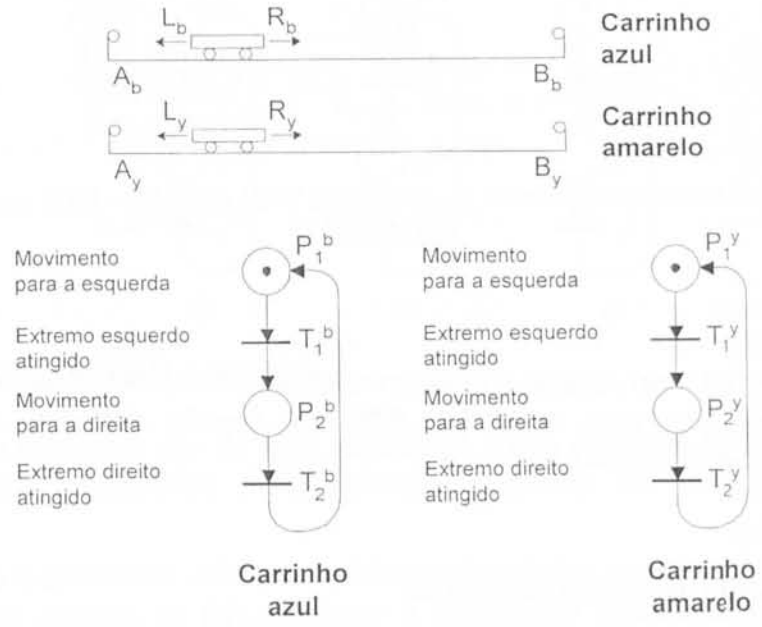


Figura 5.8 – Dois sistemas e suas RP correspondentes.

5.5.1.1 O conceito de cor

Observando o modelo da Figura 5.8, constata-se que este é constituído por duas RP idênticas, onde só muda a cor do veículo a que se refere cada uma delas: azul (b) ou amarelo (y). A pergunta que ressalta é se o sistema não pode ser descrito por uma única RP.

Se as duas RP forem fundidas numa só RP, obtém-se a RP a) da Figura 5.9. Esta rede não descreve correctamente o funcionamento dos dois carrinhos, porque o conhecimento do estado da RP é insuficiente para se saber o estado de cada um dos dois carrinhos. Por exemplo, quando as duas posições P_1 e P_2 estão marcadas, não se sabe qual é o carrinho que se está a mover para a esquerda e qual é o carrinho que se está a mover para a direita.

Para se modelizar correctamente o movimento de ambos os carrinhos, é necessário associar a cada uma das duas marcas existentes um identificador sobre o carrinho que cada uma delas representa. No contexto das RPC, a informação associada a uma marca denomina-se por cor. Neste exemplo, por coincidência, a informação associada a cada marca é precisamente a cor do carrinho em causa: b para o carrinho azul e y para o carrinho amarelo. As posições P_1 e P_2 podem então conter marcas coloridas, com cores pertencentes ao conjunto {b, y}. A transição T_1 está sensibilizada em relação à cor y, se houver uma marca y na posição P_1. O disparo de T_1 consiste em remover uma marca y de P_1 e em adicionar uma marca da mesma cor (y) à posição P_2 (RP b) da Figura 5.9). A transição T_1 também pode ser disparada com marcas de cor b. O conjunto de cores de disparo associado à transição T_1 é {b, y}.

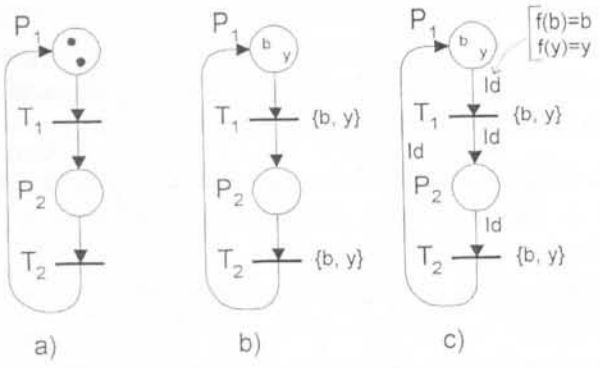


Figura 5.9 – (a) Modelo não descrevendo correctamente o funcionamento dos dois carrinhos. b) Modelo RPC sem funções explícitas. c) Modelo RPC com as funções dos arcos.

5.5.1.2 O conceito de função de um arco

Numa RPC há uma função associada a cada arco, que estabelece a relação entre a cor associada à transição, que é escolhida para a disparar (cor de disparo), e as cores associadas às marcas na posição correspondente (cor da marca).

Na RP c) da Figura 5.9, a função identidade, representada por 'Id', está associada a todos os arcos, uma vez que não há quaisquer transformações de cor. Isto significa que o disparo de uma transição em relação a uma dada cor, corresponde simplesmente em remover da posição de entrada uma marca da mesma cor e em adicionar à posição de saída uma marca da mesma cor.

O exemplo anterior apresenta a função mais simples que se pode encontrar nos arcos de uma RPC. Todavia, é possível definir funções mais complexas. Quando se observa o movimento de um carrinho, é possível identificar uma similaridade: logo que um carrinho chega a um dos extremos, a acção desencadeada é inverter o sentido do seu movimento. Com uma RPC, é possível representar o movimento em ambos os sentidos com apenas uma posição.

Para distinguir o estado do movimento de um carrinho, é necessário adicionar mais informação às marcas da RP. Para além da cor do veículo, é necessário adicionar um identificador do sentido do movimento, ou seja, uma cor do conjunto {r,l}: r significa "movimento para a esquerda" e l significa "movimento para a direita".

A Figura 5.10 mostra uma nova RP, que com apenas uma posição P_1, modeliza o movimento em ambos os sentidos de ambos os veículos, à custa da definição de marcas com uma cor, que consiste num tuplo com dois elementos: a cor do veículo e o sentido do movimento. As marcas podem ter qualquer uma das cores do conjunto {<b, l>, <b, r>, <y, l>, <y, r>}. Conforme é indicado na RP, a transição T_1 fica sensibilizada com a presença de uma marca, que pode ter associada qualquer cor pertencente ao conjunto anterior. O arco de entrada da transição não realiza qualquer transformação de cor na marca recebida pela transição, aquando do seu disparo. Por outro lado, após o disparo de T_1, esta devolve uma marca a P_1, não mudando a primeira componente da cor (a cor do veículo em causa), e invertendo o sentido do movimento do veículo, ou seja, transformando a segunda componente de r para l ou de l para r. Esta transformação é indicada pela função f do arco que liga T_1 a P_1.

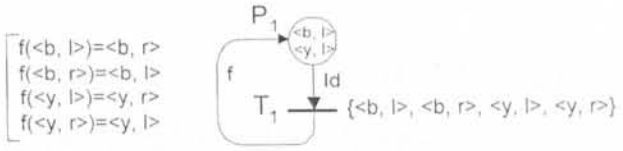


Figura 5.10 – Modelo RPC compacto do funcionamento de ambos os carrinhos.

5.5.2 Extensões das RP coloridas

Todas as extensões das RP não autónomas e não coloridas (RP sincronizadas, RP temporizadas, RP interpretadas e RP estocásticas) podem ser combinadas com as RP coloridas.

Para isso é necessário: adicionar a informação relativa à cor das marcas à informação associada a uma posição da RP; adicionar a informação relativa às cores de disparo à informação associada às transições da RP não colorida.

5.6 RP orientadas a objectos

Nos últimos anos, tem sido feito um esforço significativo no sentido de se aplicar nas RP o paradigma da orientação a objectos [28].

Nalguns casos, tentou-se permitir a estruturação orientada a objectos dos modelos baseados em RP. Segundo esta filosofia, uma RP é um objecto que pode ser instanciado, herdado e reutilizado em modelos mais complexos [29] [30] [31] [32] [33].

Noutros casos, tentou-se definir um formalismo orientado a objectos, que permitisse a utilização da tipos de dados abstractos, para definir os tipos das marcas (a cor das marcas). Nesta abordagem, uma marca é um objecto que encapsula dados e funções para manipular os seus dados [34].

Tal como aconteceu com as linguagens de programação, no âmbito das RP tem-se tentado ao longo dos tempos aumentar o seu nível de abstracção, tornando os modelos mais intuitivos e reutilizáveis.

6. *Ferramentas de suporte à modelização baseada em Redes de Petri*

As RP são uma poderosa metodologia de análise de sistemas discretos. Na sua forma mais simples (RP autónomas), permitem a descrição qualitativa dos sistemas. Por ser um formalismo passível de ser representado graficamente, permite visualizar facilmente comportamentos intrínsecos dos sistemas (concorrência, sincronização, partilha de recursos, existência de erros de concepção, *deadlocks*, detecção de gargalos nos sistemas, etc). Para além disso, as RP permitem a sua formalização matemática, através da álgebra linear. Esta importante característica torna as RP uma metodologia rigorosa e sistemática, permitindo o desenvolvimento de técnicas de análise baseadas na álgebra linear, as quais permitem determinar propriedades da RP e, consequentemente, do próprio sistema modelizado.

Com o desenvolvimento de extensões às RP autónomas, através da introdução da interacção do comportamento das RP com a ocorrência de eventos externos ou com a passagem do tempo (RP não autónomas), as RP tornaram-se uma ferramenta ainda de maior aplicabilidade. Estes desenvolvimentos permitem a descrição quantitativa e formal dos sistemas discretos reais (sistemas operativos, sistemas de fabrico, etc), bem como a análise do seu desempenho, durante a fase da sua concepção, graças à introdução do factor tempo.

Com a definição das RP coloridas e das RP de alto nível, passou a ser possível a construção de modelos mais compactos e o tratamento de problemas mais complexos, sem pôr em risco a legibilidade e a utilidade dos modelos de sistemas discretos baseados em RP. A possibilidade de definição de tipos de dados abstractos para as marcas da RP colorida, e a introdução de técnicas orientadas a objectos na definição de modelos baseados em RP, tem aumentado progressivamente o nível de abstracção permitido pelas RP, tornando os modelos mais poderosos, legíveis e reutilizáveis.

Todo este potencial oferecido pelo paradigma das RP só poderá ser concretizado na sua aplicação na resolução de problemas reais, se a sua utilização for suportada no desenvolvimento de ferramentas de *software*, que permitam, entre outras funcionalidades:

- a utilização de uma interface gráfica para a criação de RP coloridas, temporizadas e estocásticas;
- a possibilidade de simulação dos modelos com animação gráfica de *tokens* (designação vulgarmente utilizada para designar as marcas de uma RP);
- a compilação de informação estatística sobre o desempenho do sistema modelizado e a possibilidade de exportar essa informação e permitir o seu tratamento por outros programas (e.g. Excel);
- a possibilidade de realização de simulações rápidas;
- a utilização de técnicas de análise das propriedades de RP, nomeadamente espaço de estados e invariantes;
- ...

Neste capítulo, serão apresentadas de forma sucinta as várias ferramentas actualmente disponíveis no mercado, para a modelização baseada em RP. Será dado particular ênfase ao Artifex, uma vez que este foi o pacote de *software* utilizado na realização do trabalho de modelização, descrito nos capítulos 8 e 9, e nas simulações apresentadas no capítulo 10.

A Tabelda 6.1 condensa a informação relativa a ferramentas de análise e de simulação de RP, actualmente disponíveis no mercado. Seguem-se algumas notas que visam auxiliar a compreensão daa informação representada na tabela.

Nos tipos de redes suportadas, o significado das designações utilizadas é o seguinte:

- RP básicas – RP não autónomas, cujas marcas não têm qualquer informação associada (não são coloridas) e são representadas por pontos negros;
- RP temporizadas – RP que permitem a introdução do factor tempo, podendo ser utilizadas na análise do desempenho dos sistemas modelizados; os tempos envolvidos são, em princípio, fixos;
- RP estocásticas – RP não autónomas, em que as transições podem ter tempos estocásticos associados, modelizados por variáveis aleatórias, normalmente descritas por funções de densidade de probabilidade exponenciais;
- RP de alto nível – RP coloridas, possibilitando normalmente a definição de tempos de disparo das transições fixos ou estocásticos, e suportando eventualmente a definição de tipos de dados abstractos.

No que diz respeito às funcionalidades:

- a análise de desempenho simples significa a possibilidade de se ter acesso a estatísticas relativas às RP simuladas, tais como, número médio de marcas numa posição, número de marcas que entraram e saíram de uma posição, número de vezes que uma transição foi disparada, etc;
- a análise de desempenho avançada prevê, para além das funcionalidades anteriores, a definição de estatísticas relativas ao sistema particular que se esteja a modelizar, a exportação de dados estatísticos, para permitir o seu tratamento noutros programas, e a representação gráfica dos mesmos.

No que diz respeito ao tipo de licença, o significado das abreviaturas é o seguinte:

- "Grátis" – o *software* é distribuído gratuitamente;
- "Com." – o *software* está disponível apenas comercialmente;
- "Com.(Grát.Ac.)" – o *software* está disponível comercialmente, contudo pode ser distribuído gratuitamente a instituições académicas ou de I&D;
- "Com.(Desc.Ac.)" – o *software* está disponível comercialmente, sendo efectuados descontos na sua distribuição a instituições académicas ou de I&D.

Tabela 6.1 – Ferramentas de suporte à modelização com RP.

Nome do Software	Redes de Petri Suportadas	Principais Características	Ambientes HW/SW	Tipo de Licença
ALPHA/Sim	• RP de alto nível • RP temporizadas	• Editor gráfico • Simulação com animação gráfica dos tokens • Simulação rápida • Análise de desempenho simples	• MS Windows	Com.(Desc.Ac.)
Artifex	• RP de alto nível • RP temporizadas	• Editor gráfico • Simulação com animação gráfica dos tokens • Simulação rápida • Análise de desempenho simples	• Sun • HP • Linux	Com.(Desc.Ac.)
CodeSign	• RP de alto nível • RP temporizadas	• Editor gráfico • Simulação com animação gráfica dos tokens • Simulação rápida	• Sun • MS Windows • Macintosh	
Design/CPN	• RP de alto nível	• Editor gráfico • Simulação com animação gráfica dos tokens • Simulação rápida • Espaços de estados • Análise de desempenho simples	• Sun • Linux • Macintosh	Grátis
ELSIR	• RP de alto nível • RP básicas • RP estocásticas • RP temporizadas	• Editor gráfico • Simulação com animação gráfica dos tokens • Simulação rápida • Análise estrutural • Análise de desempenho simples	• Sun • HP • Linux	Com.(Desc.Ac.)
ExSpect	• RP de alto nível • RP básicas • RP estocásticas • RP temporizadas	• Editor gráfico • Simulação com animação gráfica dos tokens • Espaços de estados • Invariantes de posições • Invariantes de transições • Análise estrutural • Análise de desempenho simples	• Sun • MS Windows	Com.(Desc.Ac.)
F-net	• RP temporizadas	• Editor gráfico • Simulação com animação gráfica dos tokens • Simulação rápida • Espaços de estados • Invariantes de posições • Invariantes de transições • Análise estrutural • Análise de desempenho simples • Análise de desempenho avançada	• MS Windows	Com.(Desc.Ac.)
Looping	• RP de alto nível	• Editor gráfico • Simulação com animação gráfica dos tokens • Simulação rápida	• MS Windows	Com.(Grát.Ac.)

(continuação da Tabela 6.1)

Nome do Software	Redes de Petri Suportadas	Principais Características	Ambientes	Tipo de Licença
Netmate	• RP de alto nível • RP básicas • RP temporizadas	• Editor gráfico • Simulação com animação gráfica dos tokens • Simulação rápida • Invariantes de posições • Invariantes de transições • Análise estrutural	• MS Windows	Com.(Desc.Ac.)
PEP	• RP de alto nível • RP básicas	• Editor gráfico • Simulação com animação gráfica dos tokens • Espaços de estados condensados • Invariantes de posições • Invariantes de transições • Redução de redes • Análise estrutural	• Sun • Linux	Grátis
PETRI Maker	• RP básicas	• Editor gráfico • Simulação com animação gráfica dos tokens • Simulação rápida • Espaços de estados condensados • Invariantes de posições • Invariantes de transições • Análise de desempenho simples	• MS Windows	Grátis
Pntalk	• RP de alto nível • RP básicas	• Editor gráfico • Simulação com animação gráfica dos tokens • Simulação rápida • Análise de desempenho simples	• Sun • MS Windows	Grátis
POSES++	• RP de alto nível • RP básicas • RP estocásticas • RP temporizadas	• Simulação Rápida	• Sun • HP • Linux • MS Windows	Grátis
SYROCO	• RP de alto nível • RP temporizadas	• Editor gráfico • Simulação rápida • Análise de desempenho simples	• C++	Com.(Grát.Ac.)
THORN/DE	• RP de alto nível • RP temporizadas	• Editor gráfico • Simulação rápida • Análise de desempenho simples	• Sun • Linux	Com.(Grát.Ac.)
Visual SimNet	• RP de alto nível • RP básicas • RP estocásticas • RP temporizadas	• Editor gráfico • Simulação com animação gráfica dos tokens • Simulação rápida • Espaços de estados • Invariantes de posições • Invariantes de transições • Análise estrutural • Análise de desempenho simples • Análise de desempenho avançada	• MS DOS • MS Windows	Com.(Grát.Ac.)

6.1 Breve descrição de alguns pacotes de software actualmente disponíveis

6.1.1 ALPHA/Sim

Trata-se de uma ferramenta de uso geral, para a simulação gráfica de eventos discretos, baseada em RP. Permite ao utilizador construir graficamente uma RP, introduzir dados de entrada (atrasos temporais, regras de encaminhamento, marcação inicial, etc), executar a simulação do modelo e visualizar os resultados da simulação.

Os modelos podem ser construídos de forma hierárquica, segundo as abordagens *bottom-up* ou *top-down*. Os modelos ALPHA/Sim podem ser também "linkados" a código externo; este código pode ser escrito em qualquer linguagem que possa ser compilada na plataforma *host*. O ALPHA/Sim colecciona automaticamente estatísticas das populações das posições, atrasos, taxas de disparo das transições e atributos. O ALPHA/Sim tem sido utilizado em diversas aplicações incluindo aplicações militares, de comando, de controlo, sistemas computacionais, sistemas de fabrico e sistemas de filas de espera.

6.1.2 Artifex

O Artifex permite: projectar graficamente, modelizar e analisar sistemas de eventos discretos; construir simuladores e protótipos de sistemas baseados em computador e criar simuladores de tempo real do ambiente que rodeia a aplicação, no sentido de automatizar o seu teste.

O Artifex também permite a prototipagem rápida, incluindo GUIs (*general user interfaces*), e gerar automaticamente toda a aplicação em C para ambientes baseados numa estação de trabalho ou para sistemas embebidos.

O ArtifexTM é uma família de sete produtos altamente integrados, configuráveis conforme as necessidades: Artifex.Modelo, Artifex.Validate, Artifex.GUI, Artifex.Data, Artifex.Measure, Artifex.Report e Artifex.Deploy.

Quando o pacote de *software* é distribuído com desconto a instituições académicas ou de I&D, nem todos estes componentes são fornecidos. Nesses casos, não se tem acesso ao componente Artifex.Deploy.

6.1.3 CodeSign

A ferramenta CodeSign utiliza RP sincronizadas pelo tempo. O modelo de dados, bem como a hierarquia do componente, são estritamente orientados a objectos. Cada classe, incluindo as classes dos componentes, é um tipo de *token* admissível. As transições têm associadas funções de guarda e funções de transição.

O CodeSign foi implementado em Parcplace Visualworks Smalltalk.

6.1.4 Design/CPN

O Design/CPN resulta do trabalho desenvolvido na Universidade de Aarhus, na Dinamarca, pelo Grupo CPN do Departamento de Ciência dos Computadores, no âmbito do desenvolvimentto de métodos de análise e de ferramentas de suporte às RP coloridas (conhecidas pela sigla CPN – *coloured Petri nets*). Trata-se de um pacote de *software* desenvolvido em estrita cooperação entre o Grupo CPN e a empresa norte-americana Meta Software, entre 1989 e 1995.

O Design/CPN é uma das ferramentas de RP mais elaboradas, permitindo tipos de dados abstractos e manipulações complexas de dados, através da sua linguagem de suporte: Standard ML.

O pacote foi inicialmente distribuído comercialmente, mas entretanto a Meta Software forneceu os seus direitos de distribuição à Universidade de Aahrus, devido ao facto da sua estratégia comercial ter mudado, e o Design/CPN ter deixado de ser interessante, do ponto de vista comercial. Desde então, a ferramenta é distribuída gratuitamente a instituições académicas e de I&D.

A ferramenta consiste em três partes integradas:

- CPN Editor – suporta a construção, modificação e verificação sintáctica dos modelos CPN;
- CPN Simulator – suporta a simulação interactiva e automática de modelos CPN;
- Ocurrence Graph Tool – suporta a construção e análise de grafos de ocorrência dos modelos CPN (espaços de estados ou árvores de cobertura).

No início da realização do trabalho descrito nesta Dissertação, tentou-se utilizar o Design/CPN como ferramenta de suporte à modelização com RPC. Para isso, utilizou-se a versão 3.0.4 para Linux. Entretanto, esta foi preterida em favor do Artifex, por duas razões principais: dificuldade no domínio da linguagem de programação associada ao Design/CPN (Standard ML), que se revelou pouco adequada à modelização de um sistema complexo, como é um SAGVs; erros detectados no programa (*bugs*), relacionados com uma má gestão de memória, aquando da realização de algumas simulações com a primeira versão de um modelo relativamente complexo de um SAGVs.

A Figura 6.1 mostra uma RPC construída no editor do Design/CPN. Trata-se de um modelo simples de um protocolo de transporte, para a transmissão de pacotes entre um emissor e um receptor, sobre uma rede de comunicação não fiável. As elipses são as posições e os rectângulos são as transições da RP. Este exemplo é utilizado em [27], para introduzir os conceitos principais subjacentes à teoria das RPC.

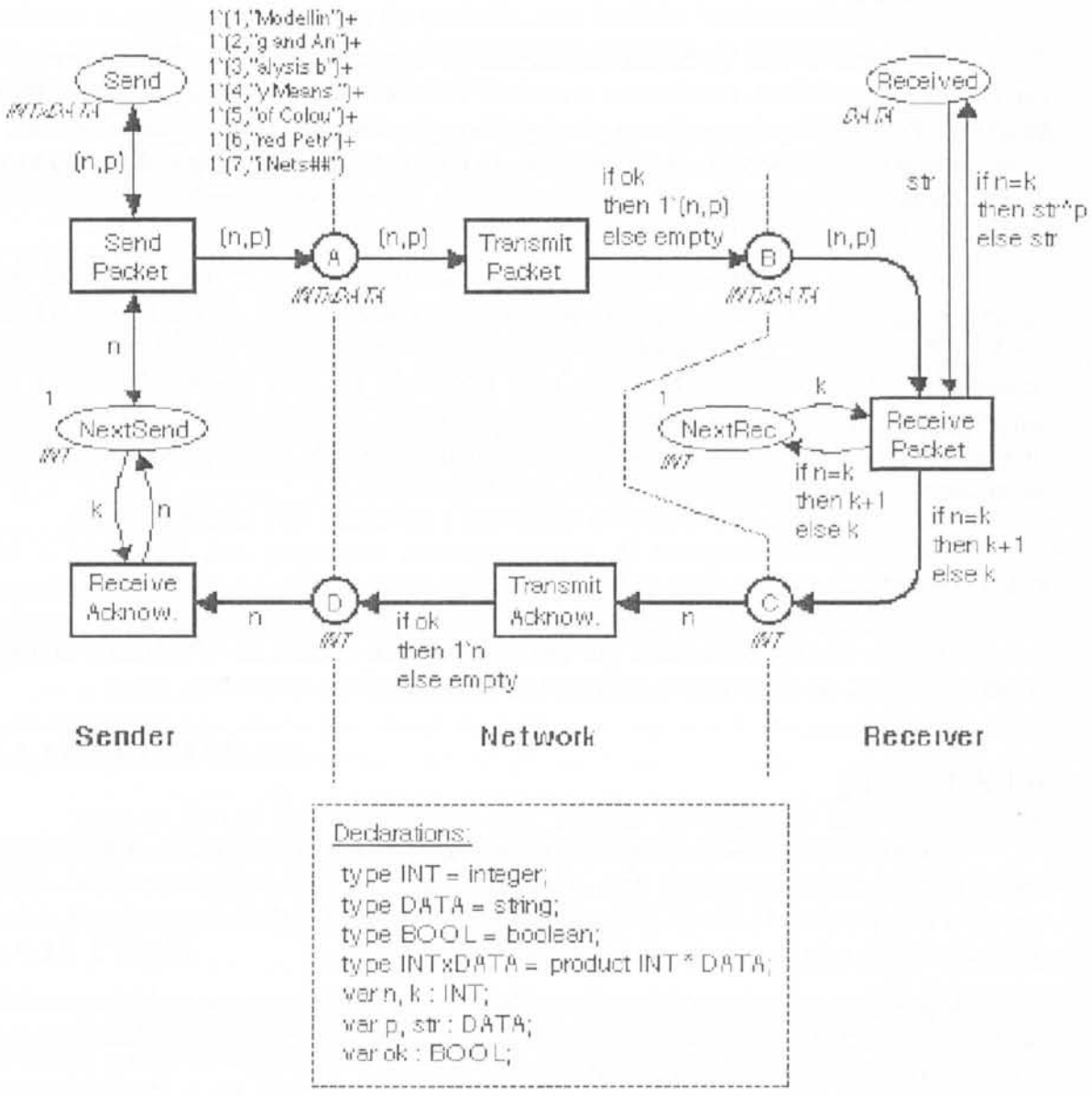


Figura 6.1 – Exemplo de uma RPC construída com a ferramenta Design/CPN.

6.1.5 ELSIR

Trata-se de uma ferramenta de modelização, baseada em RPC, ou seja, RP com controlo de fluxos e encapsulamento e herança do controlo de fluxo. Permite a análise estruturada, a descrição dos dados, a descrição temporal e a instanciação de processos genéricos pré-modelizados.

É usada na análise de desempenho, dimensionamento de sistemas, estudos de fiabilidade, etc. Os seus principais domínios de aplicação são os sistemas embebidos, a análise de redes de computadores e os sistemas de tempo real. Fornece uma ferramenta dedicada à geração automática de relatórios das simulações, com gráficos e texto.

6.1.6 ExSpect

O ExSpect é uma poderosa ferramenta de negócio que confere às organizações a capacidade de modelizar, monitorizar e analisar processos de negócio de maneira efectiva e eficiente. Permite seguir os fluxos de trabalho, de dinheiro e de matérias-primas, para determinar o nível de serviço da organização. O ExSpect oferece o potencial para todo o tipo de simulação, e funciona como auxílio à tomada de decisões em programas de investimento, com vista à redução dos custos e ao aumento das infra-estruturas.

O ExSpect tem uma interface gráfica com o utilizador e uma boa base teórica, desenvolvida em cooperação estreita com a Universidade Técnica de Eindhoven (Holanda), desde 1980. Permite a construção de modelos executáveis com facilidade e rapidez. A reengenharia dos modelos é facilitada pela existência de uma libraria de blocos gerada automaticamente durante a fase do desenvolvimento. Existem librarias especificamente preparadas para determinadas áreas, nomeadamente fluxo de trabalho, logística, processos administrativos e outras áreas de negócio mais específicas.

6.1.7 F-net

Trata-se de uma ferramenta que permite a edição gráfica de RP. Possui também um poderoso sistema de simulação e análise de redes por gráficos de alcance.

6.1.8 Looping

O Looping (*Language for Object Oriented PetrI Nets Grpahically extended*) fornece um ambiente gráfico completo para o projecto, modelização e implementação activa de objectos distribuídos. O ObCS (*Object Control Structure*) é descrito graças às extensões gráficas das RP, que fornecem redes muito concisas.

O Looping permite gerar também código C++ ou Java, para que o objecto modelizado se transforme em processos activos. Assim, os objectos do sistema gerado podem ser postos a correr sobre vários computadores e redes.

6.1.9 Netmate

O Netmate é uma ferramenta de engenharia para o projecto, simulação, verificação e implementação de controladores industriais sequenciais. Os tipos de RP utilizadas no Netmate obedecem aos requisitos do controlo discreto. O uso de hierarquia, arcos adaptativos ou pesagem temporal é independente do tipo de rede escolhido.

Com o Netmate, os algoritmos de controlo são projectados graficamente e podem ser simulados na mesma ferramenta. Adicionalmente, existe a possibilidade de utilizar modelos de processos para a simulação simultânea do processo e do controlador. Para além disso, permite a verificação analítica integrada, baseada num novo método desenvolvido na Universidade de Kaiserlautern (Alemanha). As propriedades da RP podem ser analisadas através de uma interface com a ferramenta INA.

Pode ser gerada e implementada automaticamente uma lista de instruções padrão para aplicações baseadas em PLCs (*programmable logical controllers*). Assim, é possível não só

projectar e verificar o algoritmo de controlo, mas também implementá-lo directamente sem sair da ferramenta. Com a função de monitorização integrada na ferramenta, o desempenho do controlador PLC pode ser verificada *on-line*.

A última versão do Netmate inclui uma interface Windows-DDE para o MATLAB/Simulink, tornando possível a modelização e simulação de sistemas híbridos.

6.1.10 PEP

O *PEP (Programming Environment based on Petri Nets)* é um conjunto completo de componentes para a modelização, compilação e verificação de modelos, integrados numa interface gráfica com o utilizador. Estes componentes facilitam o projecto de sistemas paralelos através de programas paralelos, autómatos que interagem entre si, processamento algébrico ou RP. Os compiladores PEP geram RP a partir desses modelos.

Os simuladores PEP permitem a simulação automática ou disparada pelo utilizador de RP e o disparo dos programas correspondentes. O componente de verificação contém algoritmos básicos para verificar propriedades das RP. O PEP contém uma boa interface com o pacote Starke's INA, no qual é possível realizar vários algoritmos de análise estrutural de RP.

6.1.11 PETRI Maker

Trata-se de um editor gráfico de RP. Permite ao utilizador criar as suas próprias funções em C++. Para além disso, permite controlar o próprio sistema modelizado.

6.1.12 PNtalk

A linguagem PNtalk e o sistema PNtalk são baseados em RP orientadas a objectos (OOPN's – *Object-Oriented Petri Nets*) [31]. Estas consistem em classes que representam o comportamento e os métodos dos objectos. Os *tokens* em OOPN's representam objectos básicos e objectos descritos por RP.

A dinâmica das OOPN's utiliza conceitos inovadores de invocação de procedimentos remotos. As OOPN's podem funcionar com tempos de disparo, que podem resultar de distribuições de probabilidade. O estilo da orientação a objectos do PNtalk é inspirado no Smalltalk. O PNtalk e a sua GUI são implementados em Smalltalk.

6.1.13 POSES++

O POSES++ é um ambiente de simulação de grande velocidade. A simulação é baseada em RP de alto nível com transições temporizadas, arcos temporizados, cardinalidade dos arcos variáveis, algoritmos de predicados de acesso (FIFO, LIFO, RAM, etc), construção de tipos de *tokens* baseados na linguagem C, etc .

Esta ferramenta foi concebida para proporcionar simulações muito rápidas de modelos de grandes dimensões (5500 transições, 7000 predicados e 40000 arcos) e simulações *on-line*. O sistema funciona segundo o modelo cliente-servidor via TCP/IP ou INTERNET. Esta

característica, e a interface com a linguagem C na linguagem de modelização, facilitam a integração de *software*.

6.1.14 SYROCO

O SYROCO é um compilador baseado em objectos cooperativos (COOs – *CoOperative Objects*), que mapeia classes COO em classes C++ de maneira eficiente. [33] É adequado à simulação de sistemas distribuídos e à implementação de aplicações concorrentes.

As COOs são RP de alto nível, que proporcionam instanciação dinâmica, comunicações síncronas e assíncronas, herança, temporização e possibilidades de processamento de informação.

A grande desvantagem do SYROCO é não dispor ainda de uma interface gráfica, que permita a edição gráfica das RP.

6.1.15 THORN/DE

O THORN/DE é uma ferramenta gráfica para a simulação de eventos discretos. Baseia-se numa classe especial de RP, chamadas *Timed Hierarquical Object-Related Nets* (THORNs) [34]. As THORNs permitem a especificação de *tokens* individuais, tempos de atraso e durações para o disparo de transições. Os modelos THORN podem ser estruturados hierarquicamente, no que diz respeito ao refinamento de transições e à invocação de sub-redes. Os tipos de *tokens* e as inscrições das transições são especificados em C++.

O Ambiente de Desenvolvimento THORN é constituído por:

- uma interface gráfica com o utilizador, para editar modelos THORN, marcações iniciais do modelo e opções de simulação, e para visualizar resultados de simulações;
- um compilador de redes para gerar programas executáveis;
- um simulador sequencial, para execuções rápidas de modelos THORN em *background*.

Todos os componentes são implementados em C++ e correm como processos autónomos.

6.1.16 Visual SimNet

O Visual SimNet é um sistema de modelização, simulação e análise de eventos discretos, baseado em RP coloridas e estocásticas, com as seguintes características:

- editor gráfico de modelos;
- simulação animada;
- medida de valores médios e de valores instantâneos;
- análise de distribuição;
- experimentação parametrizada por variáveis;

- selecção da precisão dos resultados;
- gráficos de resultados;
- função osciloscópio;
- análise de gráfico de alcance;
- exportação para os formatos PCX, DXF, HP-GL, HP-GL/2, PCL5, PostScript e LaTeX.

6.2 O software Artifex

O Artifex foi a ferramenta de *software* escolhida para a realização do trabalho de modelização e de simulação baseado em RP, que é descrito nos capítulos 8, 9 e 10. Esta secção faz uma apresentação das principais características e das principais funcionalidades do ambiente Artifex.

6.2.1 Principais características da ferramenta

O Artifex é uma família integrada de vários produtos, para conceber e desenvolver aplicações de *software*, baseadas em eventos discretos e na utilização de RP coloridas, temporizadas e estocásticas [35].

Os modelos Artifex são formais e são automaticamente traduzidos em código que implementa o comportamento definido pela RP. O modelo pode conter fragmentos de código associados, que podem ser utilizados para definir as acções das transições. A linguagem de programação utilizada para definir esses fragmentos de código é o C.

A linguagem Artifex pode ser considerada uma linguagem de programação conceptualmente suficiente para ser utilizada na especificação e simulação de sistemas. A abordagem definida é orientada a objectos, permitindo decompor grandes sistemas em subsistemas, sendo cada subsistema modelizado por um objecto. Por sua vez, cada objecto pode ser definido à custa da articulação entre outros objectos mais simples, e assim sucessivamente.

O Artifex traduz automaticamente as RP em código C, que o simulador utiliza para implementar o comportamento modelizado. A simulação permite validar o modelo e, se necessário, corrigi-lo.

Os fragmentos de código integrados no Artifex permitem o refinamento contínuo do modelo, através da adição progressiva de detalhes, de forma organizada. Este processo pode começar logo na fase de especificação do sistema, permitindo uma rápida prototipagem. De facto, é utilizado o mesmo formalismo nas fases de concepção e de implementação do sistema, de tal forma que o trabalho requerido sobre o modelo de especificação, para se obter o modelo final, é pequeno.

As simulações desenvolvidas nas primeiras fases do ciclo de vida do sistema transformam-se suavemente em actividades paralelas de desenvolvimento e teste de *software*. A validação do *software* é embebida no ciclo de vida do mesmo, enquanto o sistema está a ser concebido e produzido.

6.2.2 Componentes mais importantes do Artifex

O Artifex é uma família integrada de produtos. Cada produto suporta uma actividade específica:

- Artifex.Model – editor gráfico como qual se projecta o modelo do sistema;
- Artifex.Validate – suporta a execução e a validação do modelo;
- Artifex.GUI – melhora os modelos com GUIs Motif;
- Artifex.Data – fornece a modelização gráfica dos dados da simulação e uma linguagem de interrogação melhorada;
- Artifex.Measure – regista e visualiza os dados da simulação;
- Artifex.Report – gera automaticamente um relatório da simulação;
- Artifex.Deploy – transfere o modelo para uma aplicação executável, que pode ser dividida em vários processos.

De seguida, serão apresentadas as principais funcionalidades e características dos produtos utilizados na realização do trabalho descrito nos capítulo 8, 9 e 10.

6.2.2.1 Artifex.Model

O Artifex.Model é um editor gráfico, que permite modelizar os objectos Artifex. Os elementos básicos, que permitem a construção de RP no Artifex estão ilustrados na Figura 6.2. Todos estes elementos são interligados por arcos.



Figura 6.2 – Elementos constituintes de uma RP do Artifex.

Uma posição (*place*) é definida por um nome, um tipo de dados, uma marcação inicial e, opcionalmente, um fragmento de código C, para inicialização dos campos dos *tokens* que façam parte da marcação inicial. Perto do ícone da posição é mostrado o seu nome, separado por ':' do tipo dos seus *tokens*.

O nome é constituído apenas por caracteres maiúsculos, não pode ter espaços e tem de ser único no âmbito do modelo do objecto que esteja a ser construído. O tipo de dados define o tipo dos *tokens* que a posição pode conter. No Artifex o tipo de um *token* é sempre uma estrutura (*struct*), com um número arbitrário de campos. A marcação inicial define o número de *tokens* existentes na posição, imediatamente antes do início da simulação do modelo.

Uma posição partilhada (*shared place*) tem a mesma funcionalidade de uma posição normal, mas permite utilizar a mesma posição em várias páginas do mesmo modelo (em várias sub-redes). Uma posição com o mesmo nome, que apareça em várias páginas de um modelo, corresponde à mesma posição na RP global do objecto que esteja a ser modelizado. A existência de posições partilhadas permite partir modelos mais complexos em sub-redes, estabelecendo a comunicação entre as várias partes.

Uma posição de entrada (*input place*) define uma entrada de um objecto. É através deste tipo de posições que o objecto recebe *tokens* do exterior, com informação do mundo exterior. Para além desta funcionalidade, uma posição de entrada tem todas as funcionalidades de uma posição normal.

Uma posição de saída (output *place*) define uma saída de um objecto. É através deste tipo de posições que o objecto envia *tokens* ao exterior. Para além desta funcionalidade, uma posição de saída tem todas as funcionalidades de uma posição normal, excepto a de fornecer *tokens* que possam ser utilizados por transições internas do objecto. Por outras palavras, todos os *tokens* depositados numa posição de saída só podem ser consumidos por transições existentes fora do objecto.

Uma transição (*transition*) é definida por um nome, uma prioridade, um tipo de temporização, um tempo de disparo, um predicado e uma acção. O nome tem de ser único no âmbito de um mesmo objecto e segue as mesmas regras dos nomes das posições. Perto do ícone de uma transição é indicado o seu nome e, separado por uma vírgula, o valor da sua prioridade, se esta não tiver o valor por defeito (0).

A prioridade serve para estabelecer critérios de desempate em situações de conflito entre transições (valor entre 0 e 65535). Quando num dado instante, há várias transições em condições de ser disparadas, estas são escalonadas pela ordem imposta pelas suas prioridades, começando pela transição de maior prioridade. Quando há mais do que uma transição com a mesma prioridade, a escolha da primeira dessas transições a ser disparada é feita de acordo com uma ordem interna do escalonador do Artifex (e.g. ordem de criação das transições na fase de edição gráfica do modelo).

A temporização das transições é analisada em 6.2.2.1.2.

O predicado é opcional. Quando não é definido, tem por defeito o valor verdadeiro. Quando se especifica um predicado, este consiste numa expressão em C booleana, cuja avaliação tem de ser verdadeira, para que a transição sensibilizada possa ser disparada.

A acção de uma transição é um fragmento de código C, que pode ser de complexidade arbitrária. A sua definição é opcional.

Uma transição com um predicado definido é representada com um losango no centro do rectângulo. Se incluir uma acção, este facto é sinalizado pela presença de uma barra negra na parte inferior do rectângulo.

No predicado e na acção podem intervir os campos dos *tokens* das posições de entrada, intervenientes na avaliação do predicado ou no disparo da transição, bem como variáveis globais e parâmetros do objecto. A acção pode definir o valor dos campos dos *tokens*

depositados nas posições de saída da transição. Quer no predicado, quer na acção, podem ser chamadas funções C definidas no objecto, ou em fragmentos C (designados por *C units*) incluídos pelo objecto, bem como primitivas do próprio Artifex (e.g. alteração da cor ou do estado do objecto, tempo associado ao disparo de uma transição, etc).

A política de saída dos *tokens* de uma posição é do tipo FIFO (*first-in-first-out*). No disparo de uma transição, em cada posição de entrada é escolhido o primeiro *token* que tenha chegado a essa posição e que satisfaça o predicado da transição.

Uma página (*page*) encapsula uma sub-rede do objecto. A este ícone podem ser ligadas posições partilhadas para se aumentar a legibilidade do modelo, ou seja, para se saber quais são as posições partilhadas do objecto intervenientes na definição da sub-rede.

Um objecto (*object*) encapsula uma instância de um objecto de uma classe. A este ícone são ligadas posições que definem a comunicação entre o objecto, que esteja a ser definido, e o objecto encapsulado.

Qualquer uma das entidades apresentadas anteriormente pode conter uma descrição informal, permitindo a fácil documentação do modelo construído.

Todos os objectos podem ter definidos um conjunto de parâmetros globais e um conjunto de variáveis globais. Estas variáveis podem ser utilizadas em todos os fragmentos de código C do objecto.

Cada objecto pode incluir uma secção de código C, denominada por acções iniciais (*initial actions*), que é executada imediatamente antes do início da simulação. Esta é muito útil para abrir ficheiros, ler ficheiros de inicialização do objecto, inicializar variáveis estatísticas do modelo, definir tempos de disparo de transições, etc.

Cada objecto também pode incluir uma secção de código, denominada por acções finais (*final actions*), que é executada imediatamente após o fim de uma simulação. Esta é muito útil para fechar ficheiros (e.g. de log), libertar memória alocada dinamicamente, etc.

Cada objecto pode definir funções C, que podem ser utilizadas nos fragmentos de código C do modelo.

Resumindo, um objecto Artifex pode conter a definição dos seguintes fragmentos de código C:

- inicialização dos *tokens* da marcação inicial de uma posição,
- predicado de uma transição,
- acção de uma transição,
- acções iniciais do objecto,
- acções finais do objecto e
- funções internas.

6.2.2.1.1 Possibilidade de definição de objectos

A modelização de um sistema no Artifex é feita à custa da definição de um objecto que modelize o seu comportamento, podendo ter um número arbitrário de posições de entrada e de posições de saída. Por sua vez, a definição do objecto pode instanciar objectos de outras classes. Assim, é possível decompor o comportamento de um sistema complexo em vários

objectos, modelizando funções mais básicas, cuja articulação define o comportamento global do sistema.

Os tipos dos *tokens* definidos na RP de um objecto não são normalmente conhecidos no seu exterior. Este aspecto suscita um problema: como fazer com que dois objectos a comunicarem entre si possam conhecer simultaneamente o tipo dos *tokens* trocados entre si? Esta dificuldade é resolvida à custa da utilização de *CMN units* (*CoMmuNication units*). Estas são unidades que podem ser incluídas por vários objectos, para conhecerem a definição de tipos de *tokens* utilizados em mais do que um objecto.

Um objecto tem um estado e uma cor associados. Durante a simulação, o estado é visualizado através de um número no interior do ícone do objecto e a cor é identificada através da cor do ícone do objecto. Estas características do objecto podem ser manipuladas e alteradas a partir dos fragmentos de código C do próprio objecto, utilizando as primitivas:

- xx_getstate() – retorna o estado actual do objecto;
- xx_setstate(<estado>) – configura o estado do objecto;
- xx_getcolor() – retorna a cor actual do objecto;
- xx_setcolor(<cor>) – configura a cor do objecto.

6.2.2.1.2 Introdução do factor tempo nos modelos

No Artifex, por defeito uma transição não é temporizada. Contudo, é possível associar um tempo de disparo a uma transição.

O Artifex prevê duas variantes de temporização para uma transição:

- *release delay* – esta variante significa que a transição ao disparar não deposita logo os *tokens* nas suas posições de saída; só o faz ao fim do tempo associado à sua temporização;
- *firing delay* – esta variante significa que quando uma transição está sensibilizada e o seu predicado é avaliado como verdadeiro, a transição só é disparada ao fim do tempo associado à sua temporização, desde que o conjunto de *tokens*, que deu origem à temporização, ainda esteja disponível nas suas posições de entrada.

Uma transição pode conter ambas as formas de temporização em simultâneo, uma vez que elas correspondem a momento diferentes do seu disparo, sendo por isso independentes uma da outra. Quando uma transição é temporizada, esse facto é indicado no desenho da RP, através da letra T, após o nome da transição.

A definição dos tempos pode ser feita na definição dos parâmetros da transição. Os tempos aí definidos são os que vigorarão no início da simulação do sistema.

Para permitir tempos de disparo variáveis, e, em particular, tempos estocásticos, o Artifex tem duas primitivas que permitem configurar os tempos de uma transição em qualquer altura da simulação. A chamada a essas primitivas pode ser feita nas *initial actions* ou na acção de uma transição (ou da própria transição). Os novos tempos configurados só terão efeito no próximo disparo da transição. Associando funções aleatórias à configuração desses tempos, é possível obter-se tempos de disparo estocásticos para as transições da RP.

As primitivas que o Artifex fornece para o efeito são:

- xx_rdelay(<id_transição>, <tempo>) – define o *release delay* de uma transição;

- xx_fdelay(<id_transição>, <tempo>) – define o *firing delay* de uma transição.

6.2.2.2 Artifex.Validate

O Artifex.Validate é o produto Artifex que permite simular os modelos construídos com o Artifex.Model. Antes do início de qualquer simulação, é necessário definir os parâmetros da simulação, nomeadamente o seu nome, e se a simulação vai proceder à acumulação de estatísticas (e de que tipo de estatísticas).

Suporta a animação de *tokens* e a animação de posições e de transições, no sentido de tornar a simulação mais interactiva e perceptível. Na sua forma mais interactiva, é possível visualizar o número de *tokens* em cada posição e o número de disparos das transições. Com a simulação parada, é possível visualizar os dados contidos nos *tokens* existentes numa posição e aceder a estatísticas de posições, transições e objectos. Permite a definição de *breakpoints*, bastante úteis no *debug* e validação dos modelos.

Toda a informação entrada a partir do teclado, ou imprimida com comandos do tipo "printf", é mostrada numa janela auxiliar denominada "*simulator input/output*".

É possível configurar uma simulação mais interactiva, mas necessariamente mais lenta, ou uma simulação não interactiva, mas muito rápida. Em qualquer momento, é possível parar uma simulação. A simulação pode decorrer indefinidamente, passo-a-passo, com paragens periódicas de Δt em Δt, até ser atingido um determinado instante ou quando um *breakpoint* fique activo.

Durante a simulação, é possível visualizar várias janelas simultaneamente, correspondentes a várias sub-redes ou objectos do modelo.

6.2.2.3 Artifex.Measure

Este componente do Artifex permite visualizar estatísticas resultantes da realização de simulações de um modelo. Estas são identificadas por um nome, atribuído na definição dos seus parâmetros no Artifex.Validate.

Há três tipos de estatísticas:

- Índices de cobertura (*coverage indexes*) – dizem qual a percentagem de transições de cada objecto que foram disparadas durante a simulação;
- Métricas da simulação (*simulation metrics*) – dizem respeito a estatísticas sobre as posições, transições e objectos do sistema; são exemplos desse tipo de estatísticas: o estado e a cor de um objecto ao longo do tempo; o número de disparos, o número de avaliações de predicados e a taxa de disparo de uma transição; o comprimento médio da fila, o tempo médio de espera de um *token*, a taxa de chegada dos *tokens* e a taxa de sáida dos *tokens* de uma posição;
- Estatísticas definidas pelo utilizador (*user-defined measures*) – dizem respeito a estatísticas específicas do modelo em causa; a sua gestão é da responsabilidade do utilizador, nomeadamente a inicialização da variável estatística e a acumulação de medidas ao longo do tempo; o Artifex.Measure permite calcular o mínimo, o máximo, o valor médio e o desvio padrão do conjunto das amostras de uma variável estatística, guardadas pelo utilizador ao longo da simulação.

Em todos estes tipos de estatísticas, é possível realizar a sua representação gráfica ou exportar os dados, para permitir o seu tratamento noutros programas de análise estatística (e.g. Excel).

As primitivas Artifex a utilizar no Artifex.Model, para criar e gerir estatísticas definidas pelo utilizador são:

- xx_initmeasure(<nome_estatística>) – permite obter um identificador de uma nova variável estatística definida pelo utilizador, a partir de uma *string* (do seu nome);
- xx_writesample(<id_estatística>, <valor_real>) – permite adicionar uma amostra referente a uma estatística definida pelo utilizador; <id_estatística> é o valor retornado por 'xx_initmeasure()'.

BIBLIOTECA F.E. PORTO

7. *Especificação de um modelo de um SAGVs*

Tal como acontece noutros problemas de engenharia, na concepção de SAGVs há um compromisso flexibilidade versus desempenho, ao nível do seu controlo. Um SAGVs tem a responsabilidade de tomar decisões deveras complexas, tais como afectar a uma dada tarefa o AGV que esteja melhor colocado para a sua execução, escolher o melhor trajecto para um AGV viajar entre dois pontos, gerir o tráfego existente na rede de AGVs, etc. Se o SAGVs estiver preparado para responder a uma grande multiplicidade de situações, terá um pior desempenho do que uma solução alternativa mais limitada na quantidade de situações às quais consegue responder, porque a generalização é normalmente inimiga da optimização.

Por exemplo, um SAGVs bidireccional é naturalmente mais flexível nos movimentos possíveis para os AGVs do que um SAGVs unidireccional, porque aumenta muito a mobilidade dos AGVs no seio do *layout* do sistema de fabrico, para o mesmo *layout* de trajectórias dos AGVs. Todavia, o SAGVs bidireccional tem uma gestão de tráfego mais difícil e nalguns *layouts* aumenta significativamente a probabilidade de ocorrência de *deadlocks*. Se a resolução desses problemas não for muito optimizada, ou se a flexibilidade oferecida por um SAGVs unidireccional for suficiente para o sistema de fabrico em causa, o desempenho global do SAGVs bidireccional será mais pobre.

O exemplo anterior confirma a regra de que um aumento de flexibilidade do sistema tem o senão de penalizar o seu desempenho. Para além disso, permite constatar que a solução mais sofisticada nem sempre é a melhor solução para as exigências colocadas a um SAGVs.

Quase sempre a avaliação sistemática ou por meios analíticos das várias soluções alternativas para uma determinada aplicação revela-se impossível, dada a complexidade dos sistemas de fabrico e dos SAGVs, em particular. Para além disso, muitas vezes é difícil ter a noção das verdadeiras exigências do caso em estudo. Quando muito, será possível tecer considerações qualitativas baseadas no senso comum, com base no conhecimento da dinâmica do sistema de fabrico em causa. Como essa avaliação é apenas indicativa, há quase sempre a necessidade de experimentar a solução, para se fazer a sua avaliação de forma segura e mais concreta. Assim, é muito comum no estudo de sistemas de fabrico utilizar a simulação em computador de um modelo do sistema em causa, como meio de experimentação e de avaliação de soluções alternativas, na fase da concepção do sistema. O SAGVs, sendo parte integrante de um sistema de fabrico, não foge à regra.

Neste capítulo serão apresentadas as especificações de um modelo de um SAGVs, que serviu de base à simulação de um SAGVs, utilizando RPC. A construção desse modelo de simulação no Artifex será apresentada nos capítulos 8 e 9. A arquitectura adoptada para o SAGVs é a arquitectura de um SAGVs em três camadas, que foi definida em 2.2.

7.1 Principais entidades de um SAGVs

As principais entidades de um SAGVs são:

- o AGV,

- a Tarefa e
- o *Layout*.

7.1.1 Estados possíveis para um AGV

O estado de um AGV tem três níveis:

- Estado de movimentação – define as condições actuais de movimentação do AGV;
- Estado de alocação – define as condições actuais de controlo do AGV, por parte do controlador do SAGVs (SGAGV);
- Estado de carga – define se o AGV carrega ou não alguma carga num dado momento.

Os estados de movimentação possíveis para um AGV estão indicados na Tabela 7.1.

Tabela 7.1 – Estados de movimentação possíveis de um AGV.

Estado de movimentação	Significado
DISABLED	AGV desactivado.
ENABLED	AGV activado, mas não é conhecida a sua posição actual, nem poderá receber ordens automáticas de movimentação.
PARADO	AGV parado e disponível para receber e executar uma nova ordem de movimento.
EX_MOVIM	AGV a executar um movimento de translacção.
CARGA_AGV	AGV a executar uma operação de carga de uma peça ou palete.
DESCARGA_AGV	AGV a executar uma operação de descarga de uma peça ou palete.
BLOQUEADO	AGV impedido de realizar um movimento de translacção, devido a conflitos com outros AGVs ou problemas na gestão de tráfego.
ERRO	AGV com um erro interno.

Os estados de alocação possíveis para um AGV estão indicados na Tabela 7.2. Esta informação de estado só é relevante, quando o AGV não se encontre nos estados de movimentação DISABLED ou ENABLED, pois só nesse caso faz sentido falar-se no controlo do AGV, por parte do SGAGV.

Tabela 7.2 – Estados de alocação de um AGV.

Estado de alocação	Significado
DESALOCADO	AGV pode ser despachado livremente pelo SAGVs.
ALOCADO	AGV só pode ser afectado a tarefas de alto nível (tarefas cuja afectação aos AGVs é feita ao nível do SGM).

Os estados possíveis para o estado de carga de um AGV estão indicados na Tabela 7.3. Esta informação de estado só é relevante, quando o AGV não se encontre nos estados de movimentação DISABLED ou ENABLED.

Tabela 7.3 – Estados de carga de um AGV.

Estado de carga	Significado
DESCARREGADO	AGV descarregado (se qualquer peça ou palete).
CARREGADO	AGV carregado com uma peça ou palete.

7.1.2 Estados possíveis para uma tarefa e tipos de tarefas possíveis

Os estados possíveis para uma tarefa são os que estão indicados na Tabela 7.4.

Tabela 7.4 – Estados possíveis de uma tarefa.

Estado da tarefa	Significado
FILA	Tarefa em fila de espera.
PASSO_1	Estado em que o AGV afectado à tarefa se desloca entre a sua posição inicial e o ponto de carga.
PASSO_2	Estado em que o AGV afectado à tarefa carrega a peça ou a palete no ponto de carga.
PASSO_3	Estado em que o AGV afectado à tarefa se desloca entre o ponto de carga e o ponto de descarga.
PASSO_4	Estado em que o AGV afectado à tarefa descarrega a peça ou a palete no ponto de descarga.
CONCLUIDA	Tarefa concluída.
REJEITADA	Tarefa rejeitada, devido a incoerências nos seus parâmetros.

Os tipos de Tarefas previstos são os que estão indicados na Tabela 7.5. A definição de todos estes tipos de tarefas teve como objectivo tornar o modelo o mais próximo possível da realidade. Todavia, os tipos de tarefas mais essenciais são os tipos NORMAL e POSIC_ESPEC.

Tabela 7.5 – Tipos de tarefas existentes.

Tipo de tarefa	Descrição
NORMAL	Tarefa completa (pode ter todos os estados indicados na Tabela 7.4); o seu escalonamento é da competência do SGAGV.
NORMAL_AGV	Tarefa idêntica à do tipo NORMAL, mas escalonada pelo SGM.
POSICION	Tarefa que não pode ter os estados PASSO_2, PASSO_3 e PASSO_4; é escalonada pelo SGM e é útil para movimentar um AGV entre dois pontos.
T_CARGA	Tarefa que não pode ter os estados PASSO_3 e PASSO_4; é uma uarefa só de carga e é escalonada pelo SGAGV.
CARGA_AGV	Tarefa idêntica à do tipo T_CARGA, mas escalonada pelo SGM.
T_DESCARGA	Tarefa que não pode ter os estados PASSO_1 e PASSO_2; é uma tarefa só de descarga e é escalonada pelo SGM.
POSIC_ESPEC	Tarefa idêntica à do tipo POSICION, mas lançada e escalonada pelo SGAGV; é útil para resolver conflitos decorrentes da gestão de tráfego.

7.1.3 O layout

A representação da informação do *layout* dos AGVs segue a metodologia apresentada em 3.1.1. Esta representação guarda a estrutura básica do *layout*.

Em cada bloco do SAGVs, esta informação tem de ser complementada com informações de carácter mais específico de cada camada, tais como: a probabilidade de lançamento de uma tarefa do tipo POSICION, por parte do SGM, para um determinado nó do *layout*; ou a posição relativa de uma mesa de carga/descarga em relação ao respectivo nó do *layout* (informação relevante para o SGAGV); ou a probabilidade de introdução de um AGV num determinado nó do *layout* (informação relevante para o SPAGV).

7.2 Especificação dos blocos constituintes do SAGVs

A arquitectura adoptada para o SAGVs é a arquitectura de um SAGVs em três camadas, conforme foi definida em 2.2. Os blocos que serão objecto de análise neste capítulo e no próximo capítulo, são aqueles que dizem respeito directamente ao SAGVs, ou seja: o SGM, o SGAGV e o SPAGV (ver Figura 2.3). O bloco 'Fábrica' será abordado no capítulo 9.

Nesta secção, serão especificadas as mensagens trocadas entre os blocos, as funcionalidades de cada uma das camadas, as estratégias adoptadas e a informação existente em cada camada, sobre cada entidade do SAGVs.

7.2.1 Mensagens trocadas entre os blocos do SAGVs

De seguida, serão especificadas as mensagens que podem ser trocadas entre os blocos do SAGVs, de acordo com os fluxos de informação ilustrados pela Figura 2.3 da página 11.

7.2.1.1 Mensagens trocadas entre o SGM e o SGAGV

A Tabela 7.6 indica todos os tipos de mensagens que o SGM pode enviar ao SGAGV. Todos os parâmetros indicados com '–' não são relevantes para a mensagem em causa.

Tabela 7.6 – Mensagens que o SGM pode enviar ao SGAGV.

Tipo de mensagem	Parâmetro 1	Parâmetro 2	Parâmetro 3	Parâmetro 4
ENABLE_AGV	<id_agv>	ALOCA_AGV	–	–
ENABLE_AGV	<id_agv>	DESALOCA_AGV	–	–
DISABLE_AGV	<id_agv>	–	–	–
TAREFA_NORMAL	QUALQUER	<origem>	<destino>	<prior>
TAREFA_NORMAL	<id_agv>	<origem>	<destino>	–
TAREFA_POSICION	<id_agv>	<no>	–	–
TAREFA_CARGA	QUALQUER	<no>	–	<prior>
TAREFA_CARGA	<id_agv>	<no>	–	–
TAREFA_DESCARGA	<id_agv>	<no>	–	–
CANCELA_TAREFA	UMA_TAREFA	<id_tarefa>	–	–
CANCELA_TAREFA	TODAS_TAR_EXEC	–	–	–
CANCELA_TAREFA	TODAS_TAR	–	–	–

As mensagens do tipo ENABLE_AGV permitem activar o AGV identificado pelo parâmetro 1. Consoante o valor do parâmetro 2, o AGV fica no estado de alocação ALOCADO ou NAO_ALOCADO.

A mensagem do tipo DISABLE_AGV permite desactivar o AGV identificado pelo parâmetro 1.

As mensagens do tipo TAREFA_NORMAL permitem lançar uma tarefa do tipo NORMAL (se o parâmetro 1 valer QUALQUER) ou do tipo NORMAL_AGV (se o parâmetro 1 identificar um determinado AGV). O parâmetro p2 indica o ponto de carga e o parâmetro p3 indica o ponto de descarga. No caso da tarefa do tipo NORMAL, que é escalonada pelo SGAGV, o parâmetro p4 indica a prioridade da tarefa.

As mensagens do tipo TAREFA_CARGA permitem lançar uma tarefa do tipo T_CARGA (se o parâmetro 1 valer QUALQUER) ou do tipo CARGA_AGV (se o parâmetro 1 identificar um determinado AGV). O parâmetro p2 indica o ponto de carga. No caso da tarefa do tipo T_CARGA, que é escalonada pelo SGAGV, o parâmetro p4 indica a prioridade da tarefa.

A mensagem do tipo TAREFA_DESCARGA permite lançar uma tarefa de descarga, a afectar ao AGV identificado pelo parâmetro 1, no ponto de descarga indicado pelo parâmetro 2.

As mensagens do tipo CANCELA_TAREFA permitem ordenar o cancelamento de tarefas pendentes ou em execução. O parâmetro 1 identifica qual ou quais as tarefas a executar: se valer UMA_TAREFA, cancela a tarefa identificada pelo parâmetro 2; se valer TODAS_TAR_EXEC, cancela todas as tarefas que estejam a ser executadas; se valer TODAS_TAR, cancela todas as tarefas pendentes ou em execução, sem excepção.

A Tabela 7.7 indica todos os tipos de mensagens que o SGAGV pode enviar ao SGM. Todos os parâmetros indicados com '–' não são relevantes para a mensagem em causa.

Tabela 7.7 – Mensagens que o SGAGV pode enviar ao SGM.

Tipo de mensagem	Parâmetro 1	Parâmetro 2	Parâmetro 3	Parâmetro 4
CONFIRMACAO	SUCESSO	–	–	–
CONFIRMACAO	INSUCESSO	<erro>	–	–
TAREFA_CANCEL	<estado_tar>	<id_agv>	<razao>	–
ESTADO_AGV	<id_agv>	<estado_movim>	<estado_aloc>	<estado_carga>
ESTADO_TAREFA	<id_tarefa>	<estado_tar>	<id_agv>	–
T_POSIC_ESPEC	<id_tarefa>	<id_agv>	<no>	–

As mensagens do tipo CONFIRMACAO servem para o SGAGV notificar o SGM sobre o sucesso ou o insucesso da execução de uma tarefa ou de uma operação de cancelamento de tarefas, consoante o valor do parâmetro 1. No caso de insucesso, o parâmetro 2 indica o erro ocorrido, de acordo com os casos definidos pela Tabela 7.8.

Tabela 7.8 – Erros pelos quais o SGM pode ser notificado pelo SGAGV, através da mensagem (CONFIRMACAO, INSUCESSO, <erro>).

Erro notificado pelo SGAGV ao SGM	Significado
AGV_INEX	AGV inexistente.
AGV_INDISPON	AGV indisponível para a realização de uma tarefa escalonada pelo SGM.
AGV_CARR	Impossibilidade de realização de uma tarefa, pelo facto do AGV se encontrar carregado.
AGV_DESCARR	Impossibilidade de realização de uma tarefa, pelo facto do AGV se encontrar descarregado.
TAREFA_INEX	Ordem de cancelamento de uma tarefa que não existe.
ORIGEM_INVAL	Origem inválida de uma tarefa do tipo NORMAL ou NORMAL_AGV.
DESTINO_INVAL	Destino inválido de uma tarefa do tipo NORMAL ou NORMAL_AGV.
NO_INVALIDO	Nó inválido de uma tarefa de carga, de descarga ou de posicionamento.
CAMINHO1_INEX	Caminho correspondente ao PASSO_1 da tarefa inexistente (devido a um erro de concepção do layout).
CAMINHO2_INEX	Caminho correspondente ao PASSO_3 da tarefa inexistente (devido a um erro de concepção do layout).
OP_CAR_IMP	Operação de carga impossível, devido a impossibilidades de ordem física em AGVs que só possam efectuar operações de carga/descarga para um dos lados.
OP_DES_IMP	Operação de descarga impossível, devido a impossibilidades de ordem física em AGVs que só possam efectuar operações de carga/descarga para um dos lados.

A mensagem do tipo TAREFA_CANCEL serve para notificar o SGM sobre o resultado do cancelamento de uma tarefa. O parâmetro 1 identifica o estado da tarefa, imediatamente antes do cancelamento da tarefa. O parâmetro 2 identifica o AGV a que estava afectada, no caso de se encontrar em execução (no caso contrário este parâmetro é irrelevante). O parâmetro 3 identifica a razão do cancelamento da tarefa, de acordo com os casos indicados na Tabela 7.9.

Tabela 7.9 – Razões possíveis para o cancelamento de uma tarefa por parte do SGAGV.

Parâmetro 3 de uma mensagem TAREFA_CANCEL	Significado
ORDEM_SGM	Por ordem do SGM.
AGV_DESACTIVADO	Devido à desactivação do AGV que estava afectado à tarefa (passagem ao estado DISABLED ou ENABLED).
DEADLOCK	Devido à ocorrência de um *deadlock*.

A mensagem do tipo ESTADO_AGV notifica o SGM sobre a alteração do estado do AGV identificado pelo parâmetro 1. O parâmetro 2 indica o estado de movimentação, o parâmetro 3 indica o estado de alocação (quando relevante) e o parâmetro 4 indica o estado de carga (quando relevante).

A mensagem do tipo ESTADO_TAREFA notifica o SGM sobre a alteração do estado da tarefa identificada pelo parâmetro 1. O parâmetro 2 indica o estado actual da tarefa e o parâmetro 3 indica o AGV a que se encontra afectado (se estiver afectada a algum AGV).

A mensagem do tipo T_POSIC_ESPEC informa o SGM sobre o lançamento, por parte do SGAGV, de uma tarefa de posicionamento "especial", com o intuito de resolver um conflito relacionado com a gestão do tráfego de AGVs.

7.2.1.2 Mensagens trocadas entre o SGAGV e o SPAGV

A Tabela 7.10 indica todos os tipos de mensagens que o SGAGV pode enviar ao SPAGV. Para além dos parâmetros indicados, todas as mensagens aí indicadas, sem excepção, têm um parâmetro indicativo do AGV a que se destinam. Todos os parâmetros indicados com '–' não são relevantes para a mensagem em causa.

Tabela 7.10 – Mensagens que o SGAGV pode enviar ao SPAGV.

Tipo de mensagem	Parâmetro 1	Parâmetro 2
MOVIMENTO	<tipo_mov>	<param>
INF_EST_AGV	<estado_movim>	–
ESTADO_INTR	<result_intr>	–
CANCEL_MOV_AVAN	–	–

A mensagem do tipo MOVIMENTO ordena a um AGV a execução de um movimento. Os parâmetros 1 e 2 definem o tipo de movimento a efectuar, de acordo com os casos indicados na Tabela 7.11.

Tabela 7.11 – Valores dos parâmetros 1 e 2 de uma mensagem do tipo MOVIMENTO.

Parâmetro 1	Parâmetro 2	Significado
MOV_TRANSL	<no>	Movimento de translacção até ao nó <no>.
MOV_CARGA	<lado>	Movimento de carga no nó actual para o lado <lado> (ESQUERDA ou DIREITA).
MOV_DESCARGA	<lado>	Movimento de descarga no nó actual para o lado <lado> (ESQUERDA ou DIREITA).

A mensagem do tipo INF_EST_AGV informa o SPAGV sobre a alteração do estado de um AGV. O parâmetro 1 indica qual o novo estado de movimentação de um AGV.

A mensagem do tipo ESTADO_INTR notifica o SPAGV sobre o resultado da tentativa de introdução de um AGV (ver significado de "introdução de um AGV" em 2.2.1). O parâmetro 1 indica esse resultado, de acordo com os casos indicados pela Tabela 7.12.

Tabela 7.12 – Valores possíveis para o parâmetro 1 de uma mensagem do tipo ESTADO_INTR.

Valor parâmetro 1	Significado
AGV_INTROD	AGV introduzido com sucesso.
AGV_NOT_ENABLED	AGV não introduzido, porque o AGV está no estado DISABLED.
NO_INEX	AGV não introduzido, porque o nó indicado para a introdução não existe.
NO_OCUPADO	AGV não introduzido, porque o nó indicado para a introdução não está disponível.

A mensagem do tipo CANCEL_MOV_AVAN serve para o SGAGV ordenar o cancelamento da execução de movimentos (de translacção) enviados em avanço para um AGV. Isto é necessário quando a tarefa afectada a um AGV é cancelada ou reafectada a outro AGV.

A Tabela 7.13 indica todos os tipos de mensagens que o SPAGV pode enviar ao SGAGV. Para além dos parâmetros indicados, todas as mensagens aí indicadas, sem excepção, têm um parâmetro indicativo do AGV a que se referem. Todos os parâmetros indicados com '–' não são relevantes para a mensagem em causa.

Tabela 7.13 – Tipos de mensagens que o SPAGV pode enviar ao SGAGV.

Tipo de mensagem	Parâmetro 1	Parâmetro 2
FIM_MOVIMENTO	–	–
INTRODUCAO	<no>	<orient>
ERRO_AGV	–	–
FIM_ERRO_AGV	–	–
AGV_MANUAL	–	–
MOV_AVAN_CANCEL	–	–

A mensagem do tipo FIM_MOVIMENTO notifica a realização com sucesso do primeiro movimento (primeiro, pela ordem de recepção) existente na fila de movimentos pendentes do AGV em causa.

A mensagem do tipo INTRODUCAO solicita ao SGAGV a introdução do AGV em causa no nó indicado pelo parâmetro 1, e segundo a orientação indicada pelo parâmetro 2.

A mensagem do tipo ERRO_AGV notifica o SGAGV da ocorrência de um erro interno no AGV em causa. A mensagem do tipo FIM_ERRO notifica o SGAGV da resolução do erro.

A mensagem do tipo AGV_MANUAL notifica o SGAGV da passagem do AGV em causa ao modo automático.

A mensagem do tipo MOV_AVAN_CANCEL notifica o SGAGV do cancelamento dos movimentos recebidos em avanço para o AGV em causa, na sequência da recepção, por parte do SPAGV, de uma mensagem do tipo CANCEL_MOV_AVAN.

7.2.2 Especificação do SGM

De acordo com a arquitectura definida para o SGM em 2.2.3, as principais tarefas que o SGM suporta são:

- Lançamento de ordens de movimentação recebidas da fábrica;
- Lançamento de ordens escalonadas pelo próprio SGM, no sentido de simular a existência de tarefas especiais, cujos requisitos de tempo-real não possam ser satisfeitos pelo escalonamento realizado ao nível do SGAGV;
- Acompanhamento do estado das tarefas pendentes;
- Configuração do número de veículos do sistema, através da activação e desactivação de AGVs;
- Acompanhamento do estado dos AGVs;
- Cancelamento de tarefas.

Para levar a cabo a execução destas tarefas, o SGM tem de suportar as mensagens especificadas em 7.2.1.1.

7.2.2.1 Informação sobre um AGV a guardar no SGM

A informação sobre cada AGV, a que o SGM tem de ter acesso para cumprir as suas funções são:

- Identificador do AGV;
- Estado de movimentação;
- Estado de alocação;
- Estado de carga;
- Destino do último movimento efectuado;
- Outros dados necessários ao controlo da activação e desactivação do AGV.

7.2.2.2 Informação sobre uma tarefa a guardar no SGM

A informação sobre cada tarefa, a que o SGM tem de ter acesso para cumprir as suas funções são:

- Identificador da tarefa;
- Tipo de tarefa;
- Prioridade;
- Nós identificativos do movimento (ponto de carga e/ou ponto de descarga, ou simplesmente o nó para onde deve ser movimentado o AGV numa tarefa de posicionamento);
- Estado da tarefa.

7.2.2.3 Informação sobre o layout a guardar no SGM

O SGM não necessita de ter acesso a informação relativa à estrutura do *layout* de AGVs, para ser capaz de realizar as suas funções. No entanto, para poder lançar tarefas escalonadas pelo próprio SGM, tem de ter acesso a informação probabilística sobre os nós a escolher para esse tipo de tarefas lançadas.

Assim deverá ter acesso aos seguintes parâmetros do *layout*:

- probabilidade de um nó ser escolhido como ponto de carga de uma tarefa;
- probabilidade de um nó ser escolhido como ponto de descarga de uma tarefa;
- probabilidade de um nó ser escolhido como destino de uma tarefa de posicionamento.

Estas informações são configuráveise através de um ficheiro, para que o SGM possa funcionar com qualquer *layout* de AGVs.

7.2.3 Especificação do SGAGV

O SGAGV tem de cumprir as funções definidas na arquitectura apresentada em 2.2.2.

As estratégias adoptadas para a realização das várias funções do SGAGV são:

- Afectação de AGVs às tarefas (escalonamento) – para este efeito, o SGAGV suporta a utilização do algoritmo húngaro (ver 3.5.5.1), com ou sem reafectação dinâmica de tarefas (ver 3.5.5.2), e a utilização da maior parte das regras de despacho definidas em 3.5.4, nomeadamente VMP, VMU, TMCV, MTFS, MELFS e PCPSM (consultar a lista de abreviaturas para entender o significado destas siglas); a configuração do escalonador a utilizar é um dos parâmetros de funcionamento do SGAGV, permitindo comparar o desempenho das várias implementações suportadas;
- Encaminhamento dos AGVs – é utilizado o algoritmo do caminho mais curto apresentado em 3.4.2.1;

- Despacho e gestão do tráfego dos AGVs – é utilizada uma política de reserva e libertação de nós físicos e de nós fictícios do *layout*, de acordo com o raciocínio apresentado em 3.1.1; o SGAGV prevê o envio aos AGVs de movimentos de translacção em avanço, para evitar paragens desnecessárias dos veículos, de acordo com a discussão apresentada em 3.3.2;
- Resolução de conflitos na gestão de tráfego – a abordagem seguida na resolução do problema de *deadlocks* é a de detecção e resolução de *deadlocks* (ver 3.3.4.2); os *deadlocks* são detectados através da detecção de listas circulares do tipo (AGV bloqueado, AGV bloqueante), e são resolvidos por reencaminhamento de um dos AGVs intervenientes; o SGAGV também evita o bloqueio de AGVs, com movimentos pendentes, por AGVs parados e disponíveis, ou AGVs no estado ERRO; no primeiro caso o AGV PARADO é posicionado num nó onde não estorve a passagem dos outros AGVs em serviço; no segundo caso, os AGVs bloqueados são reencaminhados, de forma a contornar o AGV no estado ERRO.

No cumprimento das suas funções, o SGAGV tem de suportar as mensagens especificadas em 7.2.1.1 e 7.2.1.2, e de receber informação da fábrica sobre o estado das filas de espera dos pontos de carga, para poder implementar as regras de despacho MTFS, MELFS e PCPSM (em caso de dúvida, ver a lista de abreviaturas).

7.2.3.1 Informação sobre um AGV a guardar no SGAGV

A informação sobre cada AGV, a que o SGAGV tem de ter acesso para cumprir as suas funções são:

- Identificador do AGV;
- Posição actual ou para onde se esteja a deslocar (nó do *layout*);
- Última posição ocupada, antes da execução do último movimento de translacção, ou antes do início do movimento de translacção que esteja a executar, se o AGV estiver em movimento entre dois nós;
- Estado de movimentação;
- Estado de alocação;
- Estado de carga;
- Tarefa afectada (se estiver afectado a uma tarefa);
- Número de movimentos elementares pendentes;
- *Flag* indicativa da sua reafectação.

Para além das informações anteriores, que dizem respeito ao controlo do AGV, o SGAGV tem de ter acesso a informações estatísticas do comportamento do veículo, para ser possível a realização da análise do desempenho do AGV.

Para o efeito, o SAGV tem de ter acesso às seguintes estatísticas de um AGV:

- Instante em que mudou para o estado actual;
- Tempos acumulados de permanência do AGV em cada um dos estados de movimentação possíveis (ver Tabela 7.1 na página 104);

- Contagens de movimentos – número de arranques descarregado, número de trajectos descarregado, número de cargas, número de arranques carregado, número de trajectos carregado e número de descargas.

7.2.3.2 Informação sobre uma tarefa a guardar no SGAGV

A informação sobre cada tarefa, a que o SGAGV tem de ter acesso para cumprir as suas funções são:

- Identificador da tarefa;
- Tipo de tarefa;
- Prioridade;
- Nós identificativos do movimento (ponto de carga e/ou ponto de descarga, ou simplesmente o nó para onde deve ser movimentado o AGV numa tarefa de posicionamento);
- Estado da tarefa;
- Estado da tarefa, antes da chamada do processo escalonador;
- Trajectos calculados para a realização do PASSO_1 e do PASSO_3, bem como os custos associados;

Para além das informações anteriores, que dizem respeito à gestão das tarefas, o SGAGV tem de ter acesso a informações estatísticas da tarefa, para ser possível a realização da análise do desempenho do SAGVs.

Para o efeito, o SAGV tem de ter acesso às seguintes estatísticas de uma tarefa:

- Instante em que a tarefa foi recebida pelo SGAGV (instante de lançamento);
- Instante em que mudou para o estado actual;
- Tempos de permanência da tarefa nos estados FILA, PASSO_1, PASSO_2, PASSO_3 e PASSO_4;
- Número de afectações sofridas ao longo da sua execução.

7.2.3.3 Informação sobre o layout a guardar no SGAGV

O SGAGV necessita de ter acesso a informação relativa à estrutura do *layout* de AGVs, para ser capaz de realizar as suas funções de escalonamento, encaminhamento e de gestão de tráfego. A estrutura utilizada para representar o *layout* é a que está descrita em 3.1.1.

Todavia, para poder determinar se uma determinada operação de carga ou de descarga pode ser realizada, no caso em que os AGVs só podem efectuar esse tipo de operações para um dos lados (para a esquerda ou para a direita), o SGAGV necessita conhecer a posição relativa das mesas de transferência, em relação aos respectivos nós do *layout*. Esta informação é necessária para todos os nós do *layout* que não sejam do tipo TRANSITO (ver Tabela 3.4 na página 25). Para que o SGAGV possa funcionar com qualquer *layout*, a posição relativa das mesas de carga/descarga, em relação aos respectivos nós, é configurada através de um ficheiro.

A Figura 7.1 ilustra como é determinada a posição relativa de uma mesa de carga/descarga, em relação ao nó do *layout*. Tomando como referência o sentido positivo de circulação dos AGVs (arbitrado) e o referencial colocado no nó do *layout*, orientado no sentido de circulação positivo, a orientação dos eixos indicam para que lado a posição relativa vale +1 e para que lado vale -1. Observe-se que tomando como referência um AGV orientado segundo o sentido de circulação arbitrado como positivo, uma mesa colocada à sua esquerda tem uma posição relativa igual a +1 e uma mesa colocada à sua direita tem uma posição relativa igual a -1.



Figura 7.1 – Definição da posição relativa de uma mesa de carga/descarga em relação ao respectivo nó do layout e determinação do lado para o qual o AGV trabalha.

O SGAGV determina o lado para o qual o AGV tem de efectuar a operação de carga/descarga, multiplicando a posição relativa da mesa pelo sentido do AGV, em relação ao sentido de circulação positivo. Por exemplo, se o AGV tiver um sentido igual a -1 (caso b) da figura) e a posição relativa da mesa for -1, o AGV efectua o movimento para a sua esquerda porque (-1)*(-1)=(+1).

Para cada nó onde possam ocorrer operações de carga e/ou operações de descarga, o ficheiro de configuração contém uma linha com a sintaxe definida pela Tabela 7.14.

Tabela 7.14 – Sintaxe de uma linha do ficheiro de configuração da posição dos pontos de carga/descarga em relação aos respectivos nós do layout.

Sintaxe de uma linha	Significado
M:<id_no>,<pos_rel>	O ponto de carga/descarga <id_no> tem uma posição <pos_rel> em relação ao nó <id_no> do layout; o parâmetro <pos_rel> pode valer +1 ou -1, de acordo com o raciocínio explicitado pela Figura 7.1.

7.2.4 Especificação do SPAGV

O SPAGV tem de cumprir as funções definidas na arquitectura apresentada em 2.2.1.

As suas principais funções são:

- simular a execução de movimentos de translacção por parte dos AGVs;
- simular a execução de movimentos de carga e de descarga por parte dos AGVs, sincronizando-os com o dispositivo do bloco 'Fábrica' (ver Figura 2.3 na página 11) também interveniente na operação;
- introduzir os AGVs (passagem do modo manual ao modo automático);
- simular a ocorrência de erros internos nos AGVs.

Na realização destas funções, o SPAGV suporta as mensagens descritas em 7.2.1.2.

7.2.4.1 Informação sobre um AGV a guardar no SPAGV

A informação sobre cada AGV, a que o SPAGV tem de ter acesso para cumprir as suas funções são:

- Identificador do AGV;
- Estado de movimentação;
- Estado anterior;
- Posição actual;
- Outras variáveis temporais utilizadas na simulação do seu comportamento.

7.2.4.2 Informação sobre o layout a guardar no SPAGV

O SPAGV necessita de ter acesso a informação relativa à estrutura do *layout* de AGVs, para ser capaz de determinar os tempos associados aos movimentos de translacção dos AGVs, com base das distâncias indicadas para cada ramo do *layout*. Esta necessidade advém do facto do SPAGV ter de simular o comportamento dos AGVs. Num SPAGV real, a informação do *layout* é irrelevante.

Para o SPAGV simular a introdução dos AGVs, tem de conhecer a probabilidade de ocorrência desse tipo de operação em cada um dos nós do *layout* (só para aqueles cuja probabilidade seja não nula). Esta informação é configurada num ficheiro, para que o SPAGV possa funcionar com qualquer *layout*.

8. *RP estocástica e colorida das 3 camadas de gestão de um SAGVs*

Neste capítulo será descrita a construção de um modelo das três camadas de gestão de um SAGVs, baseado em RP estocásticas e coloridas, utilizando a ferramenta Artifex. O modelo construído implementa a arquitectura definida em 2.2, de acordo com a especificação apresentada no capítulo 7.

A implementação dos objectos Artifex, que fazem parte do modelo, será descrita de forma sucinta, procurando descrever os aspectos mais essenciais do raciocínio seguido. As listagens detalhadas do código C embebido no modelo são apresentadas nos anexos C e D.

O modelo Artifex de um SAGVs inserido num sistema de fabrico é apresentado na Figura 8.1. Essa é a representação de mais alto nível do modelo construído. É constituída por quatro objectos, que modelizam o comportamento dos quatro blocos definidos pela arquitectura representada na Figura 2.3 (página 11).

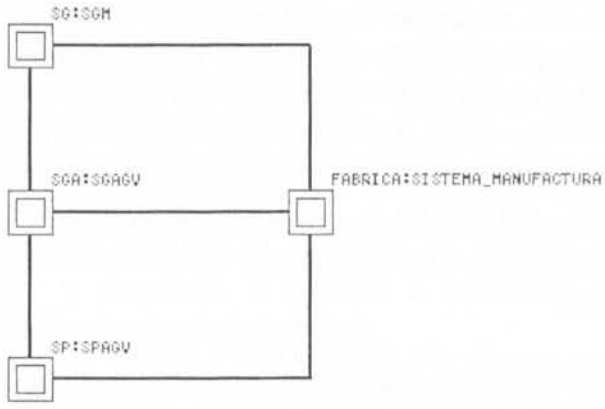


Figura 8.1 – Diagrama Artifex de um modelo de um SAGVs inserido num sistema de fabrico.

Os objectos constituintes do modelo são:

- SG – objecto da classe SGM, que modeliza o comportamento do SGM (camada superior da arquitectura);
- SGA – objecto da classe SGAGV, que modeliza o comportamento do SGAGV (camada intermédia da arquitectura);
- SP – objecto da classe SPAGV, que modeliza o comportamento do SPAGV (camada inferior da arquitectura);
- FABRICA – objecto da classe SISTEMA_MANUFACTURA, que modeliza o comportamento do sistema de fabrico.

A construção do modelo dos três primeiros objectos será descrita ao longo deste capítulo. A construção do modelo do quarto objecto é descrita no capítulo 9.

Como se pode observar na Figura 8.1, os fluxos de informação entre os vários blocos estão explicitamente representados pelas linhas que unem os quatro objectos.

8.1 Implementação de um temporizador fiável

O modelo do SAGVs construído tem de simular os tempos associados ao funcionamento do sistema, nomeadamente: o tempo de execução de um movimento, o intervalo de tempo entre avarias, o intervalo de tempo entre cancelamento de tarefas, etc.

Uma das características do Artifex é o facto dos *tokens* de uma posição serem consumidos segundo uma política FIFO (*first-in-first-out*).

Considere-se o exemplo seguinte. Há uma posição cujos *tokens* representam AGVs que iniciaram um movimento. Essa posição tem uma transição de saída, que é disparada quando o movimento termina, consumindo um *token* da posição. O atraso do disparo da transição é definido na execução da acção, que deposita AGVs na posição em causa, representando o tempo do movimento. Imagine-se que num instante t_1 é colocado um *token* de um AGV na posição, e que é configurado um tempo de disparo da transição de 5 s; imagine-se que 1 s depois, no instante t_2, é depositado um outro *token* de um AGV, e que é configurado um atraso de disparo de 2 s. Nesta situação, a transição será dispara em t_2+2 (antes de t_1+5), mas ao contrário do que seria de esperar, o *token* consumido é o que corresponde ao AGV que iniciou o movimento em $t=t_1$, devido à política FIFO da fila de *tokens*. Por outro lado, em $t=t_1+5$ ($t=t_2+4$) a transição é de novo disparada, consumindo desta vez o *token* depositado em $t=t_2$. Assim, este modelo simples do movimento dos AGVs não serve, porque os *tokens* representativos dos AGVs não são consumidos pela ordem correcta.

O exemplo anterior justifica a necessidade de modelização de um objecto que permita modelizar correctamente esse tipo de situações, que são muito frequentes ao longo do modelo do SAGVs construído.

O objecto que permite dar resposta a este tipo de situações é o que está representado pela RP da Figura 8.2 (ver informação detalhada no anexo C.1).

O objecto tem as seguintes entradas:

- TEMPO – recebe um *token* do exterior, que identifica uma nova temporização por um identificador único, e que define o tempo associado;
- ID_TEMPOR_INTERR – posição de entrada que permite a recepção de um *token*, identificando uma temporização que deva ser interrompida.

A saída ID_FIM_TEMPOR permite enviar um *token* ao exterior, identificando uma temporização que tenha terminado.

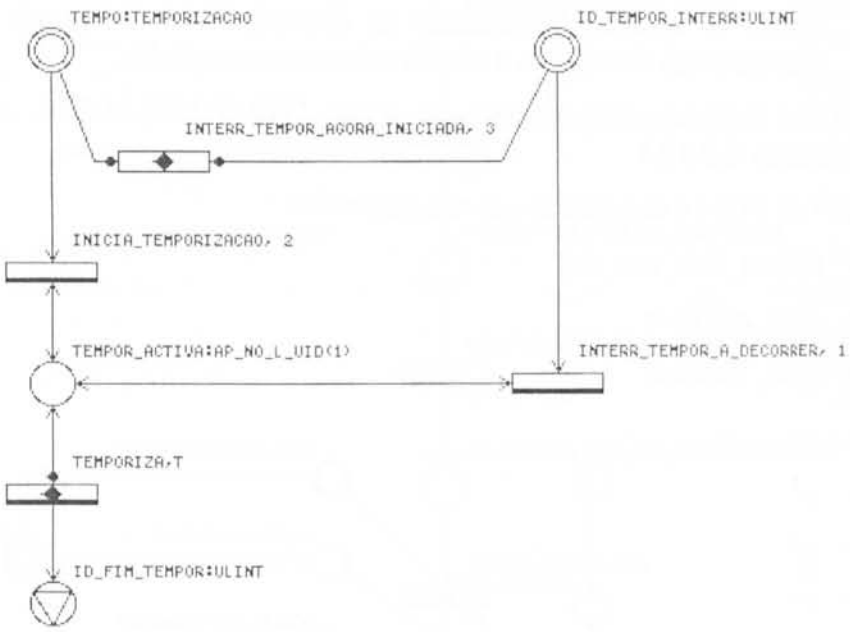


Figura 8.2 – RP de um objecto da classe TEMPORIZADOR.

O comportamento do objecto é tal, que o fim das várias temporizações recebidas é notificado ao exterior, mantendo a sua verdadeira ordem de causalidade. Para o efeito, o objecto implementa internamente uma lista ligada de temporizações, guardando para cada uma o seu identificador e o instante em que a temporização termina. Quando é recebida uma nova temporização, ou quando alguma temporização é interrompida, o objecto consulta essa lista, determina qual é a primeira temporização a terminar, configura convenientemente o atraso de disparo da transição 'TEMPORIZA', e coloca em 'TEMPOR_ACTIVA' um apontador para a temporização, que vai ser concluída em primeiro lugar. Assim, quando a transição 'TEMPORIZA' é disparada, o exterior é notificado correctamente sobre o identificador da temporização que acaba de terminar.

O código C associado às transições é apresentado no anexo C.1.8.

8.2 Classe SGM

A página principal da RP da classe SGM é apresentada na Figura 8.3 (ver informação detalhada no anexo C.2).

As posições de entrada definidas são:

- MENS_RECEB_SGAGV:MENS_SGM_SGAGV – posição através da qual são recebidos *tokens* contendo mensagens enviadas pelo SGAGV;
- NOVA_TAREFA_NORMAL:ORDEM – posição através da qual são recebidos *tokens* representando novas tarefas de movimentação solicitadas pelo sistema de fabrico.

As posições de saída definidas são:

- MENS_ENV_SGAGV:MENS_SGM_SGAGV – posição através da qual são enviados *tokens* ao exterior, contendo mensagens destinadas ao SGAGV;

- PONTO_CARGA_DISPONIVEL:ULINT – posição através da qual são enviados *tokens* ao exterior, informando da disponibilidade de pontos de carga que se encontravam destinados a tarefas entretanto canceladas.

O SGM instancia três objectos da classe 'TEMPORIZADOR'. A sua utilidade é descrita no anexo C.2.12.1.

A RP do SGM é constituída por três sub-redes:

- leitura_fich_nos_tar,
- gere_AGVs e
- gere_Tarefas.

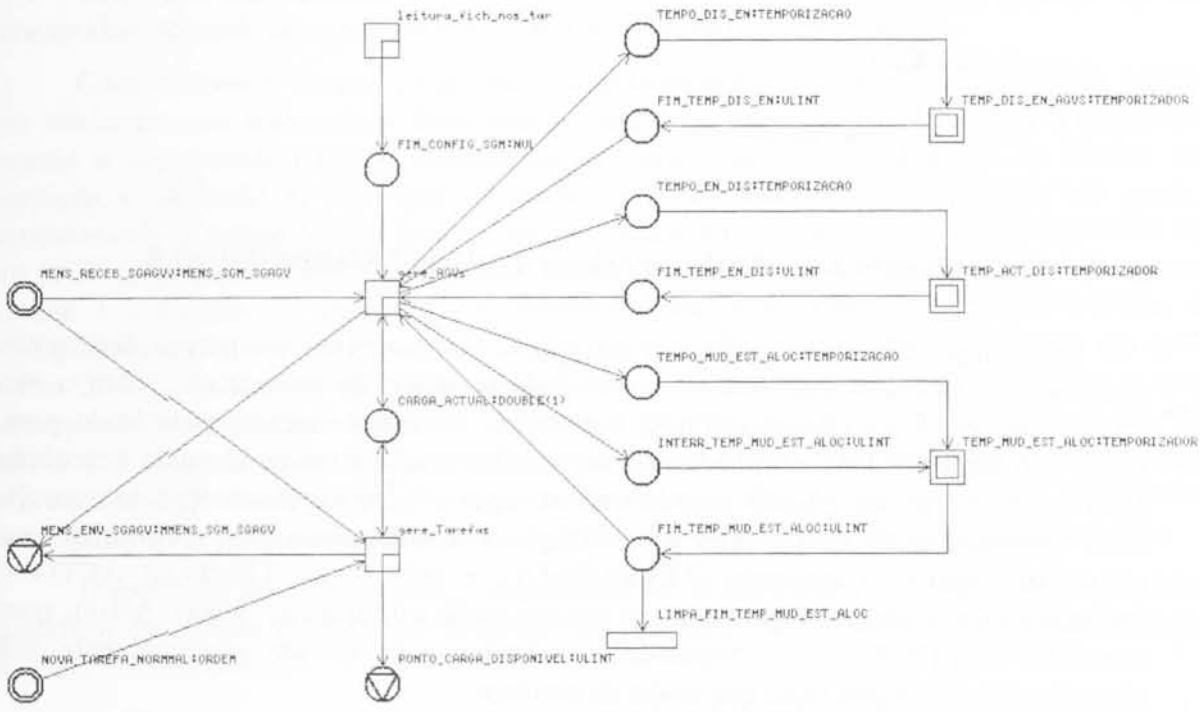


Figura 8.3 – Página principal da RP do SGM.

8.2.1 Sub-rede 'leitura_fich_nos_tar' do SGM

A sub-rede 'leitura_fich_nos_tar' é apresentada na Figura 8.4 (ver informação detalhada no anexo C.2.25).

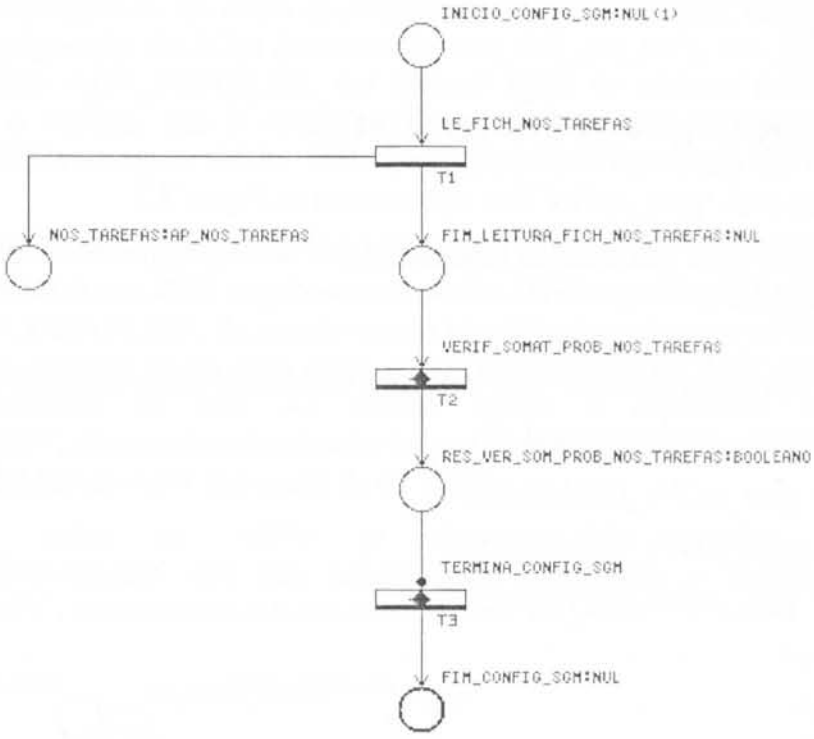


Figura 8.4 – Sub-rede 'leitura_fich_nos_tar' do SGM.

O *token* da posição 'INICIO_CONFIG_SGM', existente na marcação inicial, desencadeia o arranque do SGM. A transição 'LE_FICH_NOS_TAREFAS' efectua a leitura do ficheiro de configuração, que contém as probabilidades de ocorrência de tarefas de posicionamento em cada nó do *layout*. O *token* depositado na posição 'NOS_TAREFAS' permite aceder durante a simulação ao conteúdo da lista ligada utilizada para armazenar a informação lida.

O anexo F.5 contém um exemplo de um ficheiro de configuração das probabilidades de um nó ser alvo de uma introdução ou de um posicionamento de um AGV. Para cada nó, em que uma das probabilidades é não nula, existe uma linha com a sintaxe indicada na Tabela 8.1.

Tabela 8.1 – Sintaxe de uma linha do ficheiro de configuração de nós de introdução e de tarefas de posicionamento.

Sintaxe de uma linha	Significado
N:<id_no>,<p_introd>,<p_posic>	A probabilidade do nó <id_no> ser escolhido para introduzir um AGV é <p_introd>, e a probabilidade de uma tarefa de posicionamento lançada pelo SGM ter como destino o nó <id_no> é <p_posic>.

De seguida, a transição VERIF_SOMAT_PROB_NOS_TAREFAS verifica a coerência da informação contida no ficheiro, apurando se o somatório de todas as probabilidades é unitário. Se o resultado do teste for favorável, a transição

'TERMINA_CONFIG_SGM' é disparada, pondo termo ao processo de configuração do SGM.

8.2.2 Sub-rede 'gere_AGVs' do SGM

A sub-rede 'gere_AGVs' está representada na Figura 8.5.

Esta sub-rede modeliza as funcionalidades do SGM, que dizem respeito à gestão dos AGVs, nomeadamente a activação e desactivação de um AGV e a monitorização do estado de cada AGV.

Por sua vez, a sub-rede 'gere_AGVs' é constituída pelas seguintes sub-redes:

- gera_AGVs_DISABLED,
- gere_AGVs_DISABLED,
- gere_AGVs_ENABLED e
- gere_AGVs_ACTIVOS.

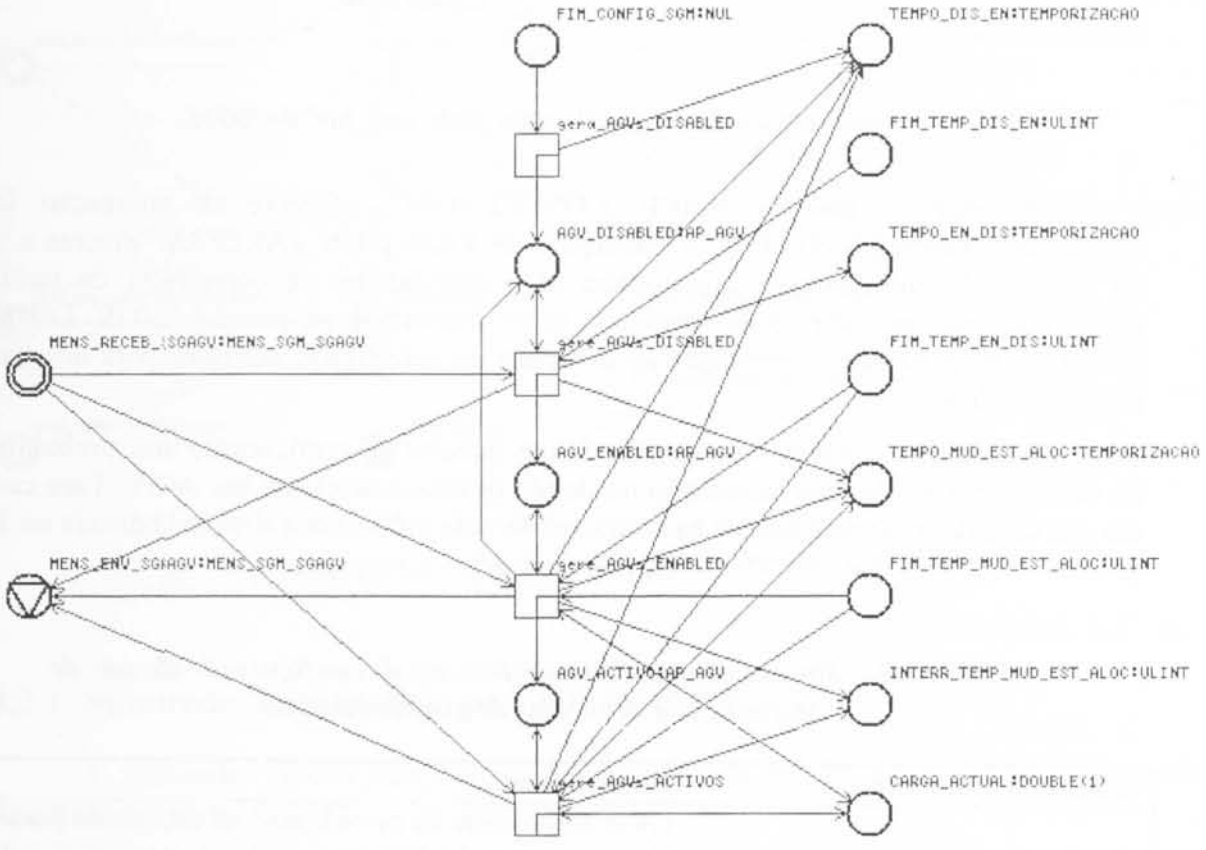


Figura 8.5 – Sub-rede 'gere_AGVs' do SGM.

8.2.2.1 Sub-rede 'gera_AGVs_DISABLED' do SGM

A sub-rede 'gera_AGVs_DISABLED' está representada na Figura 8.6 (ver informação detalhada no anexo C.2.13).

Após a deposição de um *token* na posição FIM_CONFIG_SGM, resultante do fim do processo de configuração do SGM (sub-rede 'leitura_fich_nos_tar'), esta sub-rede deposita *tokens* na posição AGV_DISABLED, em número igual ao número máximo de AGVs previstos para o sistema, que é configurável através do parâmetro 'numero_agvs' (ver parâmetros do SGM no anexo C.2.4). Cada *token* depositado representa um AGV no estado DISABLED.

A transição 'INIC_ID_AGV' deposita um *token* em 'ID_AGV', contendo o identificador do primeiro AGV a gerar. A transição 'GERA_AGV_DISABLED' gera um AGV em 'AGV_DISABLED', de acordo com o identificador existente no *token* consumido de 'ID_AGV', e deposita aí um novo *token* com o identificador do AGV seguinte (igual ao anterior incrementado de um). Ao mesmo tempo, é depositado um *token* em 'TEMPO_DIS_EN', desencadeando o início de uma temporização, que permitirá a passagem ao estado ENABLED do AGV que acaba de ser gerado.

Quando todos os AGVs já tiverem sido gerados, a transição 'GERA_AGV_DISABLED' não fica sensibilizada, permitindo o disparo da transição 'LIMPA_ID_AGV', que consome o *token* existente em 'ID_AGV'.

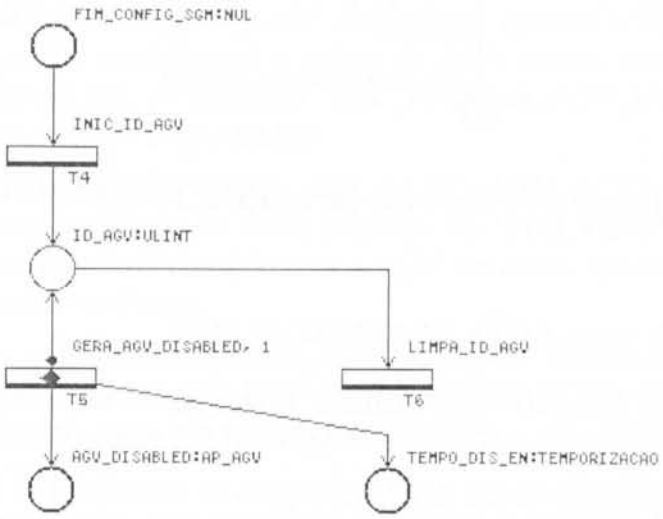


Figura 8.6 – Sub-rede 'gera_AGVs_DISABLED' do SGM.

8.2.2.2 Sub-rede 'gere_AGVs_DISABLED' do SGM

A sub-rede 'gere_AGVs_DISABLED' está representada na Figura 8.7 (ver informação detalhada no anexo C.2.15). Esta modeliza a evolução de estado que um AGV no estado DISABLED pode sofrer.

Quando é depositado um *token* em 'FIM_TEM_DIS_EN', significando que chegou o momento de activar o AGV em questão, a transição 'ENVIA_ENABLE_AGV' é disparada. Esta solicita ao SGAGV a passagem do AGV ao estado ENABLED, através do envio de uma mensagem do tipo 'ENABLE_AGV'.

Se o SGAGV confirmar a activação do AGV, a transição 'REC_CONF_SUC_EN' é disparada, depositando o AGV na posição 'AGV_ENABLED', pois este acaba de passar ao estado ENABLED. Ao mesmo tempo, são iniciadas duas temporizações para esse AGV: uma

que poderá levar o AGV a regressar ao estado DISABLED e outra que pode provocar a inversão do seu estado de alocação.

Se o SGAGV reportar algum erro, a transição 'REC_CONF_INSUC_EN' é disparada. Esta repõe o *token* do AGV na posição 'AGV_DISABLED' e inicia uma nova temporização, que levará a uma nova tentativa futura de activação do AGV.

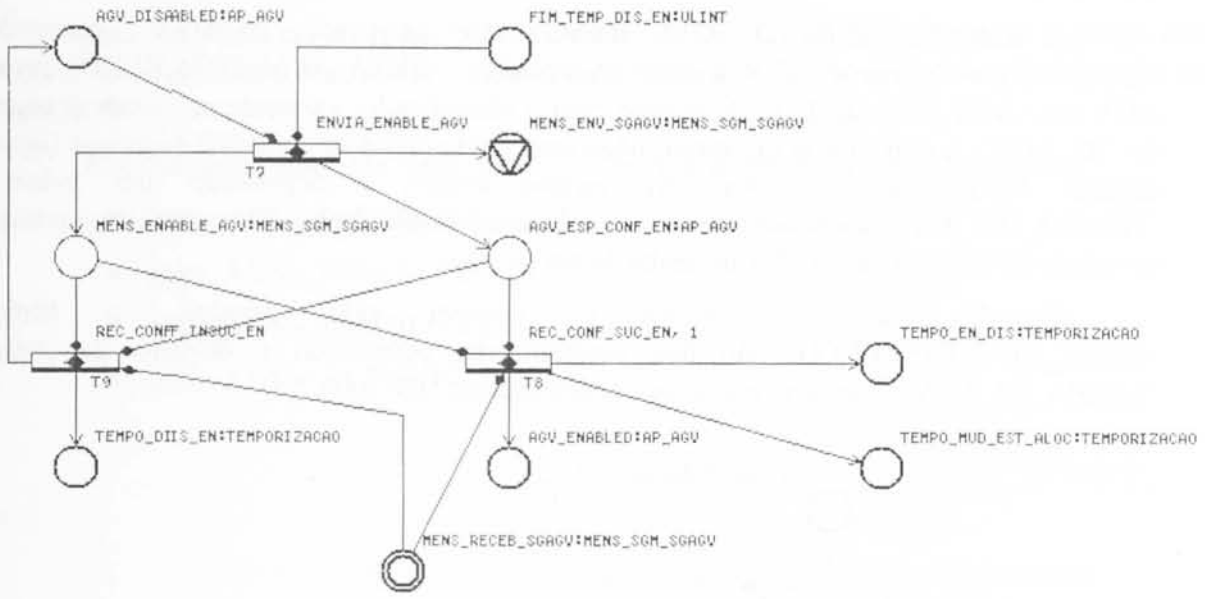


Figura 8.7 – Sub-rede 'gere_AGVs_DISABLED' do SGM.

8.2.2.3 Sub-rede 'gere_AGVs_ENABLED' do SGM

A sub-rede 'gere_AGVs_ENABLED' está representada na Figura 8.8. Esta modeliza a evolução de estado que um AGV no estado ENABLED pode sofrer.

Após a introdução de um AGV, a recepção da mensagem do tipo 'ESTADO_AGV' de alteração do seu estado provoca o disparo da transição 'REC_PASSAG_AGV_AUTOM'. Esta deposita um *token* do AGV na posição 'AGV_ACTIVO' e actualiza a carga do sistema (*token* depositado em 'CARGA_ACTUAL').

O disparo da transição 'MUDA_EST_ALOC_AGV_EN' inverte o estado de alocação do AGV, indicado pelo *token* recebido em 'FIM_TEM_MUD_EST_ALOC', e informa o SGAGV desse facto, através do envio de uma mensagem do tipo 'ENABLE_AGV'.

O disparo da transição 'DISABLE_AGV_ENABLED' coloca o AGV, indicado pelo *token* recebido em 'FIM_TEMP_EN_DIS', no estado ENABLED, informa o SGAGV desse facto, através do envio de uma mensagem do tipo 'DISABLE_AGV', e inicia uma temporização para o AGV em causa, que levará o AGV a regressar futuramente ao estado ENABLED.

O código C associado a estas transições é apresentado no anexo C.2.16.

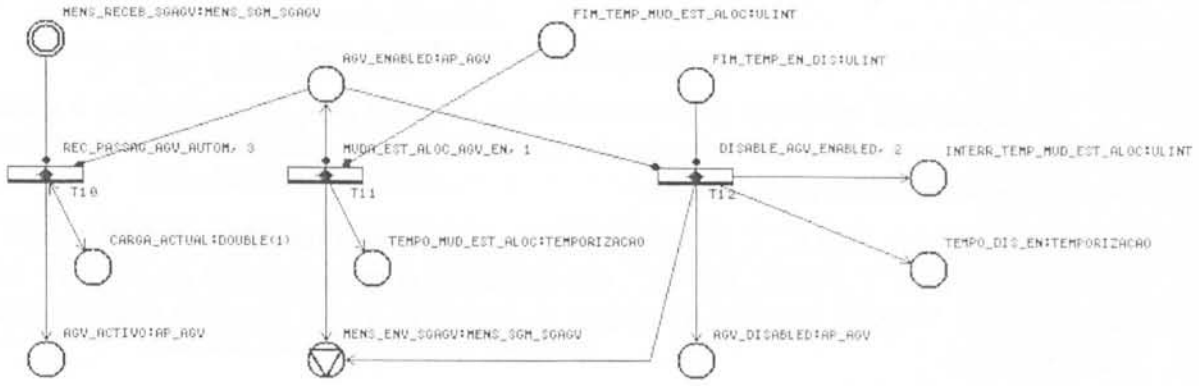


Figura 8.8 – Sub-rede 'gere_AGVs_ENABLED' do SGM.

8.2.2.4 Sub-rede 'gere_AGVs_ACTIVOS' do SGM

A sub-rede 'gere_AGVs_ACTIVOS' está representada na Figura 8.9.

A transição 'REC_MUD_EST_AGV' é disparada, quando o SGM recebe mensagens do tipo 'ESTADO_AGV', actualizando o estado do AGV activo, desde que o novo estado seja diferente de ENABLED. A transição 'REC_PASSAG_AGV_MANUAL' é disparada quando o novo estado do AGV é ENABLED.

O disparo da transição 'MUDA_EST_ALOC_AGV_ACT' inverte o estado de alocação do AGV, correspondente ao *token* recebido em 'FIM_TEMP_MUD_EST_ALOC'. Para além disso, inicia uma temporização para o AGV em causa, que permitirá nova inversão do seu estado de alocação no futuro.

O disparo da transição 'DISABLE_AGV_ACTIVO' coloca o AGV indicado pelo *token* recebido em 'FIM_TEMP_EN_DIS' no estado DISABLED. Para além disso, inicia uma temporização para o AGV em causa, que permitirá a sua passagem futura ao estado ENABLED.

O código C associado a estas transições é apresentado no anexo C.2.14.

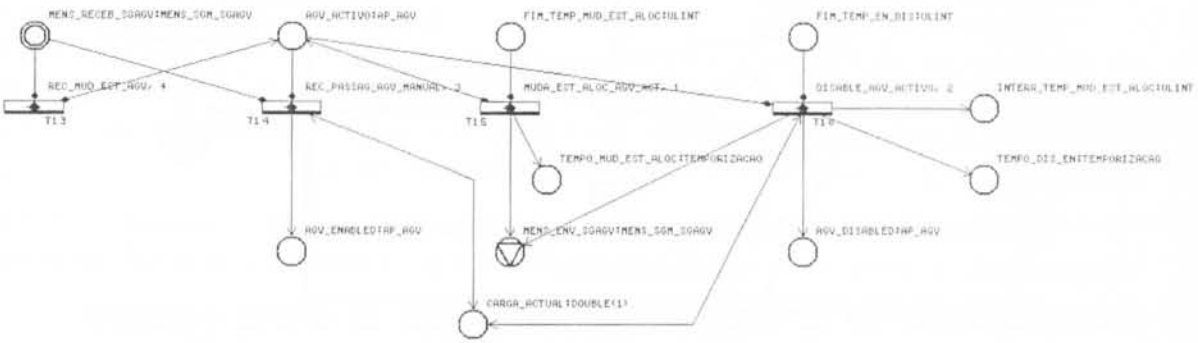


Figura 8.9 – Sub-rede 'gere_AGVs_ACTIVOS' do SGM.

8.2.3 Sub-rrede 'gere_Tarefas' do SGM

A sub-ı-rede 'gere_Tarefas' está representada na Figura 8.10.

Esta ssub-rede modeliza as funcionalidades do SGM que dizem respeito à gestão de tarefas, nomeeadamente o lançamento de novas tarefas, o cancelamento de tarefas e a monitorizaçãoo do estado de cada tarefa.

A trannsição 'PROC_T_POSIC_ESPEC' é disparada quando o SGM recebe uma mensagem doo tipo 'T_POSIC_ESPEC', que notifica a criação por parte do SGAGV de uma tarefa do tipoo 'POSIC_ESPEC'. O código C associado a esta transição é apresentado no anexo C.2.17.1.1.

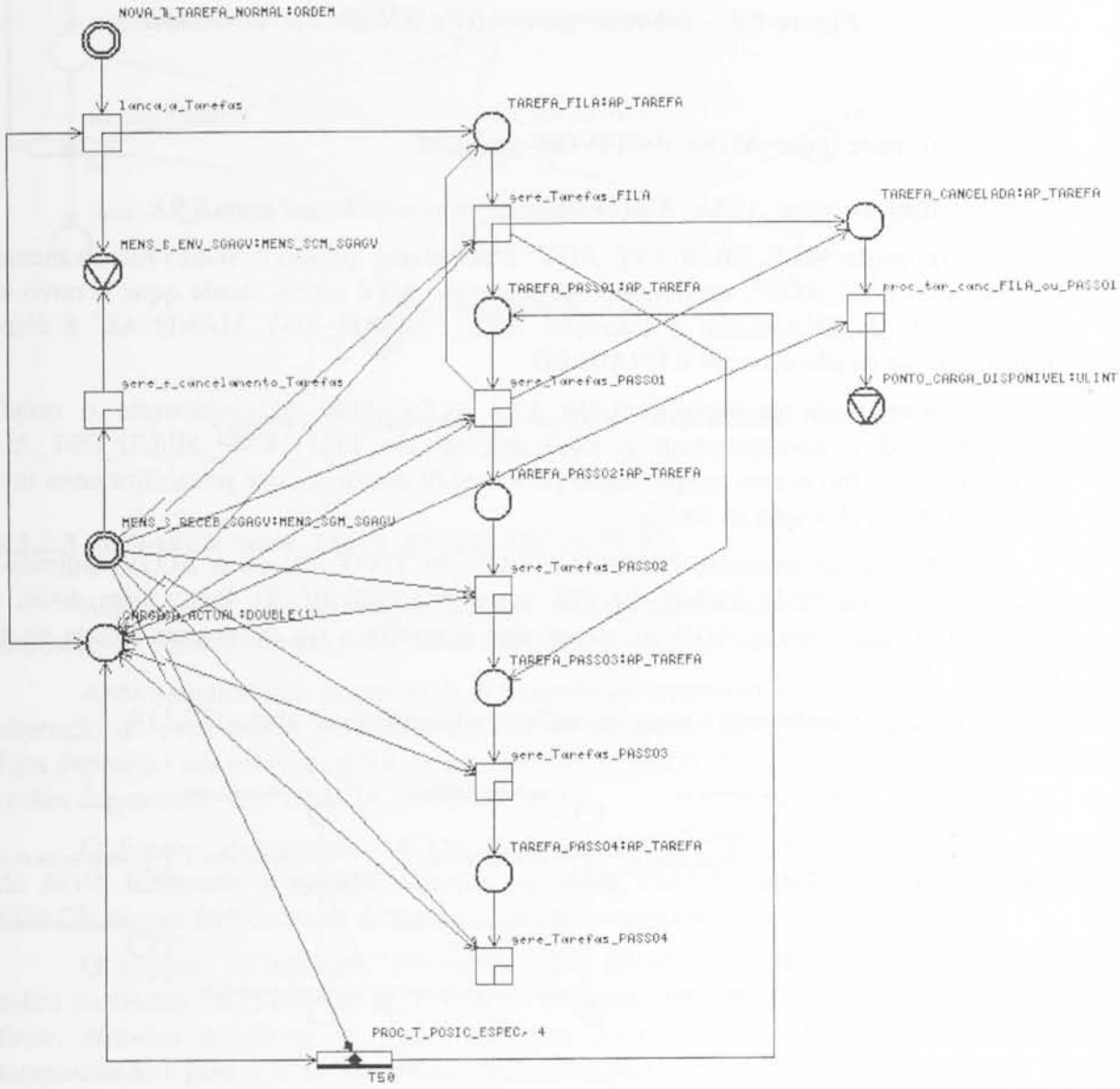


Figura 8.10 – Sub-rede 'gere_Tarefas' do SGM.

Por suua vez, a sub-rede 'gere_Tarefas' é constituída pelas seguintes sub-redes:

- laanca_Tarefas,

- gere_cancelamento_Tarefas,
- gere_Tarefas_FILA,
- gere_Tarefas_PASSO1,
- gere_Tarefas_PASSO2,
- gere_Tarefas_PASSO3,
- gere_Tarefas_PASSO4 e
- proc_tar_canc_FILA_ou_PASSO1.

8.2.3.1 Sub-rede 'lanca_Tarefas' do SGM

A sub-rede 'lanca_Tarefas' está representada na Figura 8.11 (ver informação detalhada no anexo C.2.24). Esta modeliza o envio de tarefas de movimentação para o SGAGV, quer tarefas lançadas e escalonadas pelo próprio SGM, quer tarefas provenientes do sistema de fabrico e escalonadas pelo SGAGV.

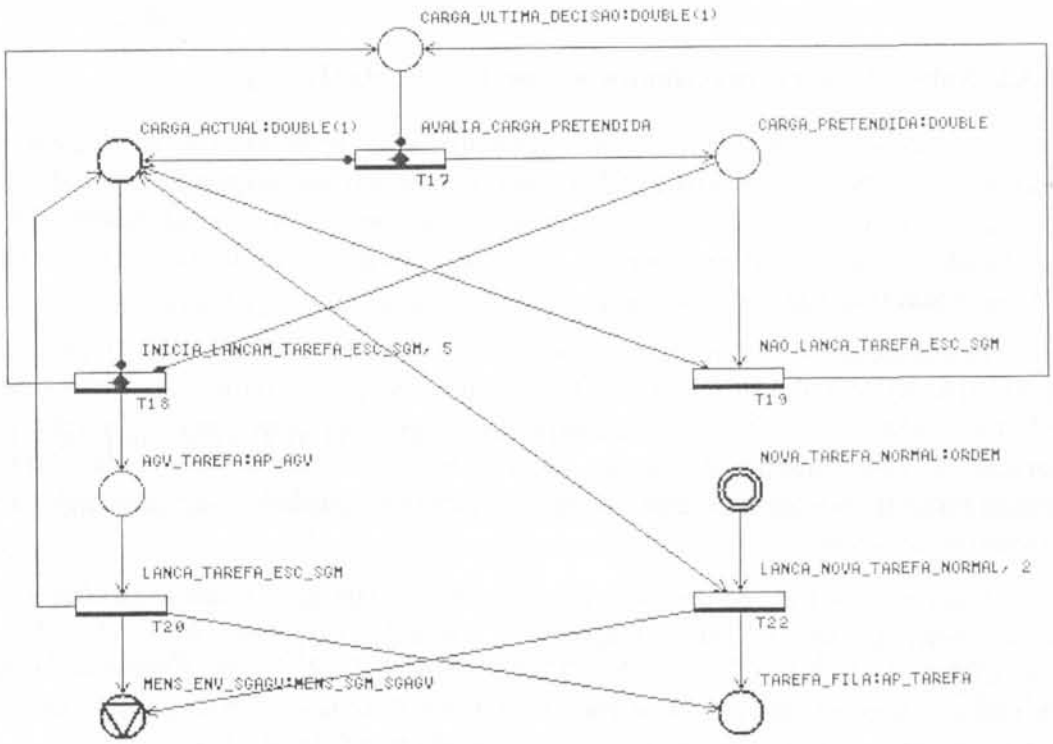


Figura 8.11 – Sub-rede 'lanca_Tarefa' do SGM.

Quando o sistema de fabrico requisita uma tarefa de movimentação, através da deposição de um *token* em 'NOVA_TAREFA_NORMAL', é disparada a transição 'LANCA_NOVA_TAREFA_NORMAL'. Esta envia ao SGAGV a mensagem do tipo 'TAREFA_NORMAL' correspondente.

A parte restante da RP simula o lançamento de tarefas do tipo 'POSICION', com base na FDP normal da carga pretendida para o sistema (ver parâmetros do SGM no anexo C.2.4). Quando o *token* representando a carga actual vê o seu valor alterado, devido ao fim da

execução de uma tarefa, ou ao cancelamento de uma tarefa, ou à activação ou desactivação de um AGV, a transição 'AVALIA_CARGA_PRETENDIDA' é disparada, estimando um valor para a carga pretendida, representada pelo valor do *token* depositado em 'CARGA_PRETENDIDA'.

Se o valor da carga actual for menor do que o valor da carga pretendida, a transição 'INICIA_LANCAM_TAREFA_ESC_SGM' é disparada, caso contrário é disparada a transição 'NÃO_LANCA_TAREFA_ESC_SGM'. No primeiro caso, a transição disparada deposita um *token* em 'AGV_TAREFA', representando o AGV que vai ser afectado à nova tarefa a lançar. O disparo da transição 'LANCA_TAREFA_ESC_SGM' desencadeia a actualização do valor da carga actual do sistema e o envio ao SGAGV de uma mensagem do tipo 'TAREFA_POSICION'.

Embora o protocolo definido na especificação (ver 7.2.2) preveja a existência de outros tipos de tarefas escalonadas pelo SGM para além das tarefas de posicionamento, o lançamento dessas tarefas acabou por não ser implementado, por se tratar de uma funcionalidade dispensável para o tipo de análise que se pretendia fazer com base no modelo desenvolvido.

Quando uma tarefa é lançada, a posição 'TAREFA_FILA' recebe um *token*, que aponta para o elemento da lista ligada de tarefas do SGM representando a nova tarefa.

8.2.3.2 Sub-rede 'gere_cancelamento_Tarefas' do SGM

A sub-rede 'gere_cancelamento_Tarefas' está representada na Figura 8.12 (ver informação detalhada no anexo C.2.23). Esta modeliza o cancelamento esporádico de tarefas pendentes dos SAGVs. O tempo entre duas operações de cancelamento de tarefas é modelizado por uma FDP exponencial, cujo valor médio é configurável (ver parâmetros do SGM no anexo C.2.4).

Inicialmente, o *token* da posição 'FIM_CANCEL_TAREFAS' dispara a transição 'PLANEIA_NOVO_CANCEL_TAREFAS', que configura o tempo de disparo da transição 'INICIA_CANCEL_TAREFAS', ou seja, o tempo que falta para a ocorrência de uma operação de cancelamento de tarefas. Quando esta transição dispara, é decidido o tipo de cancelamento de tarefas a efectuar, com base nas probabilidades definidas pelos respectivos parâmetros do SGM.

Conforme se trate de um cancelamento de uma tarefa, de todas as tarefas em execução, ou de todas as tarefas sem excepção, a transição disparada é 'CANCELA_1_TAR', 'CANCELA_TAR_EXEC' ou 'CANCELA_TODAS_TAR', respectivamente. Em qualquer dos casos, é depositado um *token* em 'ID_MENS_CANCEL_TAR' com o identificador da mensagem do tipo 'CANCELA_TAREFA' enviada ao SGAGV.

A transição 'REC_CONF_CANCEL_TAR' recebe a confirmação da operação de cancelamento (mensagem do tipo 'CONFIRMACAO'). De seguida, o próximo cancelamento é planeado, repetindo-se o processamento atrás descrito.

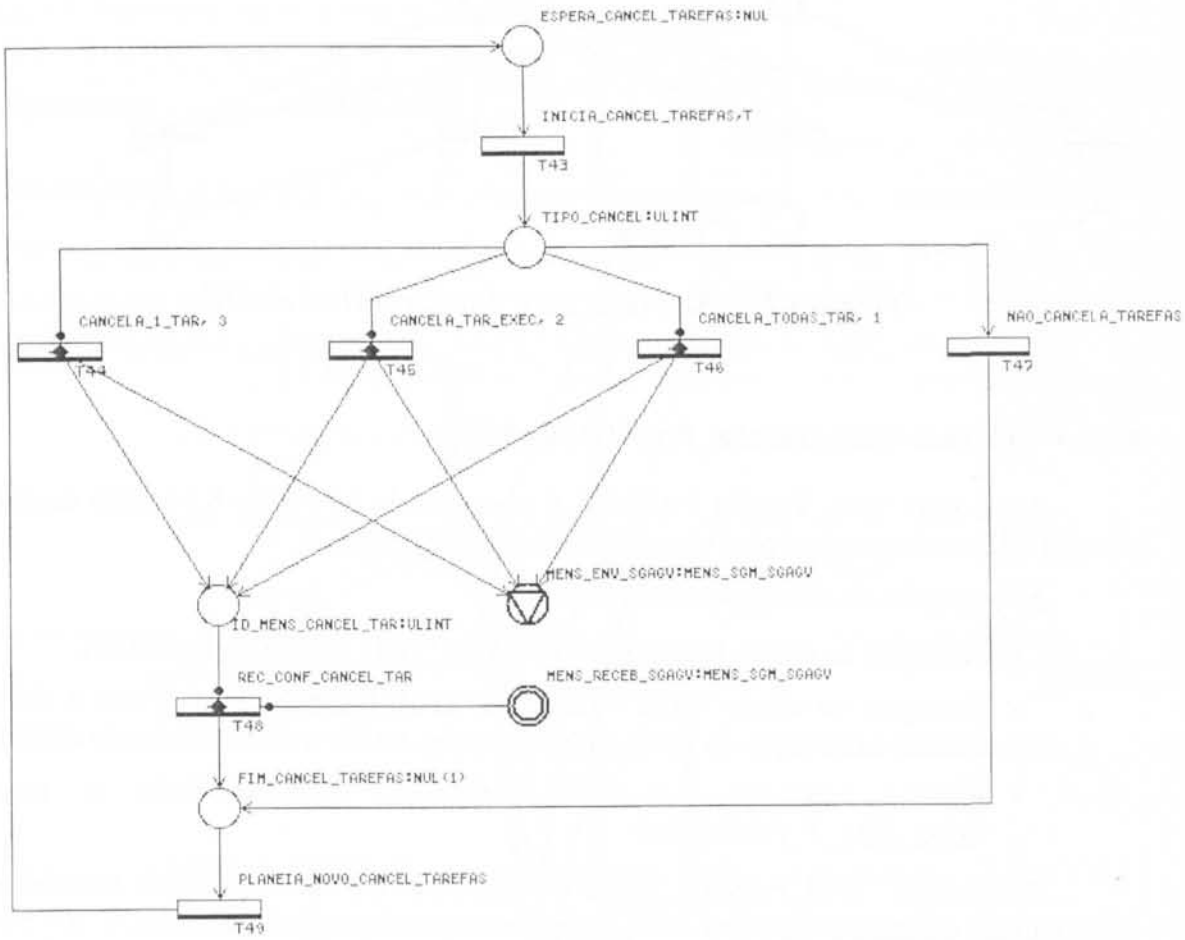


Figura 8.12 – Sub-rede 'gere_cancelamento_Tarefas' do SGM.

8.2.3.3 Sub-rede 'gere_Tarefas_FILA' do SGM

A sub-rede 'gere_Tarefas_FILA' está representada na Figura 8.13. Esta modeliza as evoluções de estado possíveis para uma tarefa no estado FILA.

As evoluções possíveis são:

- Passagem ao estado PASSO_1 – transição 'TAR_FILA_PASSO1';
- Passagem ao estado PASSO_3 de uma tarerfa do tipo 'T_DESCARGA – transição 'TAR_FILA_PASSO3'.

A transição 'REC_CANCEL_TAR_FILA' é disparada quando uma tarefa é cancelada (recepção de uma mensagem do tipo 'TAREFA_CANCEL').

A transição 'TAR_FILA_REJEITADA' é disparada quando a tarefa em causa não puder ser executada, devido a alguma incoerência (e.g. ponto de carga inexistente).

O código C associado a estas transições é apresentado no anexo C.2.18.

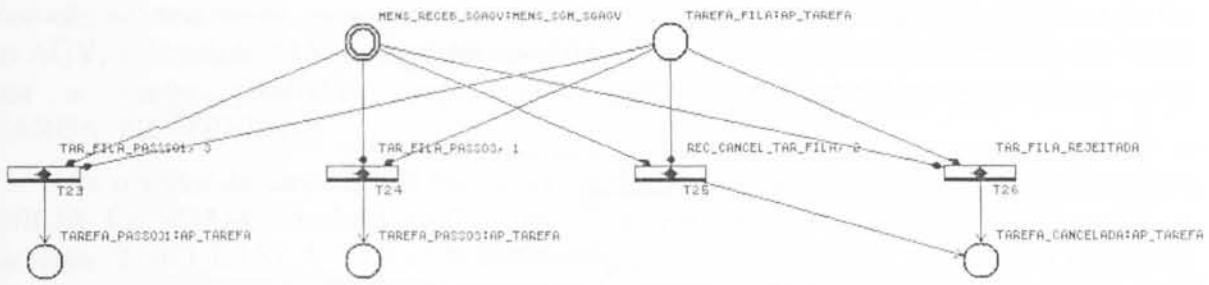


Figura 8.13 – Sub-rede 'gere_Tarefas_FILA' do SGM.

8.2.3.4 Sub-rede 'gere_Tarefas_PASSO1' do SGM

A sulb-rede 'gere_Tarefas_PASSO1' é representada na Figura 8.14. Esta modeliza a evolução de estado possível para uma tarefa no estado PASSO_1.

As ewoluções de estado possíveis são:

- Passagem ao estado PASSO_2 – transição 'TAR_PASSO1_PASSO2';
- Passagem ao estado FILA – transição 'TAR_PASSO1_FILA'; esta é disparada quando a execução da tarefa é interrompida, devido a uma reafectação dinâmica;
- Fim da execução de uma tarefa de posicionamento – transição "REC_FIM_T_POSICION'.

A transição 'TAR_PASSO1_REAF' é disparada quando uma tarefa mantém o seu estado, mas é afectada a um outro AGV, devido a uma reafectação dinâmica.

A transição 'REC_CANCEL_TAR_PASSO1' é disparada, quando uma tarefa é cancelada (recepção de uma mensagem do tipo 'TAREFA_CANCEL').

A transição 'REC_REJEICAO_TAR_PASSO1' é disparada quando a tarefa em causa não puder ser executada, devido à impossibilidade do cálculo do trajecto. Isto poderá ocorrer quando uma tarefa é reafectada.

O código C associado a estas transições é apresentado no anexo C.2.19.

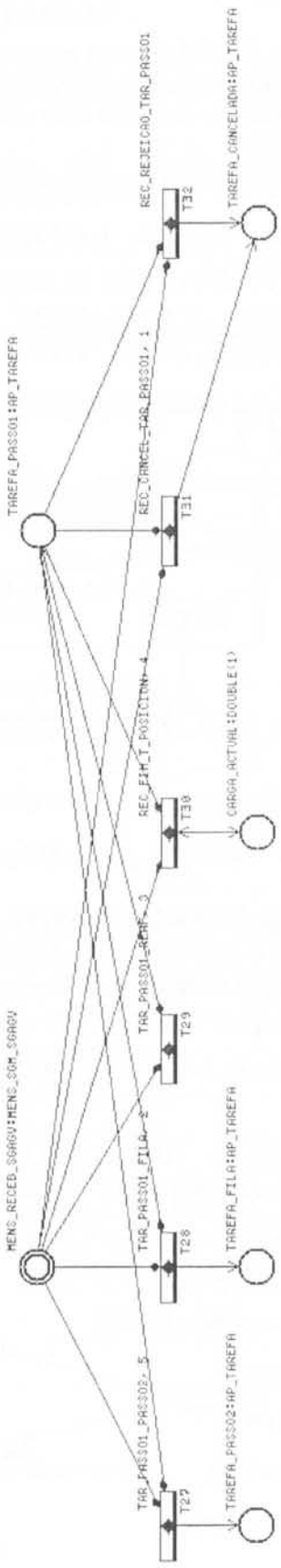


Figura 8.14 – Sub-rede 'gere_Tarefas_PASSO1' do SGM

8.2.3.5 Sub-rede 'gere_Tarefas_PASSO2' do SGM

A sub-rede 'gere_Tarefas_PASSO2' está representada na Figura 8.15. Esta modeliza as evoluções de estado possíveis para uma tarefa no estado PASSO_2.

As evoluções possíveis são:

- Passagem ao estado PASSO_3 – transição 'TAR_PASSO2_PASSO3';
- Fim de uma tarefa do tipo T_CARGA ou CARGA_AGV – transição 'REC_FIM_T_CARGA'.

A transição 'REC_CANCEL_TAR_PASSO2' é disparada quando uma tarefa é cancelada (recepção de uma mensagem do tipo 'TAREFA_CANCEL').

O código C associado a estas transições é apresentado no anexo C.2.20.

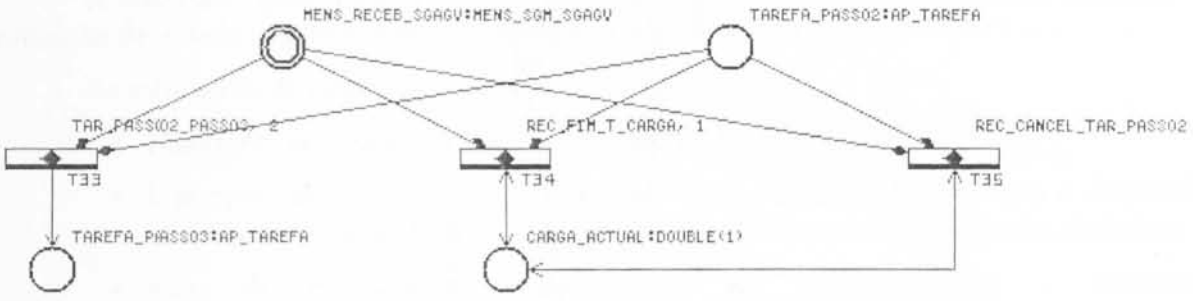


Figura 8.15 – Sub-rede 'gere_Tarefas_PASSO2' do SGM.

8.2.3.6 Sub-rede 'gere_Tarefas_PASSO3' do SGM

A sub-rede 'gere_Tarefas_PASSO3' está representada na Figura 8.16. Esta modeliza as evoluções de estado possíveis para uma tarefa no estado PASSO_3.

A passagem de uma tarefa ao estado PASSO_3 provoca o disparo da transição 'TAR_PASSO2_PASSO3'.

A transição 'REC_CANCEL_TAR_PASSO3' é disparada quando uma tarefa é cancelada (recepção de uma mensagem do tipo 'TAREFA_CANCEL').

O código C associado a estas transições é apresentado no anexo C.2.21.

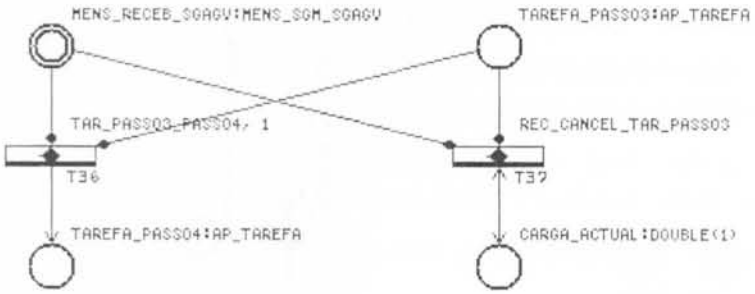


Figura 8.16 – Sub-rede 'gere_Tarefas_PASSO3' do SGM.

8.2.3.7 Sub-rede 'gere_Tarefas_PASSO4'

A sub-rede 'gere_Tarefas_PASSO4' está representada na Figura 8.17. Esta modeliza as evoluções de estado possíveis para uma tarefa no estado PASSO_4.

O fim da execução de uma tarefa provoca o disparo da transição 'REC_FIM_T_NORMAL_OU_DESCARGA'.

A transição 'REC_CANCEL_TAR_PASSO4' é disparada quando uma tarefa é cancelada (recepção de uma mensagem do tipo 'TAREFA_CANCEL').

O código C associado a estas transições é apresentado no anexo C.2.22.

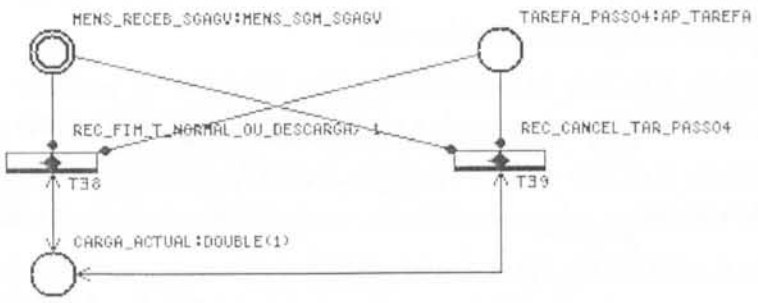


Figura 8.17 – Sub-rede 'gere_Tarefas_PASSO4' do SGM.

8.2.3.8 Sub-rede 'proc_tar_canc_FILA_ou_PASSO1' do SGM

A sub-rede 'proc_tar_canc_FILA_ou_PASSO1' está representada na Figura 8.18 (ver informação detalhada no anexo C.2.26). Esta modeliza o processamento efectuado em tarefas que sejam canceladas, quando se encontrem no estado FILA ou PASSO1.

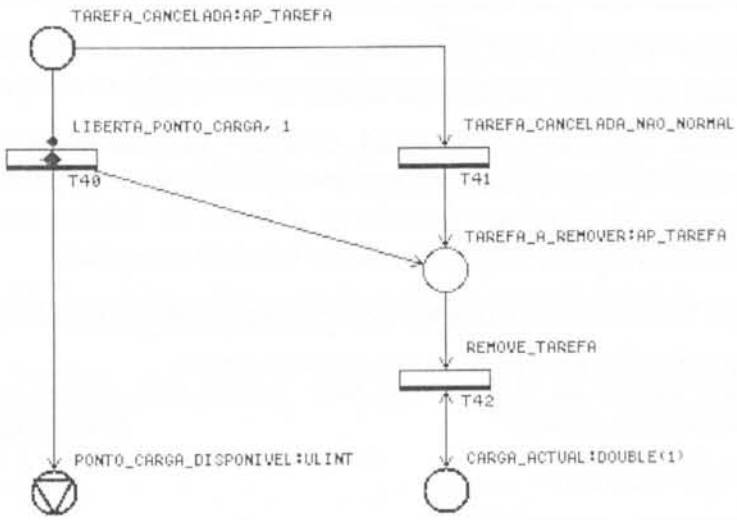


Figura 8.18 – Sub-rede 'proc_tar_canc_FILA_ou_PASSO1' do SGM.

Se a tarefa for do tipo 'NORMAL', a transição 'LIBERTA_PONTO_CARGA' é disparada, depositando um *token* em 'PONTO_CARGA_DISPONIVEL', que notificará o sistema de fabrico da libertação do ponto de carga respectivo. Se a tarefa for do tipo 'POSICION', a transição 'TAREFA_CANCELADA_NÃO_NORMAL' é disparada.

A transição 'REMOVE_TAREFA' efectua a remoção da tarefa da lista ligada de tarefas do SGM e a actualização do valor da carga actual do sistema.

8.3 Classe SGAGV

A página principal da RP da classe SGAGV é apresentada na Figura 8.19 (ver informação detalhada no anexo C.3).

As posições de entrada definidas são:

- MENS_RECEB_SGM:MENS_SGM_SGAGV – posição através da qual são recebidos *tokens* do exterior, contendo mensagens enviadas pelo SGM;
- MENS_RECEB_SPAGV:MENS_SGAGV_SPAGV – posição através da qual são recebidos *tokens* do exterior, contendo mensagens enviadas pelo SPAGV;
- CAPACIDADE_FILA:CAPC_FILA – posição através da qual são recebidos *tokens* do sistema de fabrico, durante a inicialização do sistema, com a capacidade da fila de saída existente em cada ponto de carga;
- FILA_PALETE_INSERIDA:ULINT – posição através da qual são recebidos *tokens* do sistema de fabrico, informando da deposição de paletes na fila de saída dos pontos de carga.

As posições de saída definidas são:

- MENS_ENV_SGM:MENS_SGM_SGAGV – posição através da qual são enviados *tokens* ao exterior, contendo mensagens destinadas ao SGM;
- MENS_ENV_SPAGV:MENS_SGAGV_SPAGV – posição através da qual são enviados *tokens* ao exterior, contendo mensagens destinadas ao SPAGV.

A transição 'INICIALIZA_ESTADO_FILA' é disparada quando o SGAGV recebe do sistema de fabrico um *token*, informando a capacidade da fila de saída de um ponto de carga.

A transição 'ACT_ESTADO_FILA_CARGA' permite actualizar o estado da fila de saída de um ponto de carga, quando for colocada aí uma palete pela respectiva entidade do sistema de fabrico. É disparada quando o sistema de fabrico envia um *token* com o identificador da fila de saída, onde acaba de ser inserida uma palete.

O código C associado a estas transições é apresentado no anexo C.3.12.2.

A RP do SGAGV é constituída por sete sub-redes:

- leitura_fich_config,
- cria_lista_AGVs,
- proc_mens_receb_SGM,
- cancela_movim_avan_AGVs,
- escalona_e_daspacha_tar_normais,

- gere_AGVs e
- proc_tarefas_AGVs_desactivados.

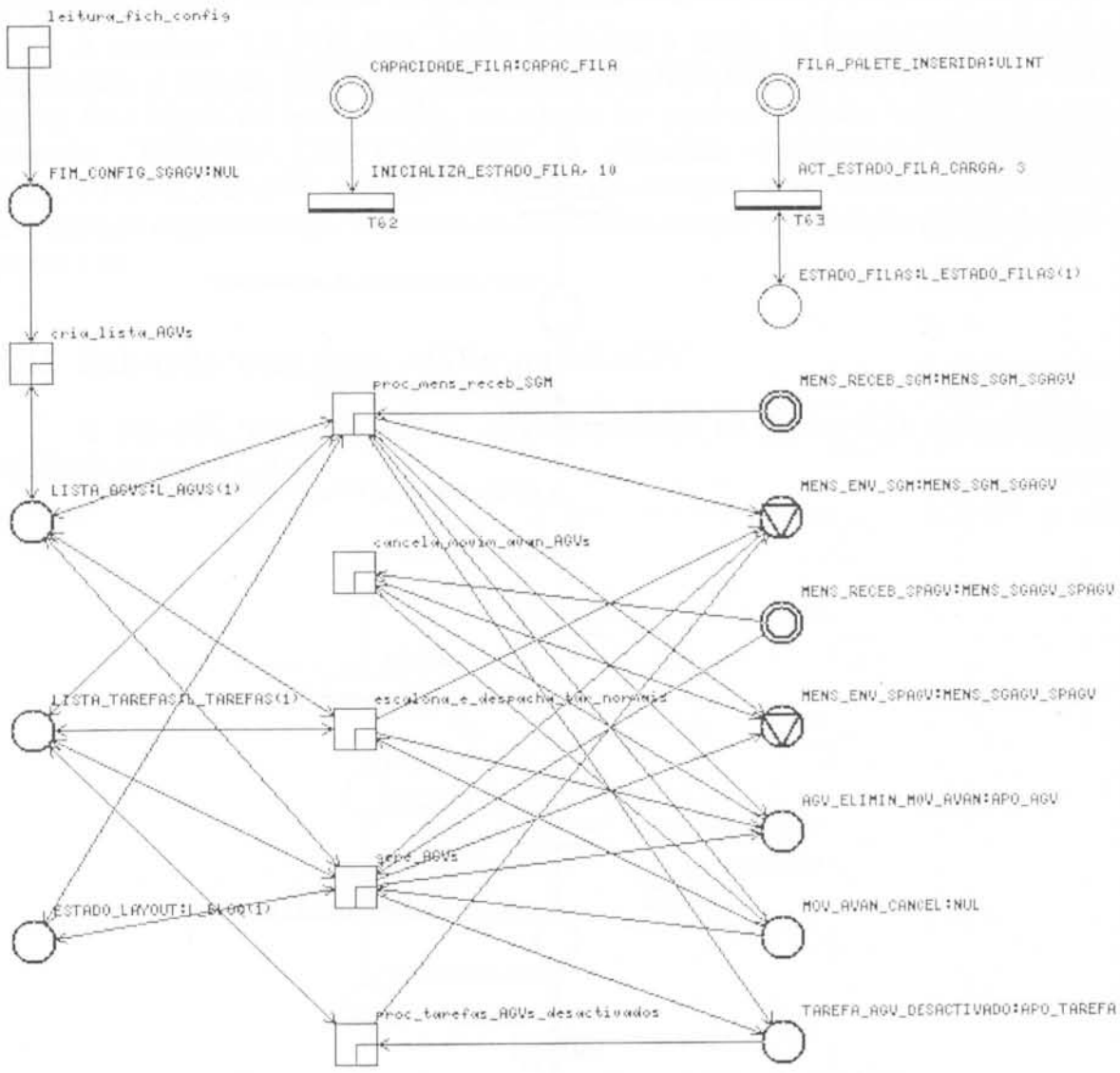


Figura 8.19 – Página principal da RP do SGAGV.

8.3.1 Sub-rede 'leitura_fich_config' do SGAGV

A sub-rede 'leitura_fich_config' está representada na Figura 8.20 (ver informação detalhada no anexo C.3.19).

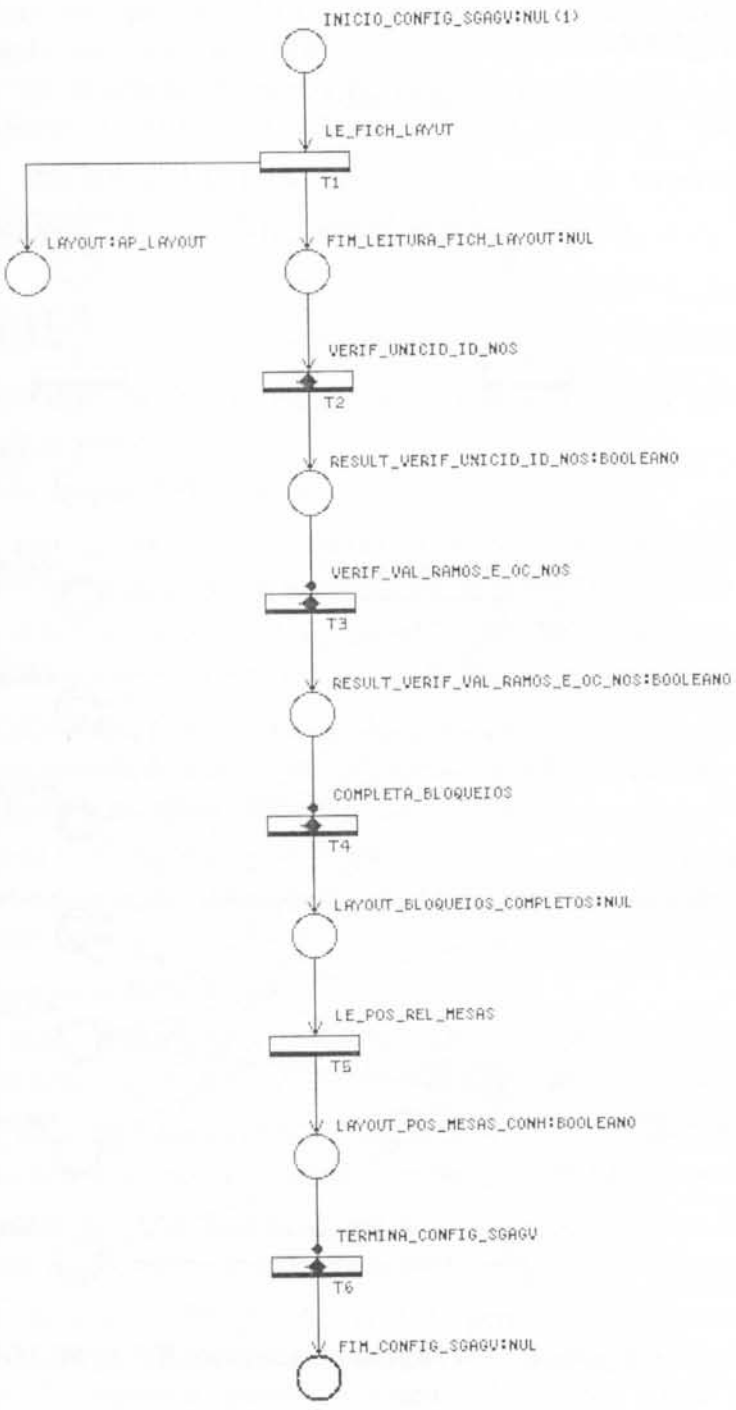


Figura 8.20 – Sub-rede 'leitura_fich_config' do SGAGV.

O *token* da posição 'INICIO_CONFIG_SGAGV', existente na marcação inicial, desencadeia o arranque do SGAGV. A transição 'LE_FICH_LAYOUT' efectua a leitura do ficheiro de configuração com a estrutura do *layout*. O *token* depositado na posição 'LAYOUT' permite aceder durante a simulação ao conteúdo da lista ligada utilizada para representar a estrutura do *layout*. A sintaxe do ficheiro de configuração de *layout* é a que está definida na Tabela 3.6 da página 26.

De seguida, a transição 'VERIF_UNICID_ID_NOS' verifica se não há nós com identificadores repetidos. Se não houver, a transição 'VERIF_VAL_RAMOS_E_OC_NOS'

verifica a validade dos ramos e das listas de ocupação de nós. Se os nós destino de todos os ramos e se os nós descriminados nas listas de ocupação de nós existirem todos, a transição 'COMPLETA_BLOQUEIOS' é disparada, sendo garantido que a lista de bloqueios de cada ramo inclui todos os nós da lista de ocupação de nós do nó destino e o próprio nó destino.

A transição 'LE_POS_REL_MESAS' efectua a leitura do ficheiro de configuração, que contém a posição dos pontos de carga/descarga relativamente aos respectivos nós do *layout*. Se a leitura for bem sucedida, ou seja, se for possível conhecer todas as posições, a transição 'TERMINA_CONFIG_SGAGV' é disparada, depositando um *token* em 'FIM_CONFIG_SGAGV'. O anexo F.4 contém um exemplo de um ficheiro de configuração de pontos de carga/descarga. A sintaxe das suas linhas é a que está definida na Tabela 7.14 da página 115.

8.3.2 Sub-rede 'cria_lista_AGVs' do SGAGV

A sub-rede 'cria_lista_AGVs' está representada na Figura 8.21. (ver informação detalhada no anexo C.3.14).

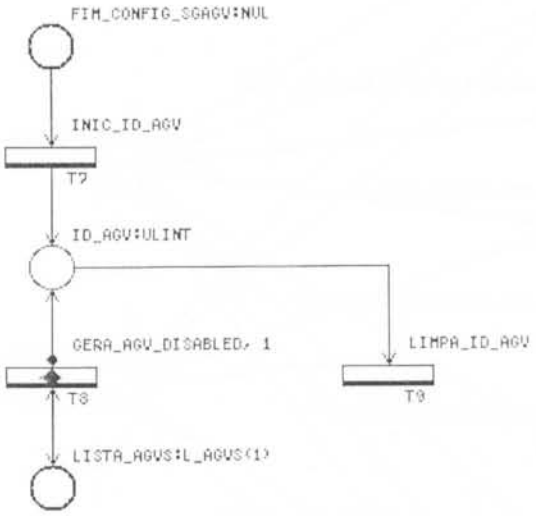


Figura 8.21 – Sub-rede 'cria_lista_AGVs' do SGAGV.

Após a deposição de um *token* na posição FIM_CONFIG_SGAGV, resultante do fim do processo de configuração do SGAGV (sub-rede 'leitura_fich_config'), esta sub-rede insere elementos na lista ligada de AGVs (cuja cabeça é apontada pelo *token* da posição 'LISTA_AGVS') em número igual ao número máximo de AGVs previstos para o sistema, que é configurável através do parâmetro 'numero_agvs' (ver parâmetros do SGAGV no anexo C.3.4). Os AGVs são inicializados no estado DISABLED.

A transição 'INIC_ID_AGV' deposita um *token* em 'ID_AGV', contendo o identificador do primeiro AGV a gerar. A transição 'GERA_AGV_DISABLED' gera um AGV, de acordo com o identificador existente no *token* consumido de 'ID_AGV', e deposita aí um novo *token* com o identificador do AGV seguinte (igual ao anterior incrementado de um).

Quando todos os AGVs já tiverem sido gerados, a transição 'GERA_AGV'_DISABLED' não fica sensibilizada, permitindo o disparo da transição 'LIMPA_ID_AGV', que consome o *token* existente em 'ID_AGV'.

8.3.3 Sub-rede 'proc_mens_receb_SGM' do SGAGV

A sub-rede 'proc_mens_receb_SGM' está representada na Figura 8.22. Esta modeliza o processamento efectuado pelo SGAGV na sequência da recepção de mensagens provenientes do SGM.

É constituída por duas sub-redes:

- proc_mens_relativas_a_AGVs e
- proc_mens_relativas_a_Tarefas.

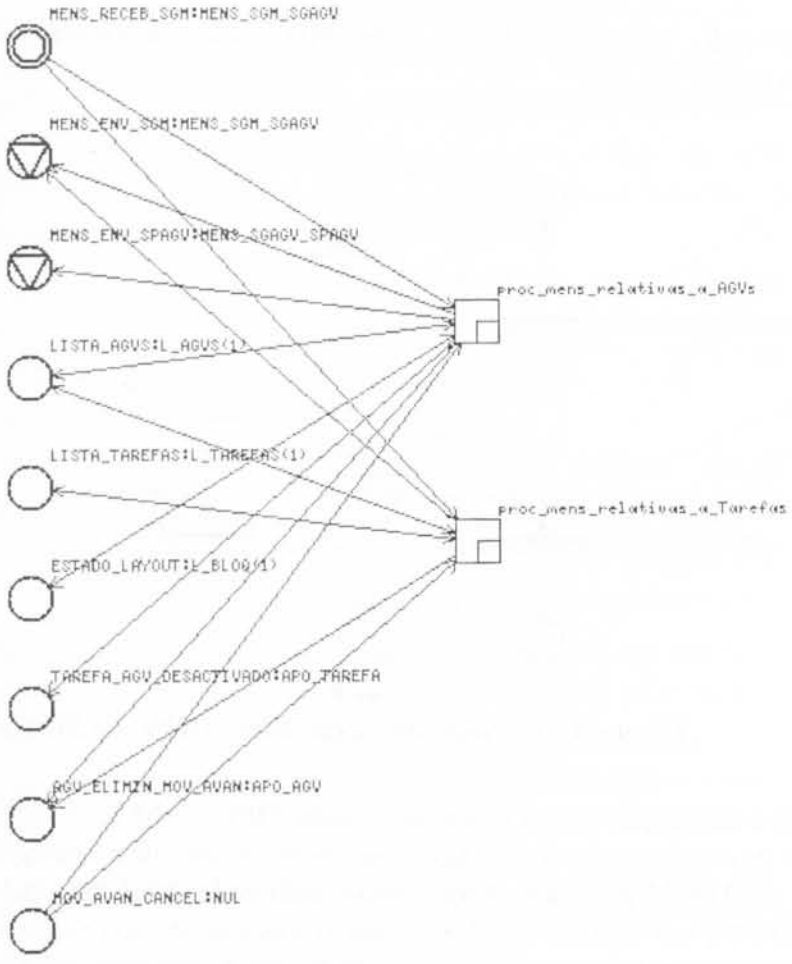


Figura 8.22 – Sub-rede 'proc_mens_receb_SGM' do SGAGV.

8.3.3.1 Sub-rede 'proc_mens_relativas_a_AGVs' do SGAGV

A sub-rede 'proc_mens_relativas_a_AGVs' está representada na Figura 8.23 (ver informação detalhada no anexo C.3.23). Esta modeliza o processamento efectuado pelo

SGAGV na sequência da recepção de mensagens respeitantes à activação, desactivação e mudança do estado de alocação dos AGVs.

A transição 'REC_MENS_ENABLE_AGV' é disparada quando o SGAGV recebe do SGM uma mensagem do tipo 'ENABLE_AGV'.

Na sequência de uma mensagem desse tipo podem ocorrer três situações diferentes, previstas pelas transições seguintes:

- 'NOTIF_ENABLE_AGV_DISABLED' – colocação de um AGV DISABLED no estado ENABLED;
- 'MUDA_EST_ALOC_AGV' – mudança do estado de alocação de um AGV;
- 'NOTIF_ERRO_ENABLE_AGV' – notificação do SGM sobre um erro na activação de um AGV que não exista.

A transição 'REC_MENS_DISABLE_AGV' é disparada quando o SGAGV recebe do SGM uma mensagem do tipo 'DISABLE_AGV'. Esta deposita um *token* em 'AGV_ELIMIN_MOV_AVAN', que dá início à eliminação dos movimentos em avanço do AGV em causa (se existirem). Findo este processo, é disparada a transição 'NOTIF_SPAGV_DISABLE_AGV', que notifica o SPAGV da passagem do AGV ao estado DISABLED.

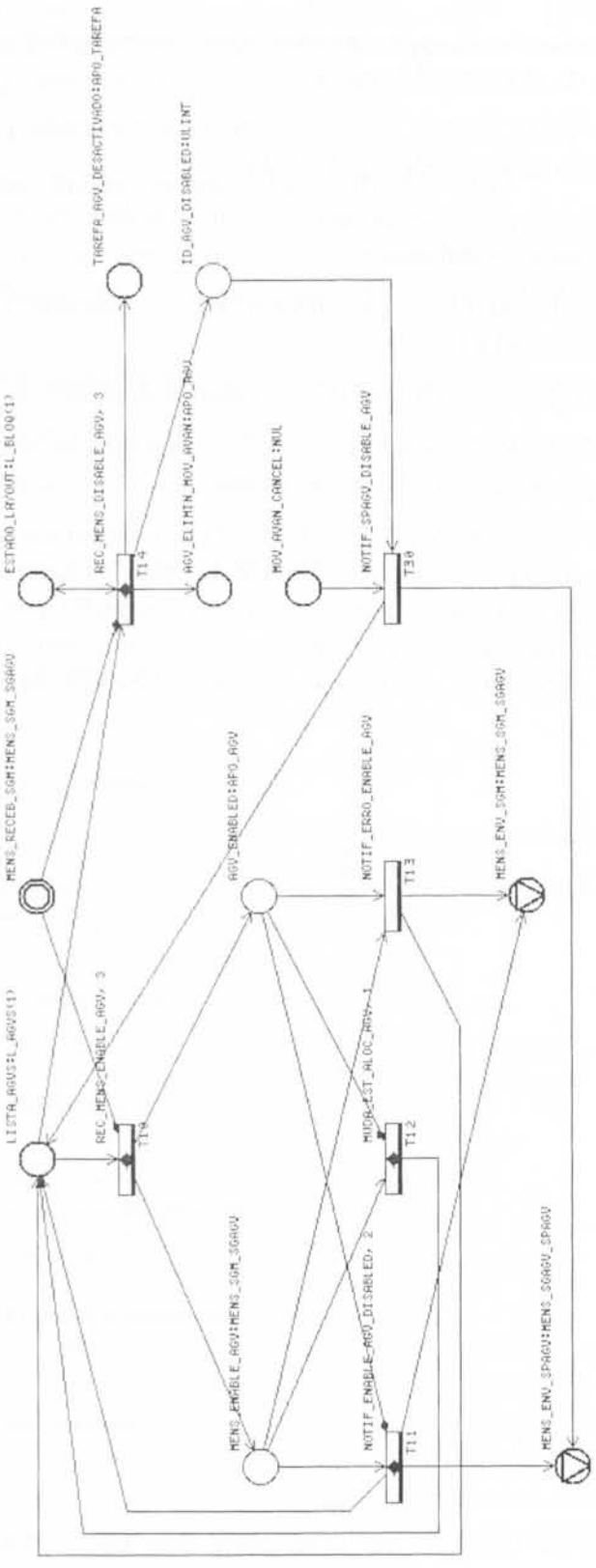


Figura 8.23 – Sub-rede 'proc_mens_relativas_a_AGVs' do SGAGV.

8.3.3.2 Sub-rede 'proc_mens_relativas_a_Tarefas' do SGAGV

A sub-rede 'proc_mens_relativas_a_Tarefas' está representada na Figura 8.24. Esta modeliza o processamento efectuado pelo SGAGV na sequência da recepção de mensagens provenientes do SGM, que digam respeito a tarefas.

É constituída por duas sub-redes:

- proc_novas_tarefas e
- proc_cancel_tarefas.

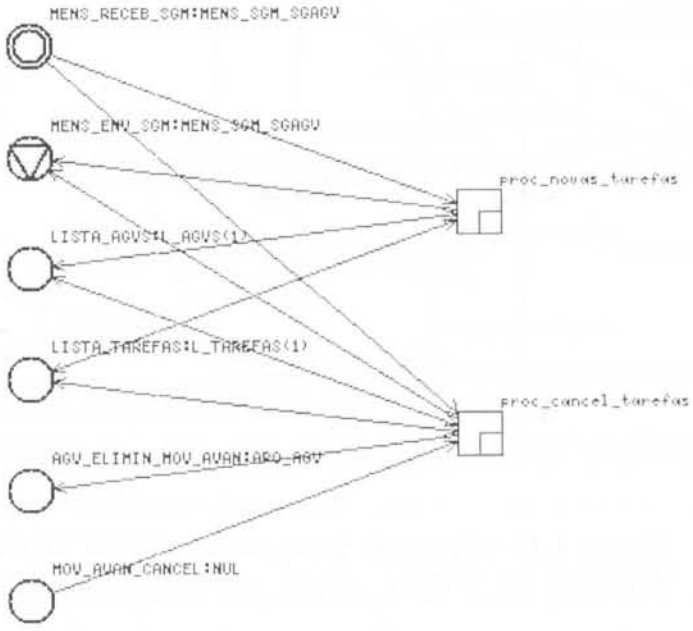


Figura 8.24 – Sub-rede 'proc_mens_relativas_a_Tarefas' do SGAGV.

8.3.3.2.1 Sub-rede 'proc_novas_tarefas' do SGAGV

A sub-rede 'proc_novas_tarefas' está representada na Figura 8.25 (ver informação detalhada no anexo C.3.24). Esta modeliza o processamento efectuado pelo SGAGV, aquando da recepção de mensagens provenientes do SGM, relativas ao lançamento de novas tarefas.

A transição 'VERIFICA_NOS_NOVA_TAREFA' é disparada por qualquer mensagem que diga respeito a uma nova tarefa, efectuando a verificação da validade dos nós associados à tarefa. A posição 'NOVA_TAREFA' recebe um *token* com a nova tarefa lançada e a posição 'ERRO_NOS' recebe um *token* com o resultado do teste de validade dos nós.

Se a tarefa for do tipo 'NORMAL' ou 'T_CARGA' e se o teste de validade dos nós tiver um resultado favorável, a transição 'INS_LISTA_NOVA_TAR_N_ESCAL' é disparada, inserindo a nova tarefa na lista de tarefas pendentes.

Se houver incoerência nos nós da tarefa, a transição 'NOTIF_ERRO_NOS_TAR' é disparada, notificando o SGM de tal facto.

Se o teste dos nós for favorável e se a tarefa tiver sido escalonada pelo SGM, a transição 'VERIFICA_FASIBIL_TAR_ESC_SGM' verifica a fazibilidade da tarefa, ou seja, se o AGV a ela afectado se encontra disponível para a executar. Se estiver disponível, a transição 'INICIA_EXEC_TAR_ESC_SGM' é disparada, inserindo a nova tarefa na lista de tarefas pendentes e dando início à execução da mesma. Se não for possível executá-la, a transição 'NOTIF_ERRO_FASIBIL_TAR' é disparada, notificando o SGM da não fazibilidade da tarefa.

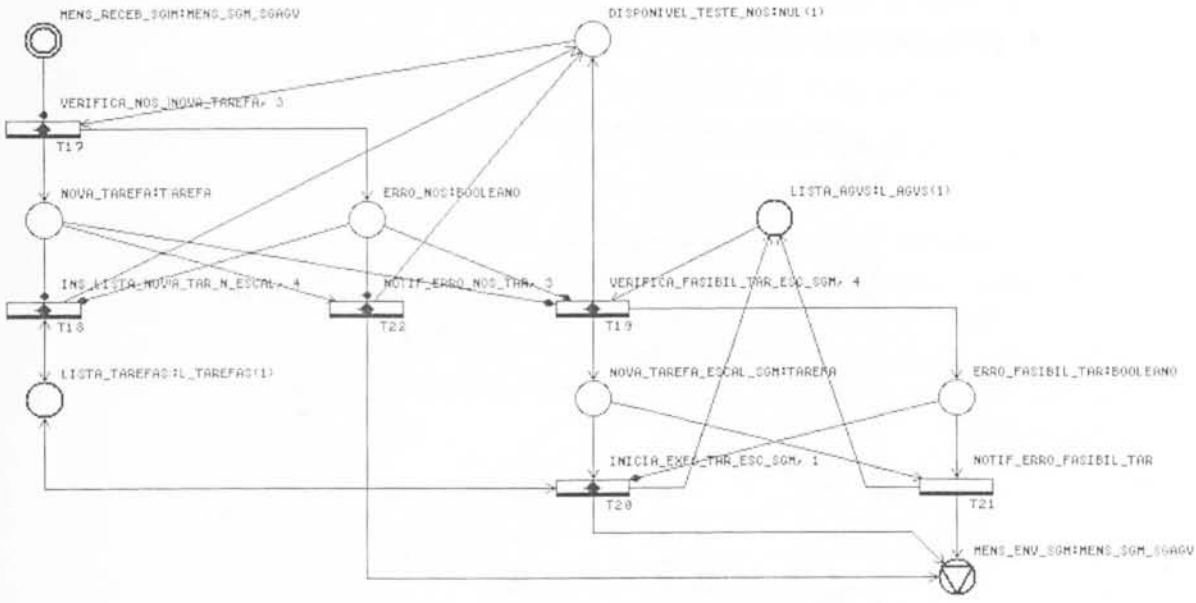


Figura 8.25 – Sub-rede 'proc_novas_tarefas' do SGAGV.

8.3.3.2.2 Sub-rede 'proc_cancel_tarefas'

A sub-rede 'proc_cancel_tarefas' está representada na Figura 8.26 (ver informação detalhada no anexo C.3.21). Esta modeliza o processamento desencadeado pela recepção de mensagens de cancelamento de tarefas.

Quando o SGAGV recebe do SGM um *token* com uma mensagem do tipo 'CANCELA_TAREFA', a transição 'INICIA_CANCEL_TAREFAS' é disparada. Esta deposita um *token* em LISTA_TAR_CANCEL com um apontador para as tarefas a cancelar, um *token* em 'ID_MENS_CANCEL' com o identificador da mensagem de cancelamento de tarefas recebida, e um *token* em 'NUM_TAR_CANCEL' com o número de tarefas a cancelar.

As transições 'CANCELA_TAR_FILA' e 'CANCELA_TAR_EXEC' efectuam o cancelamento de tarefas no estado FILA e de tarefas em execução, respectivamente. No segundo caso é necessário proceder à eliminação de movimentos que tenham sido enviados ao SPAGV em avanço.

Finalmente, a transição 'NOTIF_RESULT_PROC_CANCEL' notifica o SGM sobre o resultado do processo de cancelamento de tarefas. Se o *token* em 'NUM_TAR_CANCEL' tiver um valor maior do que zero, então o cancelamento foi bem sucedido, caso contrário foi mal sucedido.

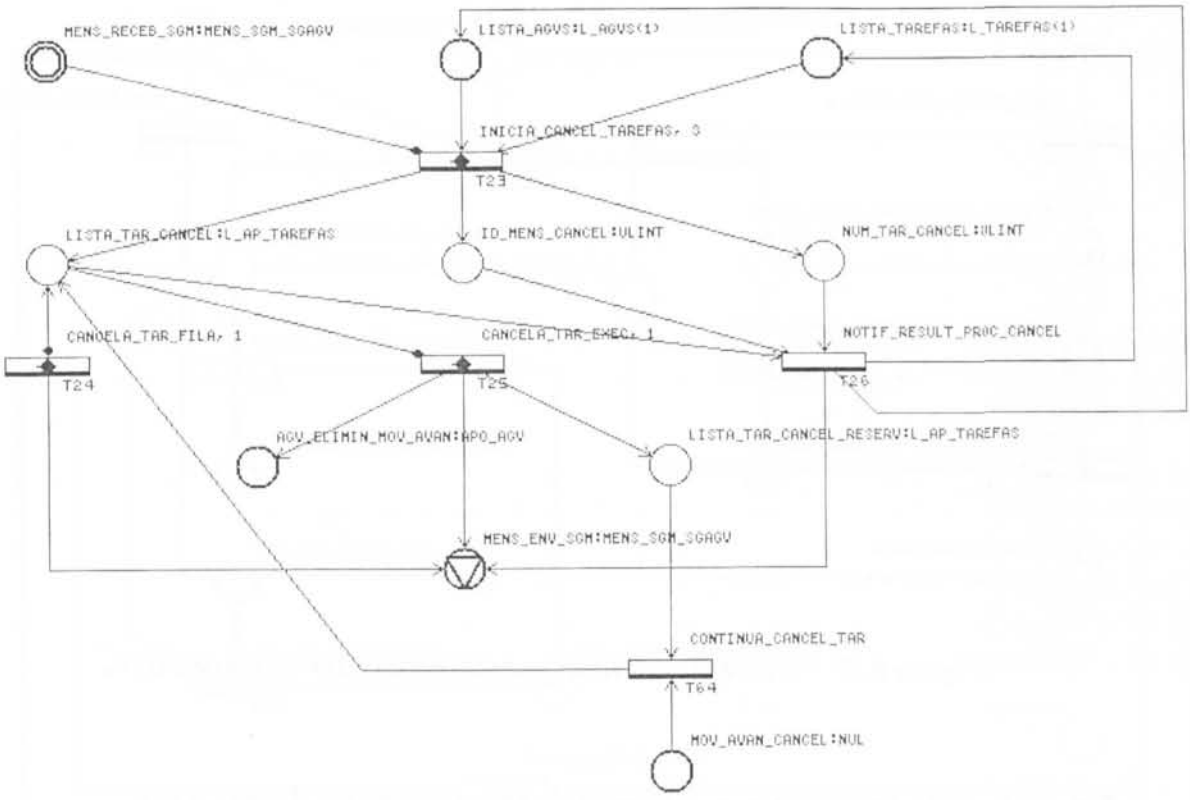


Figura 8.26 – Sub-rede 'proc_cancel_tarefas' do SGAGV.

8.3.4 Sub-rede 'cancela_movim_avan_AGVs' do SGAGV

A sub-rede 'cancela_movim_avan_AGVs' está representada na Figura 8.27 (ver informação detalhada no anexo C.3.13). Esta modeliza o cancelamento de movimentos elementares enviados ao SPAGV em avanço.

Se o AGV apontado pelo *token* da posição 'AGV_ELIMIN_MOV_AVAN' não tiver qualquer movimento em avanço, a transição 'CONF_CANCEL_MOV_AVAN' é disparada, depositando um *token* em 'MOV_AVAN_CANCEL'.

Se o AGV tiver movimentos em avanço, é disparada a transição 'ENVIA_CANCEL_MOV_AVAN', sendo enviada ao SPAGV uma mensagem do tipo 'CANCELA_MOV_AVAN'. Em resposta ao envio desta mensagem, o SPAGV envia uma mensagem do tipo 'MOV_AVAN_CANCEL', a confirmar o cancelamento pretendido, que provoca o disparo da transição 'REC_CONF_CANCEL_MOV_AVAN'. Esta deposita um *token* em 'MOV_AVAN_CANCEL'.

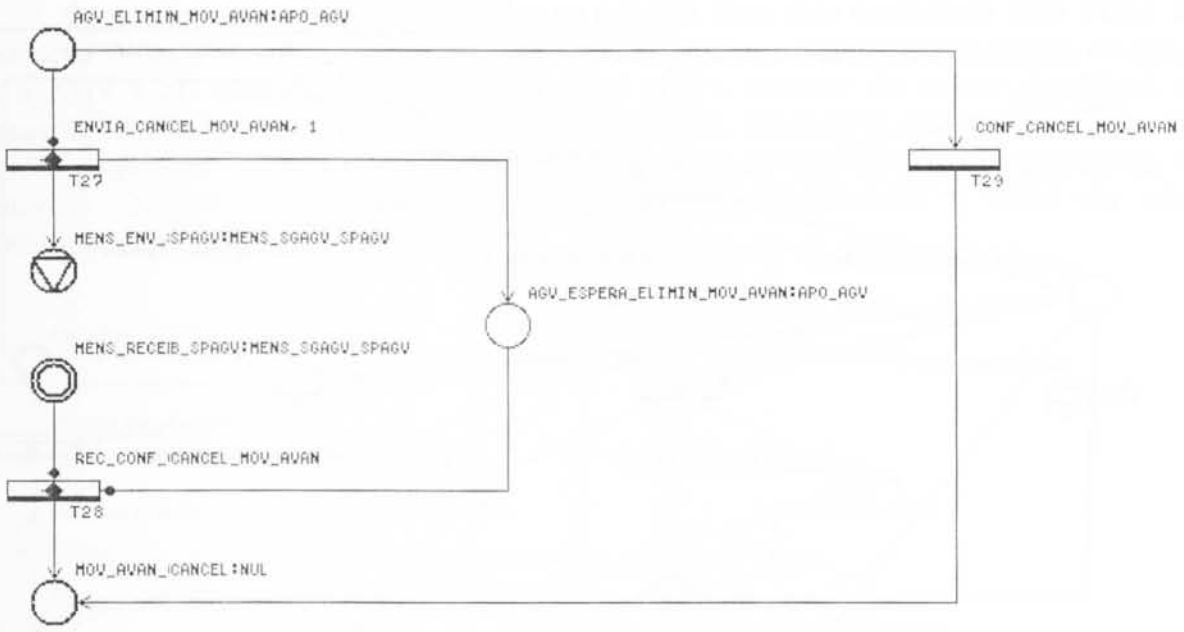


Figura 8.27 – Sub-rede 'cancela_movim_avan_AGVs' do SGAGV.

8.3.5 Sub-rede 'escalona_e_despacha_tar_normais' do SGAGV

A sub-rede 'escalona_e_despacha_tar_normais' está representada na Figura 8.28 (ver informação detalhada no anexo C.3.15). Esta modeliza o escalonamento de tarefas realizado pelo SGAGV.

Quando há pelo menos um AGV disponível e uma tarefa pendente em fila de espera, a transição 'ESCAL_DESP_TAR_NORMAIS' é disparada. Esta desencadeia a chamada à função 'escalon_despacho()' (ver sua implementação no anexo A.2.11), que realiza a afectação dos AGVs às tarefas, seguindo o critério definido pelos parâmetros 'regra_ADCT' e 'regra_ADV'. Para cada combinação possível destes parâmetros é executado um algoritmo de escalonamento diferente, para permitir a comparação de várias estratégias de escalonamento.

No fim da execução do algoritmo, a transição deposita um *token* em 'AGVS_REAFECTADOS', contendo um apontador para a cabeça de uma lista ligada de AGVs reafectados, e um *token* em 'TAR_MUD_ESTADO', contendo um apontador para a cabeça de uma lista ligada de tarefas que mudaram de estado.

Se a lista de AGVs reafectados não for vazia, é disparada a transição 'CANCEL_MOV_AVAN_AGV_REAF' e, logo depois, a transição 'ACT_AGVS_REFECTADOS', por cada AGV aí existente, cancelando-se os movimentos elementares enviados em avanço para esses AGVs.

Por cada tarefa que tenha mudado de estado, é disparada a transição 'NOTIF_SGM_MUD_EST_TAR', que notifica o SGM da mudança de estado da tarefa, através do envio de uma mensagem do tipo 'ESTADO_TAREFA'.

Quando a lista de AGVs reafectados e a lista de tarefas, que mudaram de estado, ficarem ambas vazias, a transição 'FINALIZA_ESCAL_DESP' é disparada, terminando o processo de escalonamento.

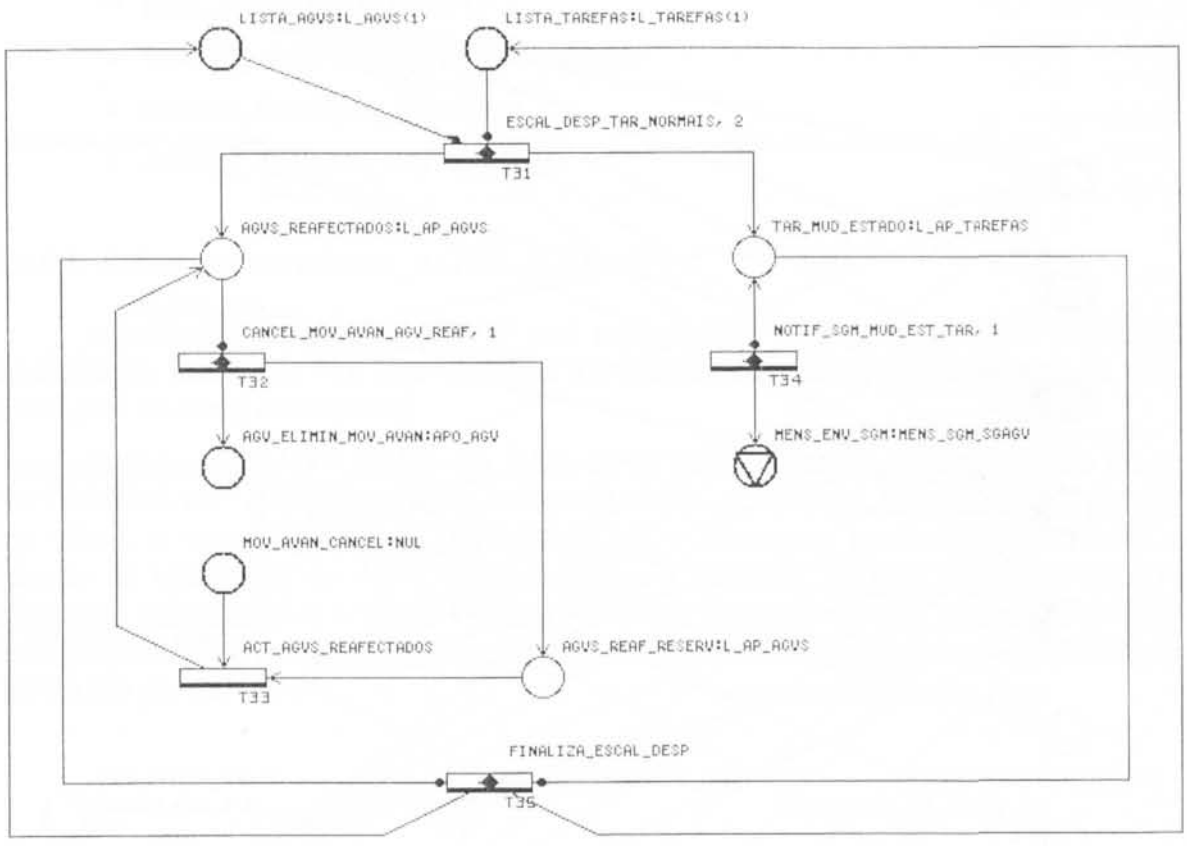


Figura 8.28 – Sub-rede 'escalona_e_despacha_tar_normais' do SGAGV.

8.3.6 Sub-rede 'gere_AGVs' do SGAGV

A sub-rede 'gere_AGVs' está representada na Figura 8.29. Esta modeliza a gestão dos AGVs, no que diz respeito ao acompanhamento do seu estado e à gestão da execução de movimentos elementares (gestão de tráfego).

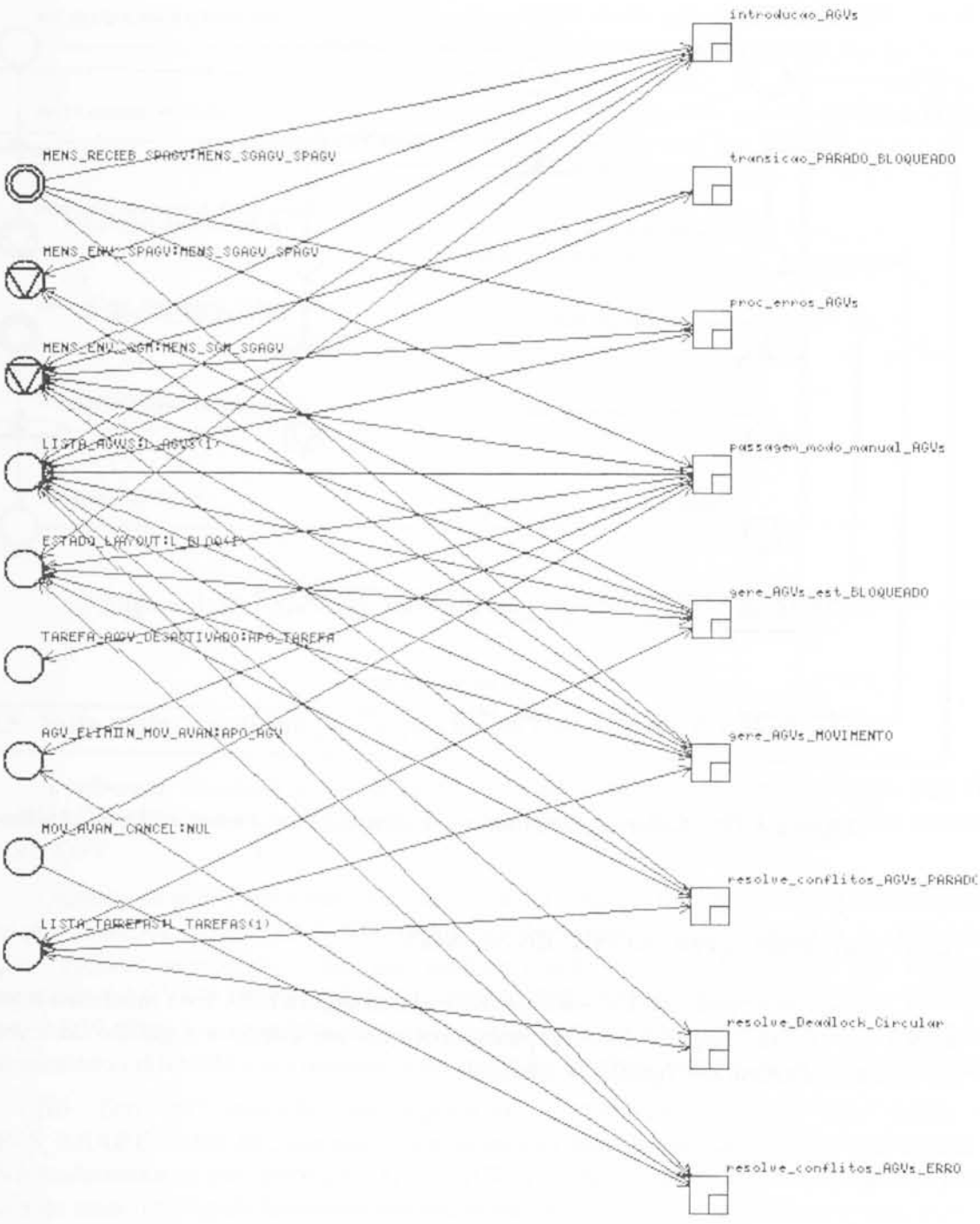


Figura 8.29- Sub-rede 'gere_AGVs' do SGAGV.

É constituída pelas seguintes sub-redes:

- introducao_AGVs,
- transicao_PARADO_BLOQUEADO,
- proc_erros_AGVs,
- passagem_modo_manual_AGVs,
- gere_AGVs_est_BLOQUEADO,

- gere_AGVs_MOVIMENTO,
- resolve_conflitos_AGVs_PARADOS,
- resolve_Deadlock_Circular e
- resolve_conflitos_AGVs_ERRO.

8.3.6.1 Sub-rede 'introducao_AGVS' do SGAGV

A sub-rede 'introducao_AGVS' está representada na Figura 8.30 (ver informação detalhada no anexo C.3.18). Esta modeliza o processamento inerente à introdução de AGVs (passagem ao modo automático).

Quando o SGAGV recebe um *token* do SPAGV, contendo uma mensagem do tipo 'INTRODUCAO', a transição 'REC_INTRODUCAO' é disparada. Se a introdução do AGV for viável, a transição 'CONF_INTROD_AGV' é disparada, notificando o SPAGV do sucesso da introdução do AGV. Caso contrário, a transição 'NOTIF_ERRO_INTROD' é disparada, informando-se o SPAGV sobre o erro que tornou inviável a introdução do AGV. Em qualquer destes casos, a confirmação é feita através do envio de uma mensagem do tipo 'ESTADO_INTR'.

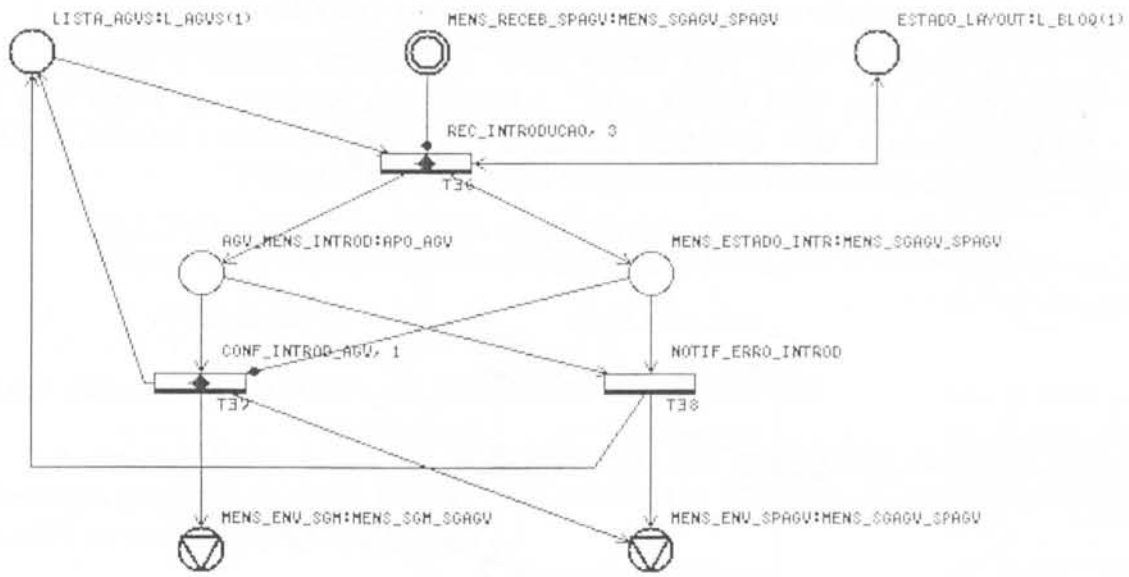


Figura 8.30 – Sub-rede 'introducao_AGVS' do SGAGV.

8.3.6.2 Sub-rede 'transicao_PARADO_BLOQUEADO' do SGAGV

A sub-rede 'transicao_PARADO_BLOQUEADO' está representada na Figura 8.31. Esta contém a transição 'AGV_PARADO_BLOQUEADO', que é disparada sempre que um AGV no estado PARADO for afectado a uma tarefa. Nesse caso, o AGV passa ao estado BLOQUEADO, ainda que seja momentaneamente (durante um intervalo de tempo nulo). Se o AGV puder prosseguir com o movimento, sairá desse estado no mesmo instante de simulação. O código C associado a esta transição está representado no anexo C.3.29.

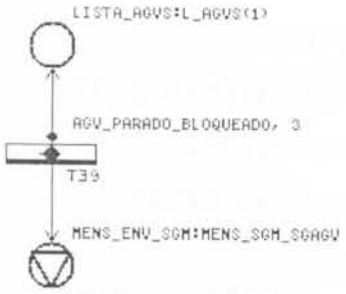


Figura 8.31 – Sub-rede 'transicao_PARADO_BLOQUEADO' do SGAGV.

8.3.6.3 Sub-rede 'proc_erros_AGVs' do SGAGV

A sub-rede 'proc_erros_AGVs' está representada na Figura 8.32. Esta modeliza o processamento do SGAGV, decorrente do tratamento de mensagens relativas à ocorrência de erros internos nos AGVs (avarias).

A transição 'REC_ERRO_AGV' é disparada quando o SPAGV envia uma mensagem do tipo 'ERRO_AGV'. Nesse caso, o SGAGV actualiza o estado do AGV e notifica o SGM desse facto, através do envio de uma mensagem do tipo 'ESTADO_AGV'.

A transição 'REC_FIM_ERRO_AGV' é disparada quando o SPAGV envia uma mensagem do tipo 'FIM_ERRO_AGV', a informar da resolução da avaria num AGV. O SGAGV coloca o AGV no estado anterior à ocorrência da avaria e informa o SGM desse facto, através do envio de uma mensagem do tipo 'ESTADO_AGV'.

O código C associado a estas transições é apresentado no anexo C.3.22.

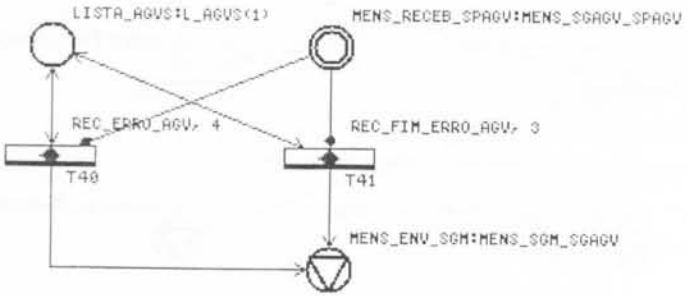


Figura 8.32 – Sub-rede 'proc_erros_AGVs' do SGAGV.

8.3.6.4 Sub-rede 'passagem_modo_manual_AGVs' do SGAGV

A sub-rede 'passagem_modo_manual_AGVs' está representada na Figura 8.33 (ver informação detalhada no anexo C.3.20). Esta modeliza o processamento do SGAGV, decorrente da passagem ao modo manual dos AGVs.

Quando o SPAGV envia uma mensagem do tipo 'AGV_MANUAL', a transição 'REC_AGV_MANUAL' é disparada, colocando-se o AGV em causa no estado ENABLED.

A posição 'TAREFA_AGV_DESACTIVADO' recebe um *token* com a tarefa afectada ao AGV em causa (o apontador aponta para NULL se o AGV não estiver a executar qualquer tarefa). Para além disso, a colocação de um *token* em 'AGV_ELIMIN_MOV_AVAN' desencadeia o cancelamento dos movimentos elementares do AGV, que tenham sido enviados em avanço para o SPAGV. Finalizado o cancelamento desses movimentos, a transição 'NOTIF_AGV_ENABLED' dispara, sendo o SGM informado sobre a passagem do AGV ao estado ENABLED, através do envio de uma mensagem do tipo 'ESTADO_AGV'.

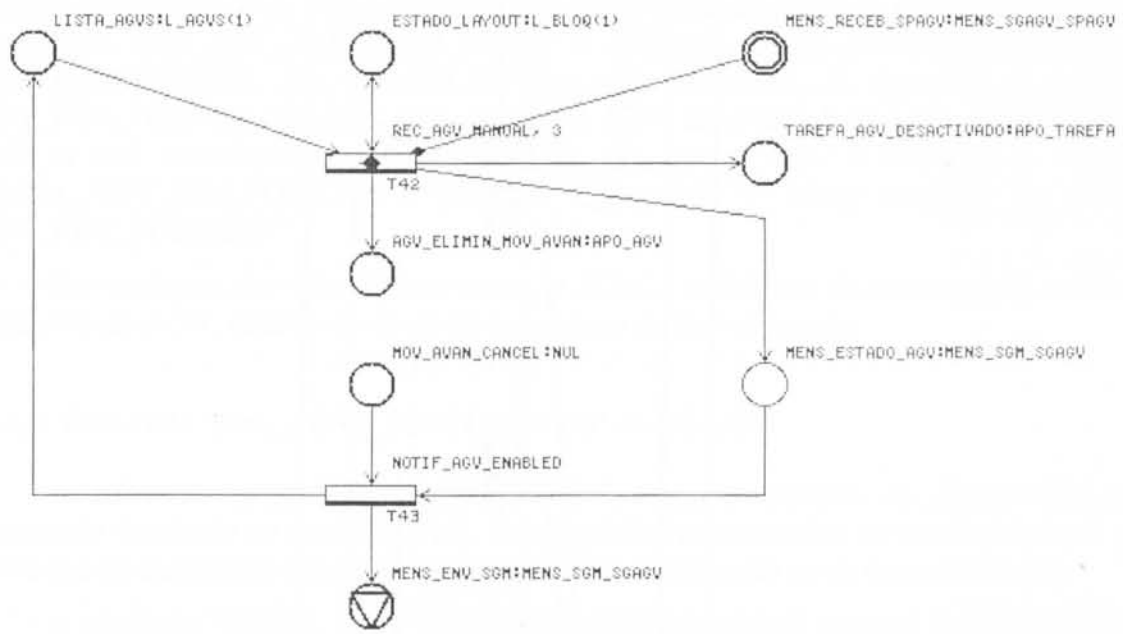


Figura 8.33 – Sub-rede 'passagem_modo_manual_AGVs' do SGAGV.

8.3.6.5 Sub-rede 'gere_AGVs_est_BLOQUEADO' do SGAGV

A sub-rede 'gere_AGVs_est_BLOQUEADO' está representada na Figura 8.34 (ver informação detalhada no anexo C.3.17). Esta modeliza as evoluções de estado possíveis para um AGV no estado BLOQUEADO.

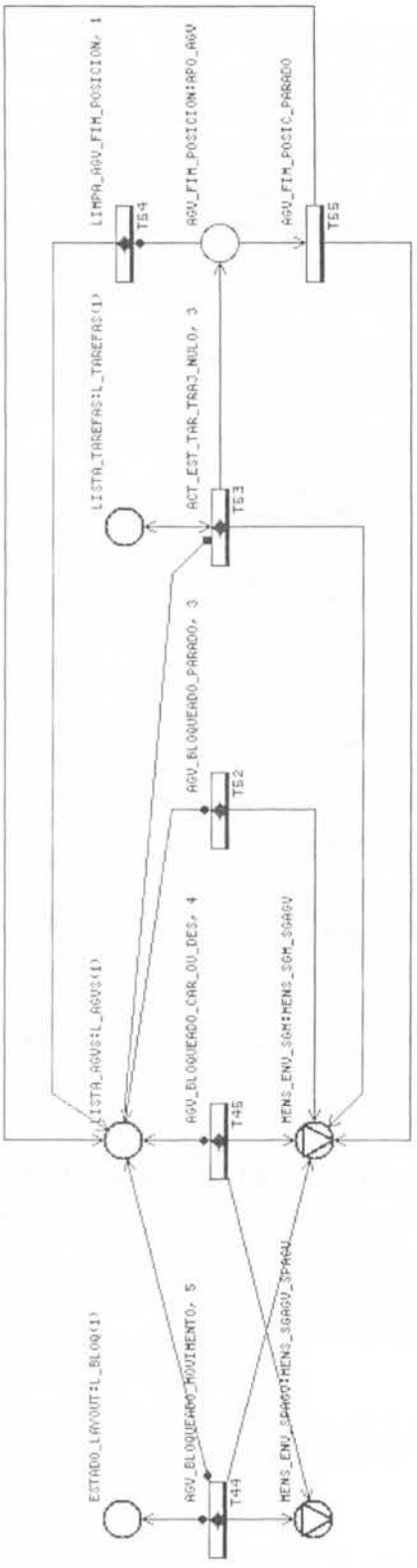


Figura 8.34 – Sub-rede 'gere_AGVs_est_BLOQUEADO' do SGAGV.

Se um AGV tiver um movimento de translacção para executar, e se o estado do *layout* o permitir (se conseguir reservar todos os nós necessários), a transição 'AGV_BLOQUEADO_MOVIMENTO' é disparada, colocando o AGV no estado EX_MOVIM e enviando ao SPAGV a menagem do tipo 'MOVIMENTO' correspondente.

Se um AGV tiver um movimento de carga ou de descarga para executar, ou seja, se a tarefa à qual se encontre afectado estiver no estado PASSO_2 ou PASSO_4, respectivamente, a transição 'AGV_BLOQUEADO_CAR_OU_DES' é disparada colocando o AGV no estado respectivo (CARGA_AGV ou DESCARGA_AGV, respectivamente).

Se um AGV tiver terminado a execução de uma tarefa , é disparada a transição 'AGV_BLOQUEADO_PARADO', colocando o AGV no estado PARADO (disponível).

Quando um AGV for afectado a uma tarefa, cujo passo 1 não implique a realização de qualquer movimento (e.g. se o AGV já estiver posicionado no ponto de carga de uma tarefa), a transição 'ACT_EST_TAR_TRAJ_NULO' é disparada, sendo actualizado o estado da tarefa correspondente. Se a tarefa for de posicionamento, é disparada a transição 'AGV_FIM_POSIC_PARADO', para colocar o AGV no estado PARADO, uma vez que a tarefa já está virtualmente executada. Se não for esse o caso, é disparada a transição 'LIMPA_AGV_FIM_POSICION', para ser consumido o *token* existente na posição 'AGV_FIM_POSICION'.

Em qualquer dos três últimos casos, o SGM é notificado da mudança de estado da tarefa e/ou do AGV, através do envio de mensagens de tipo adequado.

8.3.6.6 Sub-rede 'gere_AGVs_MOVIMENTO' do SGAGV

A sub-rede 'gere_AGVs_MOVIMENTO' está representada na Figura 8.35 (ver informação detalhada no anexo C.3.16). Esta modeliza as evoluções de estado possíveis para AGVs que se encontrem a executar movimentos (de translacção ou de carga/descarga).

A transição 'ENVIA_MOV_AVANCO' dispara quando é possível enviar ao SPAGV mais um movimento elementar em avanço, para um AGV no estado EX_MOVIM.

Quando o SPAGV confirma a execução de um movimento, através do envio de uma mensagem do tipo 'FIM_MOVIMENTO', a transição 'REC_FIM_MOVIM' é disparada. Neste caso podem ocorrer três situações distintas:

- AGV_MANTEM_EST_EX_MOVIM – transição disparada quando o AGV, que acaba de terminar a execução de um movimento elementar de translacção, ainda tem mais movimentos pendentes (enviados em avanço), mantendo-se o seu estado de movimentação;
- AGV_SAI_EST_EX_MOVIM – transição disparada quando o AGV, que acaba de terminar a execução de um movimento elementar de translacção, não tem qualquer movimento pendente, passando ao estado BLOQUEADO;
- AGV_TERMINA_OP_CD – transição disparada quando um AGV termina a execução de um movimento de carga ou de descarga, passando ao estado BLOQUEADO.

Nos últimos dois casos, a posição 'TAREFA_MUDOU_ESTADO' recebe um *token* correspondente à tarefa afectada ao AGV, que acaba de mudar de estado, e é notificado o SGM da alteração do estado do AGV. Se o AGV estiver afectado a alguma tarefa, a transição 'NOTIF_SGM_MUD_EST_TAREFA' informa o SGM da alteração do estado da tarefa, caso contrário, a transição 'LIMPA_TAREFA_MUDOU_ESTADO' consome o *token* depositado

[1] Quando um AGV termina a execução de qualquer movimento, este fica no estado BLOQUEADO.

em 'TAREFA_MUDOU_ESTADO'. Este último caso pode ocorrer na sequência da passagem ao modo manual de um AGV, ou da reafectação dinâmica de tarefas.

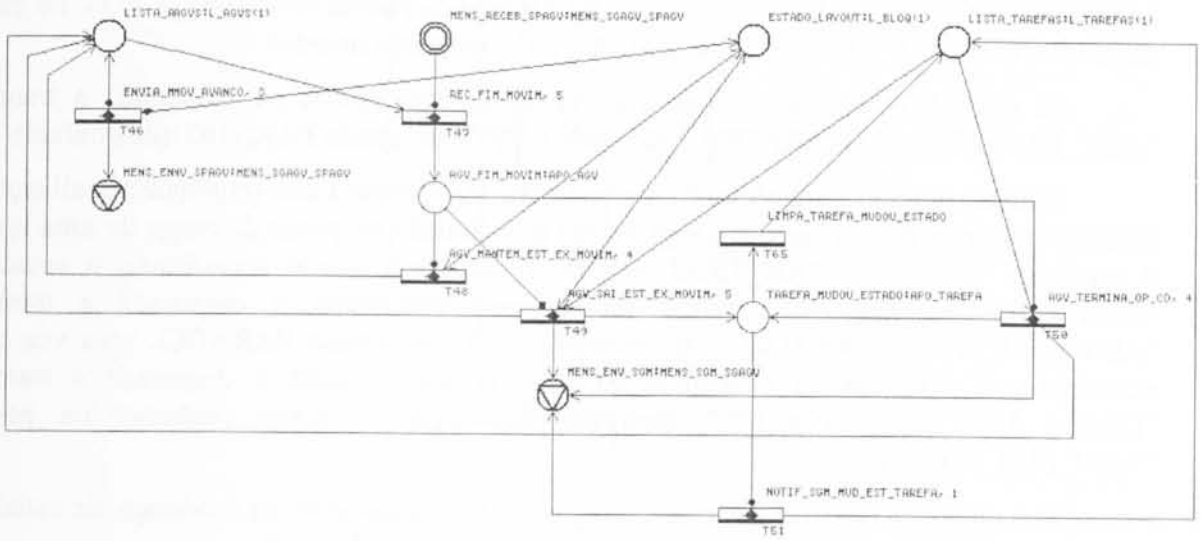


Figura 8.35 – Sub-rede 'gere_AGVs_MOVIMENTO' do SGAGV.

8.3.6.7 Sub-rede 'resolve_conflitos_AGVs_PARADOS' do SGAGV

A sub-rede 'resolve_conflitos_AGVs_PARADOS' está representada na Figura 8.36. Esta modeliza a resolução de conflitos na gestão de tráfego, provocados pelo bloqueamento de AGVs a executarem trajectos, por parte de AGVs no estado PARADO (disponíveis).

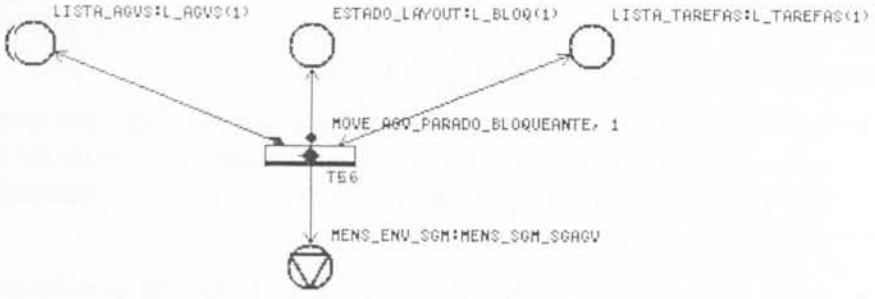


Figura 8.36 – Sub-rede 'resolve_conflitos_AGVs_PARADOS' do SGAGV.

Neste tipo de situações, a transição 'MOVE_AGV_PARADO_BLOQUEANTE' é disparada, sendo chamada a função 'agv_bloq_por_agv_PARADO()' (ver a sua implementação no anexo A.2.13). A função procura movimentar o AGV bloqueante para um nó vizinho, onde não estorve a passagem dos outros AGVs. Dentre os vários nós candidatos, é escolhido aquele que implique o menor custo na movimentação do AGV bloqueante. Para o efeito, o SGAGV lança uma tarefa do tipo 'POSIC_ESPEC', informando o SGM de tal facto, através do envio de uma mensagem do tipo 'T_POSIC_ESPEC'. O código C associado a esta transição é apresentado no anexo C.3.28.

8.3.6.8 Sub-rede 'resolve_Deadlock_Circular' do SGAGV

A sub-rede 'resolve_Deadlock_Circular' está representada na Figura 8.37. Esta modeliza o processamento efectuado pelo SGAGV, relativo à detecção e resolução de *deadlocks* na gestão de tráfego.

O predicado da transição 'RESOLVE_DEADLOCK_CIRCULAR' consiste em detectar a ocorrência de um *deadlock*, através da chamada à função 'existe_deadlock_circular()' (ver a sua implementação no anexo A.2.13). Esta função percorre a lista de AGVs e a lista de bloqueios de nós do *layout*, procurando formar uma lista circular de AGVs no estado BLOQUEADO, com a configuração (AGV_BLOQUEADO, AGV_BLOQUEANTE). Se for detectada uma lista circular, então existe um *deadlock*, e o predicado da transição é avaliado como verdadeiro.

O disparo da transição desencadeia a chamada à função 'resolve_deadlock_circular()' (ver a sua implementação no anexo A.2.13). Esta função procura reencaminhar um dos AGVs intervenientes no *deadlock*, de forma a eliminar o ciclo de espera circular (ver). É calculado um novo trajecto para o AGV, cujo reencaminhamento implique um custo mínimo.

O código C associado a esta transição é apresentado no anexo C.3.26.

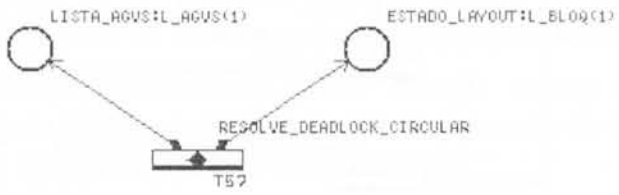


Figura 8.37 – Sub-rede 'resolve_Deadlock_Circular' do SGAGV.

8.3.6.9 Sub-rede 'resolve_conflitos_AGVs_ERRO' do SGAGV

A sub-rede 'resolve_conflitos_AGVs_ERRO' está representada na Figura 8.38 (ver informação detalhada no anexo C.3.27). Esta modeliza a resolução de conflitos na gestão de tráfego, provocados pelo bloqueamento de AGVs a executarem trajectos, por parte de AGVs no estado ERRO.

Neste tipo de situações, a transição 'RES_CONFL_C_AGVS_EST_ERRO' é disparada, sendo chamada a função 'resolve_bloq_por_AGVs_erro()' (ver a sua implementação no anexo A.2.13). A função procura reencaminhar todos os AGVs bloqueados pelo AGV no estado ERRO, ou seja aqueles AGVs cujos trajectos necessitem dos nós actualmente reservados para o AGV no estado ERRO. A função procura calcular um novo trajecto para todos estes AGVs, de forma que estes não necessitem reservar os nós bloqueados pelo AGV bloqueante, para executarem os seus trajectos (pode não ser possível, se não existirem caminhos alternativos).

A transição deposita um *token* na posição 'AGVS_REDIR', que contém um apontador para a cabeça de uma lista de AGVs redireccionados. Para todos os AGVs nessa lista, o disparo das transições 'CANCEL_MOV_AVAN_AGV_REDIR' e 'ACT_AGVS_REDIR' permite eliminar os movimentos enviados ao SPAGV em avanço, para os AGVs em causa. O

processo termina com o disparo da transição 'FINALIZA_RESOL_CONFL_AGVS_ERRO', quando a lista de AGVs ficar vazia.

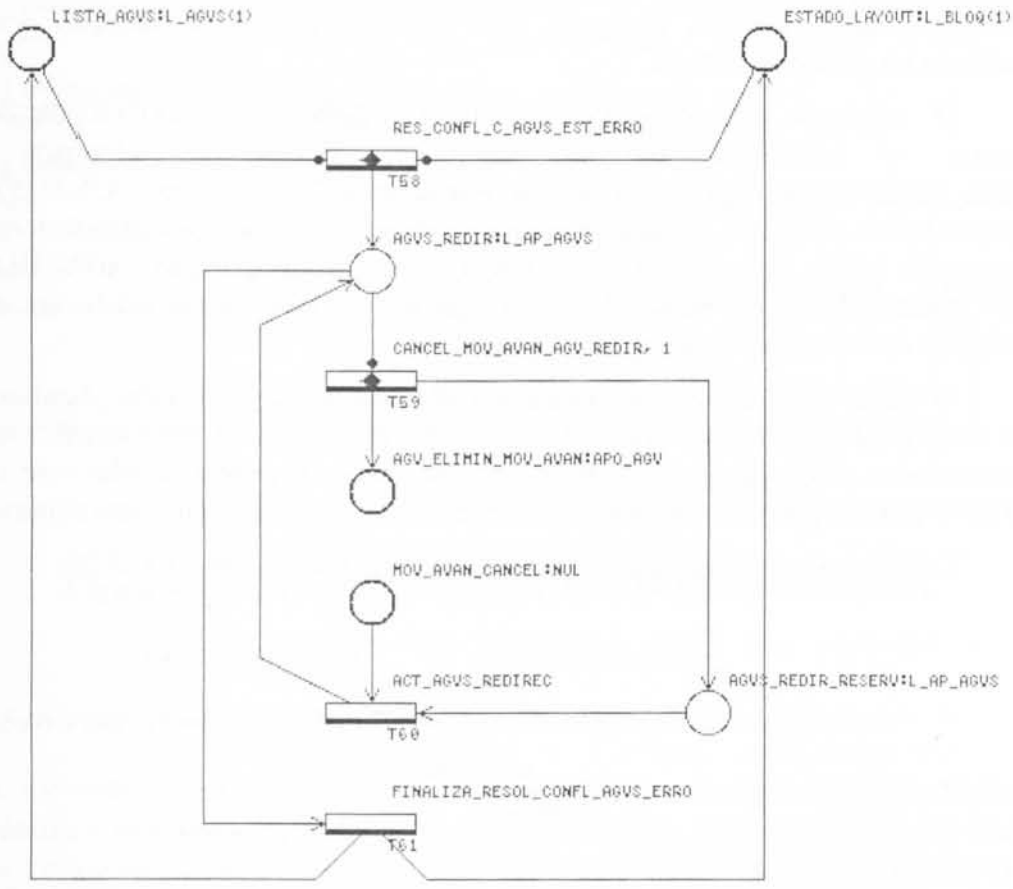


Figura 8.38 – Sub-rede 'resolve_conflitos_AGVs_ERRO' do SGAGV.

8.3.7 Sub-rede 'proc_tarefas_AGVs_desactivados' do SGAGV

A sub-rede 'proc_tarefas_AGVs_desactivados' está representada na Figura 8.39. Esta modeliza o processamento efectuado sobre tarefas, cuja execução seja interrompida pela desactivação do AGV a que se encontrem afectadas.

A transição 'TRATA_TAR_AGV_DESACTIVADO' coloca a tarefa em causa no estado FILA, se for possível retomar a execução da tarefa mais tarde. Isto não é possível se a tarefa for do tipo 'NORMAL' ou 'T_CARGA', e se já tiver sido efectuado o passo 1 da tarefa, ou se a tarefa em causa tiver sido escalonada pelo SGM. Nestes casos, a tarefa é cancelada.

O SGM é notificado da alteração do estado da tarefa ou do seu cancelamento, através do envio de uma mensagem do tipo 'ESTADO_TAREFA' ou 'TAREFA_CANCEL', respectivamente.

Se a tarefa apontada pelo *token* da posição 'TAREFA_AGV_DESACTIVADO' não existir, a transição 'LIMPA_TAREFA_AGV_DESACTIVADO' consome esse *token*.

O código C associado a estas transições é apresentado no anexo C.3.25.

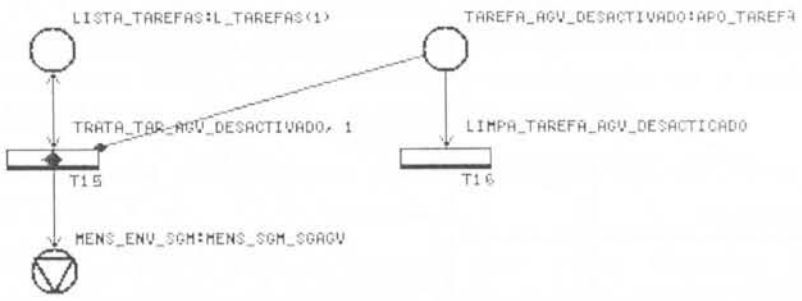


Figura 8.39 – Sub-rede 'proc_tarefas_AGVs_desactivados' do SGAGV.

8.4 Classe SPAGV

A página principal da RP da classe SPAGV é apresentada na Figura 8.40 (ver informação detalhada no anexo C.4).

As posições de entrada definidas são:

- MENS_RECEB_SGAGV:MENS_SGAGV_SPAGV – posição através da qual são recebidos *tokens* do exterior, contendo mensagens enviadas pelo SPAGV;
- FIM_OPERACAO_CARGA:ULINT – posição através da qual o SPAGV é notificado pelo sistema de fabrico do fim da execução de uma operação de carga, no ponto de carga indicado pelo *token* recebido;
- FIM_OPERACAO_DESCARGA:ULINT – posição através da qual o SPAGV é notificado pelo sistema de fabrico do fim da execução de uma operação de descarga, no ponto de descarga indicado pelo *token* recebido.

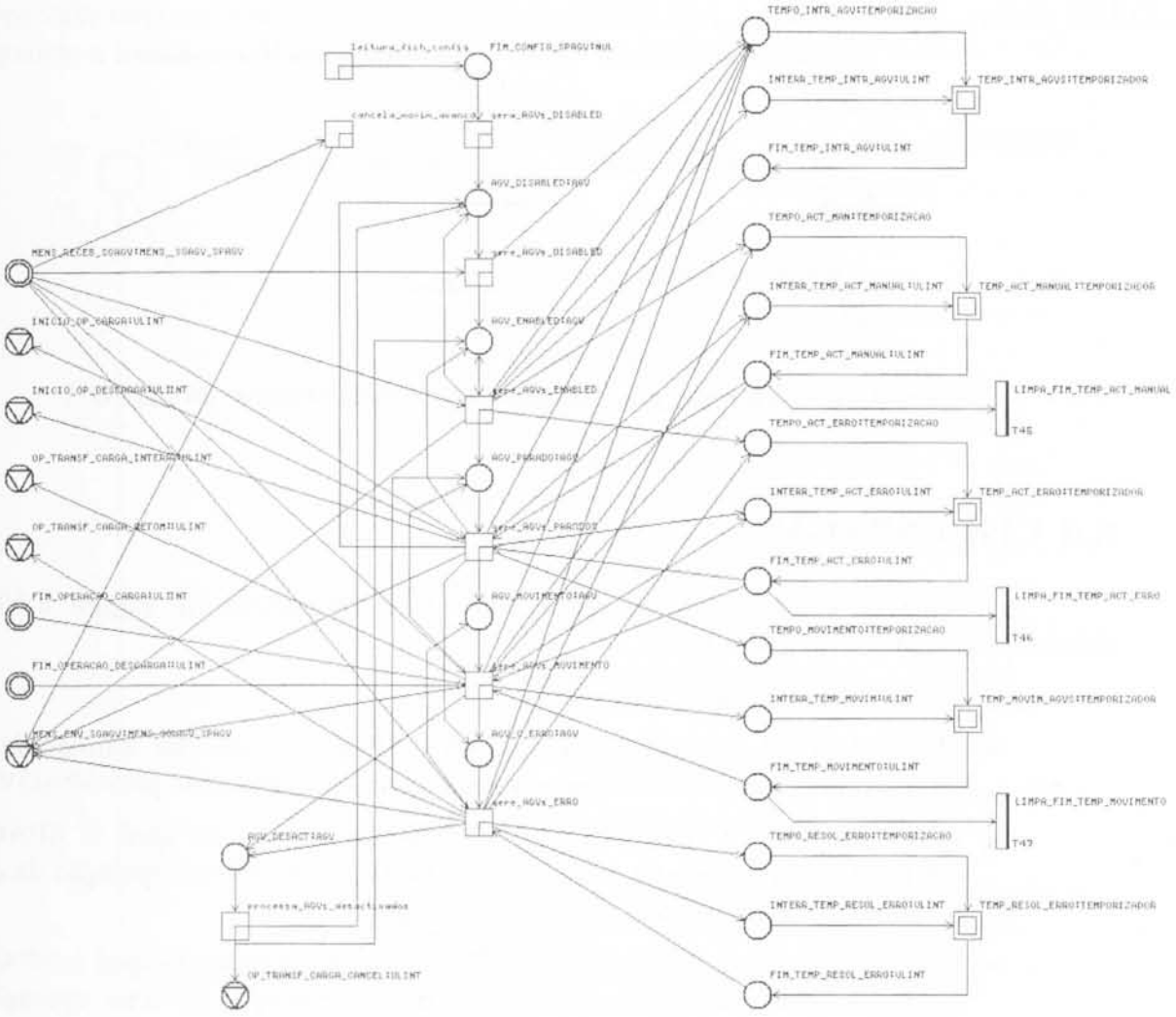


Figura 8.40 – Página principal da RP do SPAGV.

As posições de saída definidas são:

- MENS_ENV_SGAGV:MENS_SGM_SGAGV – posição através da qual são enviados *tokens* ao exterior, contendo mensagens destinadas ao SGAGV;
- INICIO_OP_CARGA:ULNT – posição através da qual o SPAGV informa o sistema de fabrico do início da execução de uma operação de carga, no ponto de carga indicado pelo *token* enviado;
- INICIO_OP_DESCARGA:ULNT – posição através da qual o SPAGV informa o sistema de fabrico do início da execução de uma operação de descarga, no ponto de descarga indicado pelo *token* enviado;
- OP_TRANSF_CARGA_INTERR:ULINT – posição através da qual o sistema de fabrico é notificado da interrupção de uma operação de carga/descarga no ponto indicado pelo *token* enviado, devido a uma avaria no AGV;
- OP_TRANSF_CARGA_RETOM:ULINT – posição através da qual o sistema de fabrico é notificado do reinicio de uma operação de carga/descarga no ponto indicado pelo *token* enviado, após a resolução de uma avaria no AGV;

- OP_TRANSF_CARGA_CANCEL:ULINT – posição através da qual o sistema de fabrico é notificado do cancelamento de uma operação de carga/descarga no ponto indicado pelo *token* enviado, devido à desactivação ou à passagem ao modo manual do AGV envolvido na operação em causa.

O SPAGV instancia cinco objectos da classe TEMPORIZADOR. A sua utilidade é descrita no anexo C.4.12.1.

As transições 'LIMPA_FIM...' existem para evitar situações indesejáveis no funcionamento dos respectivos temporizadores. Estas poderão ocorrer muito pontualmente, quando é depositado um *token* de interrupção da temporização, ao mesmo tempo que esta termina. O código C associado a estas transições é apresentado no anexo C.4.12.2.

A RP do SPAGV é constituída por nove sub-redes:

- leitura_fich_config,
- gera_AGVs_DISABLED,
- cancela_movim_avanco,
- processa_AGVs_desactivados,
- gere_AGVs_DISABLED,
- gere_AGVs_ENABLED,
- gere_AGVs_PARADOS,
- gere_AGVs_MOVIMENTO e
- gere_AGVs_ERRO.

8.4.1 Sub-rede 'leitura_fich_config' do SPAGV

A sub-rede 'leitura_fich_config' está representada na Figura 8.41 (ver informação detalhada no anexo C.4.20).

O *token* da posição 'INICIO_CONFIG_SPAGV', existente na marcação inicial, desencadeia o arranque do SPAGV. A transição 'LE_FICH_LAYOUT' efectua a leitura do ficheiro de configuração com a estrutura do *layout*. O *token* depositado na posição 'LAYOUT' permite aceder durante a simulação ao conteúdo da lista ligada, utilizada para representar a estrutura do *layout*. Para um mesmo SAGVs, o ficheiro de *layout* lido pelo SPAGV é o mesmo que é lido pelo SGAGV.

De seguida, a transição 'LE_FICH_NOS_INTRODUCAO' efectua a leitura do ficheiro de configuração, que contém a probabilidade de ocorrência da introdução de um AGV nos nós do *layout*. O *token* depositado na posição 'NOS_INTR' permite aceder durante a simulação ao conteúdo da lista ligada, utilizada para representar os nós onde podem ser introduzidos AGVs. O ficheiro de configuração lido pelo SPAGV é o mesmo ficheiro que o SGM lê para obter a probabilidade de cada nó ser escolhido como destino de uma tarefa de posicionamento (ver 8.2.1).

A transição 'VERIF_SOMAT_PROB_NOS_INTR' verifica se o somatório das probabilidades é unitário. Se for unitário, a transição 'TERMINA_CONFIG_SPAGV' é disparada, depositando um *token* na posição 'FIM_CONFIG_SPAGV' e pondo termo à configuração do SPAGV.

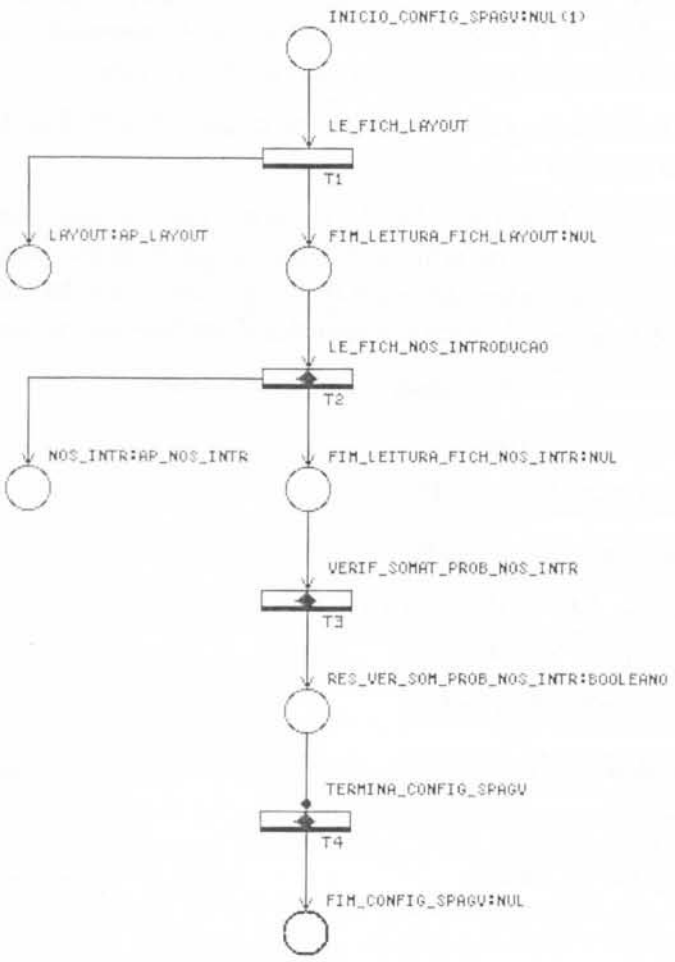


Figura 8.41 – Sub-rede 'leitura_fich_config' do SPAGV.

8.4.2 Sub-rede 'gera_AGVs_DISABLED' do SPAGV

A sub-rede 'gera_AGVs_DISABLED' está representada na Figura 8.42 (ver informação detalhada no anexo C.4.14).

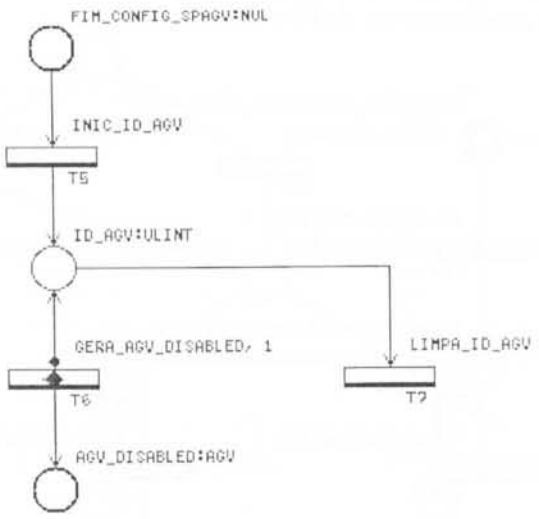


Figura 8.42 – Sub-rede 'gera_AGVs_DISABLED' do SPAGV.

Após a deposição de um *token* na posição FIM_CONFIG_SPAGV, resultante do fim do processo de configuração do SPAGV (sub-rede 'leitura_fich_config'), esta sub-rede deposita *tokens* na posição AGV_DISABLED, em número igual ao número máximo de AGVs previstos para o sistema, que é configurável através do parâmetro 'numero_agvs' (ver parâmetros do SPAGV no anexo C.4.4). Cada *token* depositado representa um AGV no estado DISABLED.

A transição 'INIC_ID_AGV' deposita um *token* em 'ID_AGV', contendo o identificador do primeiro AGV a gerar. A transição 'GERA_AGV_DISABLED' gera um AGV em 'AGV_DISABLED', de acordo com o identificador existente no *token* consumido de 'ID_AGV', e deposita aí um novo *token* com o identificador do AGV seguinte (igual ao anterior incrementado de um).

Quando todos os AGVs já tiverem sido gerados, a transição 'GERA_AGV_DISABLED' não fica sensibilizada, permitindo o disparo da transição 'LIMPA_ID_AGV', que consome o *token* existente em 'ID_AGV'.

8.4.3 Sub-rede 'cancela_movim_avanco' do SPAGV

A sub-rede 'cancela_movim_avanco' está representada na Figura 8.43 (ver informação detalhada no anexo C.4.13). Esta modeliza o cancelamento de movimentos elementares enviados em avanço.

Quando o SPAGV recebe uma mensagem do tipo 'CANCELA_MOV_AVAN', a transição 'REC_CANCEL_MOV_AVAN' é disparada, depositando na posição 'ID_AGV_MOV_AVAN_CANCEL' um *token* com o identificador do AGV em causa.

Enquanto houver mensagens do tipo MOVIMENTO, destinadas a esse AGV, na fila de *tokens* da posição de entrada 'MENS_RECEB_SGAGV', a transição 'CANCELA_MOV_AVANCO' é disparada. Esta vai consumindo esses *tokens*, cancelando portanto os movimentos correspondentes. Quando não houver mais movimentos a cancelar, a transição 'NOTIF_SGAGV_CANCEL_MOV_AVAN' é disparada, sendo enviada ao

SGAGV uma mensagem do tipo 'MOV_AVAN_CANCEL', confirmando o cancelamento de todos os movimentos pendentes do AGV.

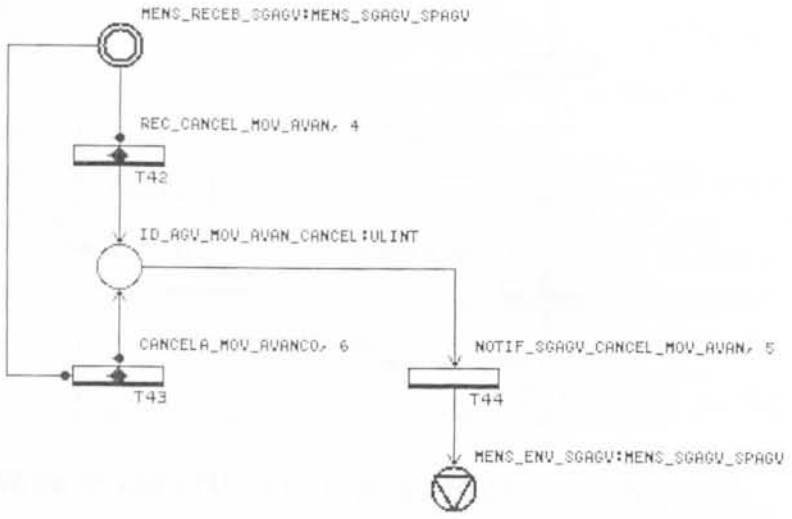


Figura 8.43 – Sub-rede 'cancela_movim_avanco' do SPAGV.

8.4.4 Sub-rede 'processa_AGVs_desactivados' do SPAGV

A sub-rede 'processa_AGVs_desactivados' está representada na Figura 8.44 (ver informação dletalhada no anexo C.4.21). Esta modeliza o processamento efectuado pelo SPAGV, após a desactivação de um AGV.

Se o AGV desactivado estava a efectuar uma operação de carga/descarga, a transição 'CANCELA_OP_TRCAR' é disparada, sendo o sistema de fabrico notificado do cancelamento da operação, através da colocação de um *token* na posição de saída 'OP_TRANSF_CARGA_CANCEL', identificando o ponto do *layout* em causa. Se não for este o caso, a transição 'LIMPA_AGV_DESACT' é disparada.

Consoante a desactivação do AGV implique a sua passagem ao estado ENABLED ou ao estado DISABLED, é disparada a transição 'AGV_DESACT_ENABLED' ou a transição 'AGV_DESACT_DISABLED', respectivamente.

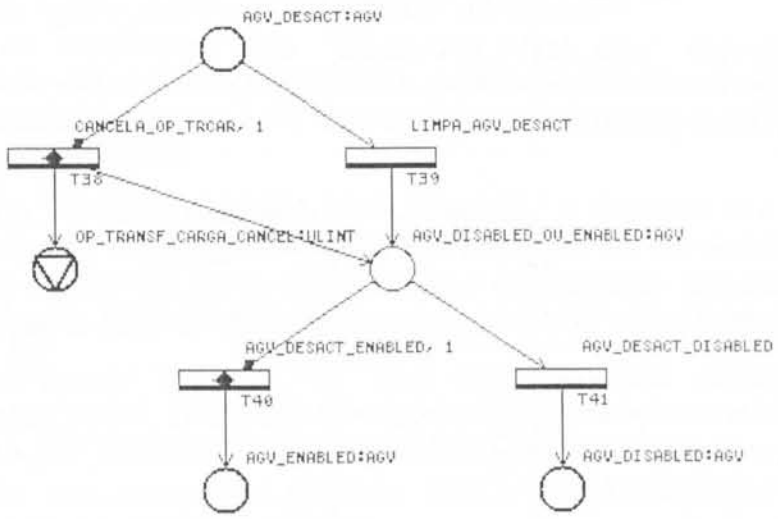


Figura 8.44 – Sub-rede 'processa_AGVs_desactivados' do SPAGV.

8.4.5 Sub-rede 'gere_AGVs_DISABLED' do SPAGV

A sub-rede 'gere_AGVs_DISABLED' está representada na Figura 8.44. Esta modeliza a evolução de estado possível para um AGV no estado DISABLED.

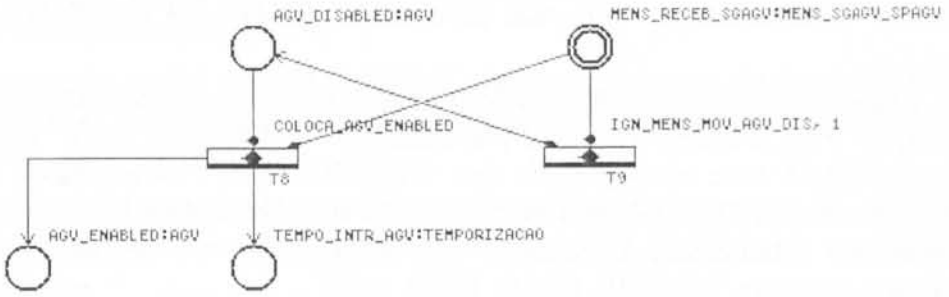


Figura 8.44 – Sub-rede 'gere_AGVs_DISABLED' do SPAGV.

Quando um AGV DISABLED passa ao estado ENABLED, após notificação do SGAGV, a transição 'COLOCA_AGV_ENABLED' actualiza o estado do AGV e deposita o *token* correspondente na posição 'AGV_ENABLED'. Para além disso, é iniciada uma temporização que levará à futura introdução do AGV, através da colocação de um *token* na posição 'TEMPO_INTR_AGV'.

A transição 'IGN_MENS_MOV_AGV_DIS' permite consumir mensagens incoerentes do tipo 'MOVIMENTO', destinadas a AGVs no estado DISABLED.

O código C associado a estas transições é apresentado no anexo C.4.15.

8.4.6 Sub-rede 'gere_AGVs_ENABLED' do SPAGV

A sub-rede 'gere_AGVs_ENABLED' está representada na Figura 8.45 (ver informação detalhada no anexo C.4.16). Esta modeliza as evoluções de estado possíveis para um AGV no estado ENABLED.

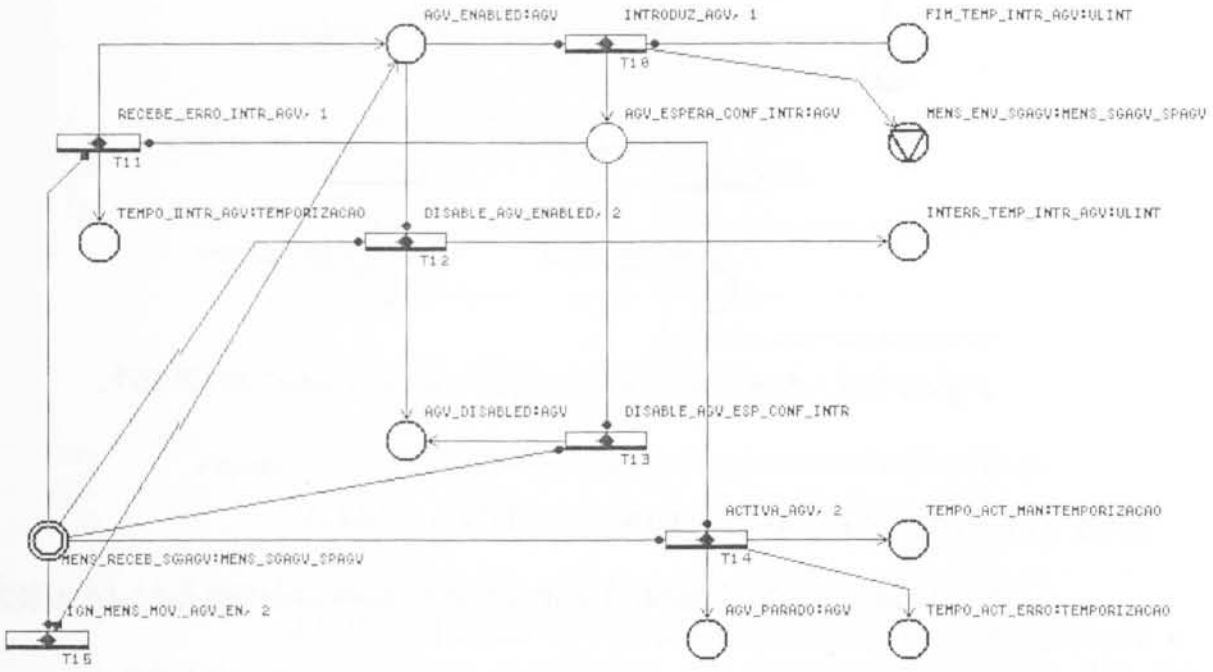


Figura 8.45 – Sub-rede 'gere_AGVs_ENABLED' do SPAGV.

Quando o SPAGV decide introduzir um AGV, ou seja, após a deposição de um *token* na posição 'FIM_TEM_INTR_AGV', a transição 'INTRODUZ_AGV' é disparada. Esta envia ao SGAGV uma mensagem do tipo 'INTRODUCAO'. Na sequência deste evento, o SPAGV recebe do SGAGV uma mensagem do tipo 'ESTADO_INTR'. Se o AGV ficar efectivamente introduzido, a transição 'ACTIVA_AGV' é disparada, caso contrário, é disparada a transição 'RECEBE_ERRO_INTR_AGV'.

A transição 'ACTIVA_AGV' actualiza o estado do AGV, coloca um *token* do AGV na posição 'AGV_PARADO' e inicia duas temporizações para o AGV em causa: uma que poderá simular a passagem do AGV ao modo manual e outra que poderá simular a ocorrência de uma avaria no AGV.

A transição 'DISABLE_AGV_ENABLED' é disparada, se o SPAGV receber uma mensagem do tipo 'INF_EST_AGV', a notificar a passagem do AGV ENABLED ao estado DISABLED.

A transição 'DISABLE_AGV_ESP_CONF_INTR' tem a mesma funcionalidade, mas aplica-se a AGVs que estejam à espera da confirmação da sua introdução.

8.4.7 Sub-rede 'gere_AGVs_PARADOS' do SPAGV

A sub-rede 'gere_AGVs_PARADOS' está representada na Figura 8.46 (ver informação detalhada no anexo C.4.19). Esta modeliza as evoluções de estado possíveis para um AGV no estado PARADO.

A transição 'RECEBE_ORDEM_MOV_TRANSL' é disparada se o SPAGV receber uma mensagem do tipo 'MOVIMENTO', que diga respeito a um movimento de translacção. Esta inicia o movimento do AGV, configurando a temporização associada, com base na informação do custo do ramo do *layout*, através do qual o AGV se vai movimentar.

Se o movimento a iniciar for uma carga ou uma descarga, a transição 'RECEBE_ORDEM_MOV_TRCARGA' é disparada. Se o movimento for uma carga, a transição 'INICIA_OP_CARGA' é disparada, notificando o sistema de fabrico desse facto. Se o movimento for uma descarga, a transição 'INICIA_OP_DESCARGA', que tem uma funcionalidade análoga à anterior, é disparada.

A transição 'DISABLE_AGV_PARADO' é disparada, se o SPAGV receber uma mensagem do tipo 'INF_EST_AGV', a notificar a passagem do AGV PARADO ao estado DISABLED. Neste caso é necessário interromper a temporização que poderia levar o AGV a passar ao estado ERRO ou ao estado ENABLED, depositando um *token* nas posições 'INTERR_TEMP_ACT_ERRO' e 'INTERR_TEMP_ACT_MANUAL', respectivamente.

A transição 'AGV_PARADO_MANUAL' simula a passagem de um AGV PARADO ao modo manual. O seu disparo é desencadeado pela colocação de um *token* na posição 'FIM_TEMP_ACT_MANUAL'. Ao disparar, é iniciada uma nova temporização, que poderá levar à introdução futura do AGV, e é interrompida a temporização, que poderia levar à simulação de uma avaria no AGV.

A transição 'AGV_PARADO_ERRO' simula a passagem de um AGV PARADO ao estado ERRO. O seu disparo é desencadeado pela colocação de um *token* na posição 'FIM_TEMP_ACT_ERRO'.

Nos dois últimos casos, o SGAGV é notificado da alteração do estado do AGV, através do envio de uma mensagem do tipo 'AGV_MANUAL' ou 'ERRO_AGV', respectivamente.

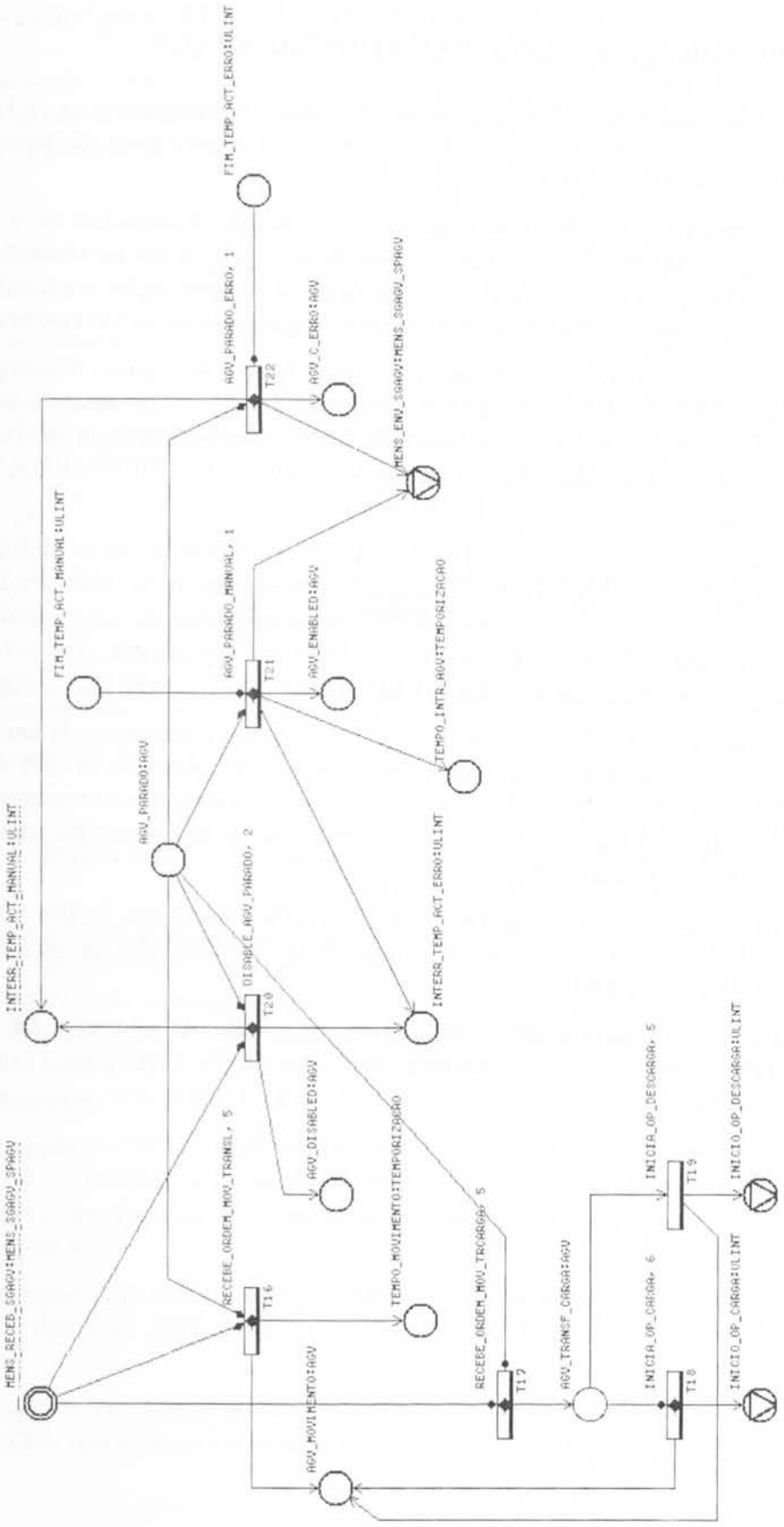


Figura 8.46 – Sub-rede 'gere_AGVs_PARADOS' do SPAGV.

8.4.8 Sub-rede 'gere_AGVs_MOVIMENTO' do SPAGV

A sub-rede 'gere_AGVs_MOVIMENTO'está representada na Figura 8.47 (ver informação detalhada no anexo C.4.18). Esta modeliza as evoluções de estado possíveis para um AGV em movimento (num dos estados EX_MOVIM, CARGA_AGV ou DESCARGA_AGV).

A transição 'PARA_AGV' é disparada quando o movimento de translacção de um AGV termina. O AGV passa ao estado PARADO, ainda que possa ser de forma transitória. Se houver movimentos em avanço, o AGV regressará ao estado EX_MOVIM em tempo nulo.

A transição 'TERMINA_OP_CARGA' é disparada quando uma operação de carga termina, e a transição 'TERMINA_OP_DESCARGA' é disparada quando uma operação de descarga termina. Em ambos os casos, o AGV passa ao estado PARADO.

Em todos os casos anteriores, o SGAGV é notificado do fim da execução do movimento, através do envio de uma mensagem do tipo 'FIM_MOVIMENTO'.

A transição 'DISABLE_AGV_MOVIMENTO' é disparada, se o SPAGV receber uma mensagem do tipo 'INF_EST_AGV', a notificar a passagem do AGV em movimento ao estado DISABLED. Neste caso é necessário interromper a temporização que poderia levar o AGV a passar ao estado ERRO ou ao estado ENABLED, depositando um *token* nas posições 'INTERR_TEMP_ACT_ERRO' e 'INTERR_TEMP_ACT_MANUAL', respectivamente.

A transição 'AGV_MOVIM_MANUAL' simula a passagem de um AGV em movimento ao modo manual. O seu disparo é desencadeado pela colocação de um *token* na posição 'FIM_TEMP_ACT_MANUAL'. Ao disparar, é iniciada uma nova temporização, que poderá levar à introdução futura do AGV, e é interrompida a temporização, que poderia levar à simulação de uma avaria no AGV.

Nos dois casos anteriores, a deposição de um *token* na posição 'AGV_DESACT' permite que a sub-rede 'processa_AGVs_desactivados' faça a gestão da desactivação do AGV.

A transição 'AGV_MOVIM_ERRO' simula a passagem de um AGV em movimento ao estado ERRO. O seu disparo é desencadeado pela colocação de um *token* na posição 'FIM_TEMP_ACT_ERRO'. Se o AGV estiver a executar uma operação de carga/descarga, a transição 'INTERR_OP_TRCAR' informa o sistema de fabrico da interrupção da operação, caso contrário, a transição disparada é a transição 'LIMPA_AGV_MOV_ERRO'.

Nos dois últimos casos, o SGAGV é notificado da alteração do estado do AGV, através do envio de uma mensagem do tipo 'AGV_MANUAL' ou 'ERRO_AGV', respectivamente.

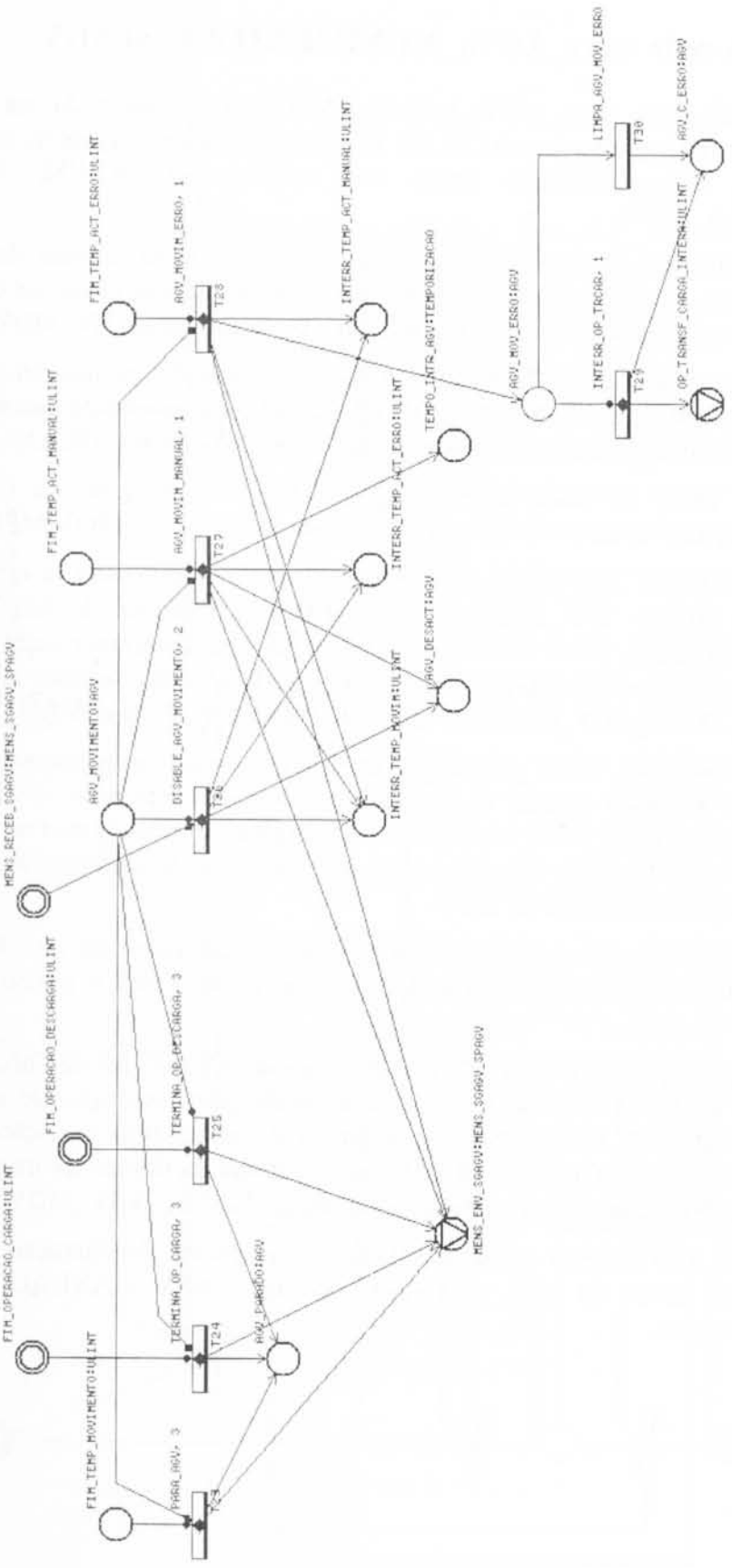


Figura 8.47 – Sub-rede 'gere_AGVs_MOVIMENTO' do SPAGV.

8.4.9 Sub-rede 'gere_AGVs_ERRO' do SPAGV

A sub-rede 'gere_AGVs_ERRO' está representada na Figura 8.48 (ver informação detalhada no anexo C.4.17). Esta simula os AGVs que passem ao estado ERRO, na sequência de uma avaria.

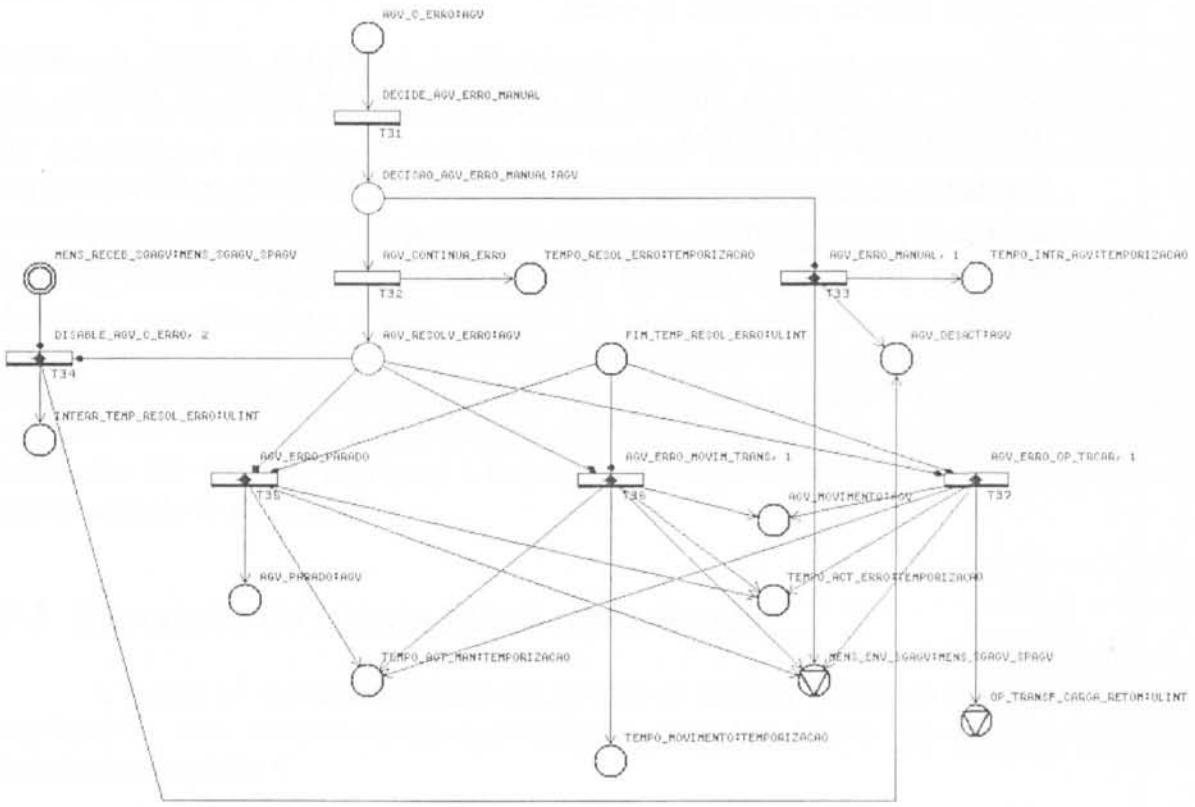


Figura 8.48 – Sub-rede 'gere_AGVs_ERRO' do SPAGV.

Com o disparo da transição 'DECIDE_AGV_ERRO_MANUAL', é decidido se o AGV vai passar ao modo manual, ou se vai continuar no estado ERRO. Esta decisão é baseada na probabilidade definida pelo parâmetro 'p_erro_manual' do SPAGV (ver parâmetros do SPAGV no anexo C.4.4).

Se for decidido que o AGV vai passar ao modo manual, a transição 'AGV_ERRO_MANUAL' é disparada. Esta coloca um *token* nas posições 'AGV_DESACT' e 'TEMPO_INTR_AGV', permitindo a gestão da operação que o AGV pudesse estar a executar, e o planeamento da futura introdução do AGV.

Se for decidido que o AGV continua no estado ERRO, é depositado um *token* em 'TEMPO_RESOL_ERRO', iniciando-se uma temporização, que simula o tempo necessário à resolução da avaria.

Terminada esta temporização, o *token* depositado em 'FIM_TEMP_RESOL_ERRO', provoca o disparo de uma de três transições:

- AGV_ERRO_PARADO – é disparada se o AGV estava no estado PARADO, antes da ocorrência da avaria;

- AGV_ERRO_MOVIM_TRANS – é disparada se o AGV estava a executar um movimento de translacção, antes da ocorrência da avaria;
- AGV_ERRO_OP_TRCAR – é disparada se o AGV estava a executar um movimento de carga/descarga, antes da ocorrência da avaria.

No segundo caso, a temporização que simula o movimento de translacção é reiniciada com um valor correspondente ao tempo que faltava para terminar o movimento, imediatamente antes da ocorrência da avaria.

No terceiro caso, o sistema de fabrico é informado de que a operação de carga/descarga foi retomada, através da deposição de um *token* na posição de saída 'OP_TRANSF_CARGA_RETOM'.

Em todos os casos, o AGV em causa é colocado no estado em que se encontrava antes da ocorrência da avaria. Também em todos os casos, são iniciadas duas temporizações, que poderão levar o AGV a passar ao modo manual, ou à ocorrência de um erro interno no AGV.

9. RP estocástica e colorida de um sistema de fabrico dotado de um SAGVs

O capítulo anterior descreveu a implementação de um modelo para as três camadas de gestão de um SAGVs (ver arquitectura ilustrada pela Figura 2.3 da página 11), baseado nas RP estocásticas e coloridas do Artifex. Este capítulo descreve a modelização no Artifex da classe 'SISTEMA_MANUFACTURA', representada na Figura 8.1 da página 117.

O capítulo começa por descrever um exemplo de uma planta fabril, servida por um SAGVs, sendo identificados os vários tipos de equipamentos existentes. De seguida, é descrita a especificação e a implementação de classes Artifex para modelizar cada um dos tipos de equipamentos identificados. O capítulo prossegue com a descrição da implementação da classe 'SISTEMA_MANUFACTURA', que modeliza o sistema de fabrico utilizado como exemplo nas simulações apresentadas no capítulo 10. No final do capítulo é evidenciada a integração dos 4 blocos da Figura 8.1, que permite obter o modelo completo do sistema de fabrico dotado de um SAGVs.

9.1 Descrição da planta fabril modelizada

A Figura 9.1 mostra a planta de um sistema de fabrico, imaginada para interactuar com um SAGVs, cujo comportamento é modelizado pelos três blocos das camadas de gestão, descritos no capítulo 8.

A planta fabril é constituídas pelas seguintes células:

- um cais de recepção de matérias-primas (CRMP);
- um armazém automático de matéria-prima (AAMP);
- quatro centros de maquinação CM1, CM2, CM3 e CM4;
- seis postos de trabalho manuais PM1, PM2, PM3, PM4, PM5 e PM6;
- uma célula de montagem (CM);
- um centro de controle de qualidade (CQ);
- um armazém automático de produto acabado (AAPA);
- um cais de expedição de produto acabado (CEPA).

Os pontos indicados ao longo do *layout* do sistema de fabrico são os nós que serão servidos pelo SAGVs, ou seja, os pontos de entrada e de saída de paletes. Os pontos de carga de AGVs são indicados por uma seta que aponta para fora da célula, e os pontos de descarga são indicados por uma seta em sentido contrário.

As paletes entram na fábrica através do cais de recepção de matérias-primas, podendo ser armazenadas no respectivo armazém, ou encaminhadas directamente até um posto de trabalho. De acordo com o caminho tecnológico das peças a produzir, as paletes vão migrando entre os vários postos de trabalho. A célula de montagem realiza a montagem de

três componentes, produzindo um produto final (acabado). Este é encaminhado até ao centro de controle de qualidade, ou até ao armazém de produto acabado. As peças saiem da fábrica através do cais de expedição.

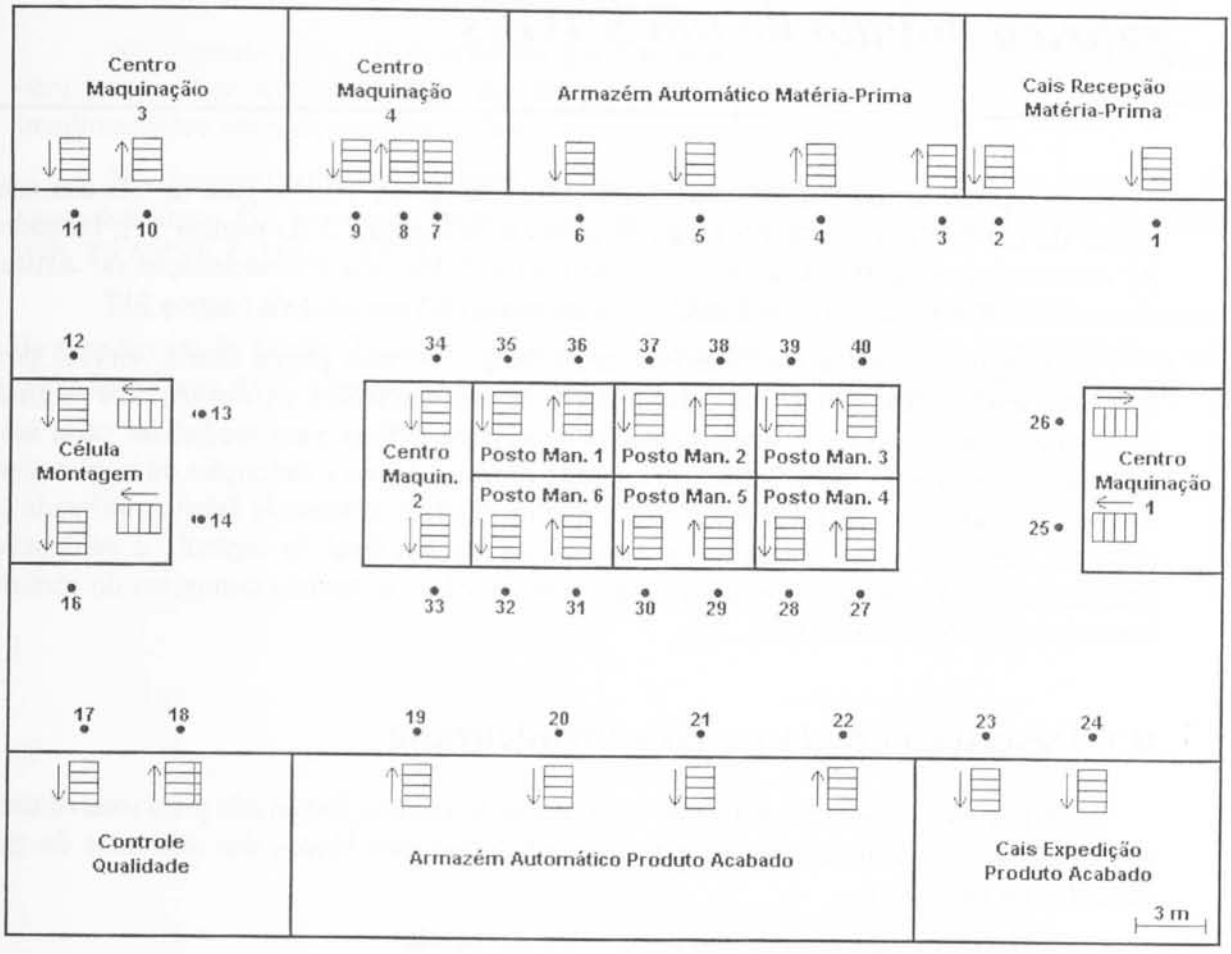


Figura 9.1 – Desenho da planta fabril exemplo utilizada na realização do trabalho.

No sistema de fabrico descrito é possível identificar quatro tipos de entidades:

- centro de recepção de matérias-primas, com um número arbitrário de pontos de carga (CRMP);
- armazém automático, com um número arbitrário de pontos de carga e de descarga (AAMP e AAPA);
- posto de trabalho manual ou automatizado, com um número arbitrário de pontos de descarga e um ponto de carga (CM1, CM2, CM3, CM4, PM1, PM2, PM3, PM4, PM5, PM6, CM e CQ);
- centro de expedição de produto acabado, com um número arbitrário de pontos de descarga (CEPA).

Cada um destes tipos de equipamento foi modelizado por uma classe Artifex. A especificação e descrição dessas classes é feita na secção seguinte. O modelo completo do sistema de fabrico é descrito na última secção.

9.2 Modelização das entidades do sistema de fabrico

A modelização das entidades do sistema de fabrico identificadas na secção anterior será descrita nesta secção.

Para o efeito criaram-se as seguintes classes Artifex:

- CENTRO_RECEPCAO;
- ARMAZEM_AUTOMATICO;
- POSTO_TRABALHO;
- CENTRO_EXPEDICAO.

9.2.1 Classe CENTRO_RECEPCAO

A classe CENTRO_RECEPCAO pretende modelizar uma entidade de um sistema de fabrico, por onde entram paletes no sistema. Admite-se que as paletes são recebidas em lotes com um número fixo de unidades, e que as paletes recebidas são encaminhadas até aos vários pontos de carga de AGVs, onde existem *buffers* com uma determinada capacidade.

A especificação desta entidade pressupõe:

- a existência de um número arbitrário de pontos de carga, dotados de um *buffer* (fila de saída FIFO), cuja capacidade é configurável e igual para todos os pontos;
- o tempo que medeia entre a recepção de dois lotes consecutivos é configurável e descrito por uma FDP exponencial;
- o número de recursos disponíveis para o transporte das paletes, desde o cais até aos pontos de carga, é configurável;
- o tempo que um recurso demora a transportar uma palete, desde o cais até uma das filas de saída, é configurável e descrito por uma FDP normal;
- o tempo que demora a execução de uma operação de carga num AGV é configurável e descrito por uma FDP normal.

O anexo D.1 contém informação detalhada sobre a implementação da classe CENTRO_RECEPCAO. A página principal da RP desta classe está representada na Figura 9.2 (ver informação detalhada no anexo D.1.11). O comportamento e configuração da entidade é definida pelos parâmetros apresentados no anexo D.1.4.

As posições de entrada definidas são:

- INICIO_OPERACAO_CARGA:ULINT – posição através da qual pode ser recebido um *token*, indicando a fila de saída em que se iniciou a carga de um AGV;
- ERRO_TEMP_AGV:ULINT – posição através da qual a entidade é notificada da ocorrência de um erro num AGV, a executar uma operação de carga no ponto indicado pelo *token* recebido, implicando a interrupção da operação;
- ERRO_PERM_AGV:ULINT – posição através da qual a entidade é notificada da ocorrência de um erro permanente num AGV, a executar uma operação de carga no ponto indicado pelo *token*, implicando o cancelamento definitivo da operação;

- ERRO_TEMP_RESOLV:ULINT – posição através da qual a entidade é notificada da resolução de um erro num AGV, a executar uma operação de carga no ponto indicado pelo *token*, permitindo a continuação da execução da operação.

As posições de saída definidas são:

- FIM_OPERACAO_CARGA:ULINT – posição onde é depositado um *token*, indicando a fila de saída em que uma operação de carga terminou;
- PONTOS_SAIDA_DISPONIVEIS:ULINT – posição onde é depositado um *token*, indicando a fila de saída que ficou disponível para a realização de uma operação de carga;
- PAL_ACABA_SAIR_ARMAZEM:PALETE – posição que recebe um *token* sempre que seja retirada uma palete da entidade;
- CAPACIDADE_FILA:CAPAC_FILA – posição através da qual é enviado ao exterior um *token* no arranque da simulação, informando da capacidade dos *buffers* das filas de saída;
- PALETE_FILA_SAIDA:PALETE – posição onde são depositados *tokens* correspondentes a paletes inseridas nos *buffers* das filas de saída;
- FILA_PALETE_INSERIDA:ULINT – posição onde são colocados *tokens* a informar o exterior sobre as filas onde são inseridas paletes.

Os objectos do tipo 'TEMPORIZADOR', que fazem parte da RP da entidade, estão descritos no anexo D.1.11.1.

Quando chega um lote de paletes ao cais, a transição 'NOVO_LOTE_PALETES' é disparada. Logo de seguida, a transição 'TERMINA_RECEPCAO_LOTE' é disparada, configurando o tempo de disparo da transição 'RECEBE_LOTE_PALETES' (tempo até à recepção do próximo lote), e actualizando o valor guardado pelo *token* existente na posição 'NUM_PALETES_A_RECEBER'. Este indica o número de paletes existentes no cais.

Os *tokens* existentes na posição 'RECURSOS_LIVRES' indicam os recursos disponíveis para o transporte de paletes, desde o cais até às filas de saída. Quando houver uma palete no cais e um recurso disponível, a transição 'INICIA_RECEPCAO_PALETE' inicia a temporização, que simula o tempo necessário ao transporte da palete desde o cais até à zona das filas de saída. Terminada a temporização, a transição 'PALETE_CHEGA_FILA' é disparada. Havendo um *token* em 'POSICOES_LIVRES_FILAS', a palete é colocada na fila correspondente, e é libertado o recurso utilizado na recepção da palete, aquando do disparo da transição 'COLOCA_PALETE_FILA_SAIDA'.

A parte restante da RP representa a realização de operações de carga de AGVs. Quando é depositado um *token* em 'INICIO_OPERACAO_CARGA', a transição 'INICIA_CARGA_AGV' coloca um *token* na posição 'CARREGANDO _AGV', correspondente à palete que esteja a ser carregada. A transição 'TERMINA_CARGA_AGV' dispara quando a operação de carga termina, libertando a posição no *buffer* donde foi retirada a palete carregada.

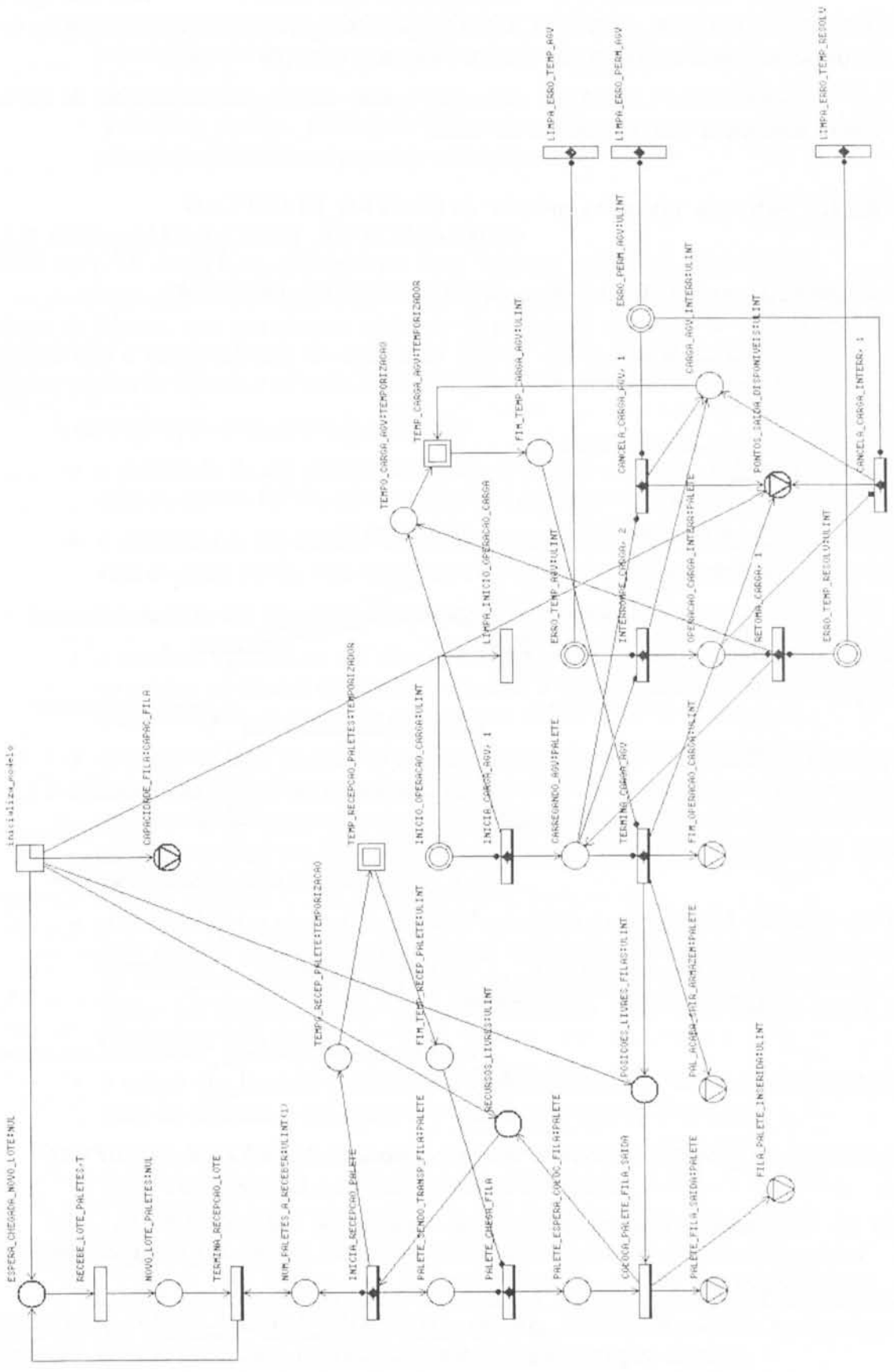


Figura 9.2 – RP da página principal da classe CENTRO_RECEPCAO.

As transições 'INTERROMPE_CARGA', 'RETOMA_CARGA', 'CANCELA_CARGA_AGV' e 'CANCELA_CARGA_INTERR' gerem situações de avaria ou de desactivação de AGVs que estejam a executar operações de carga.

As transições 'LIMPA_...' consomem *tokens* das respectivas posições de entrada, que não se destinem a filas da entidade em causa.

9.2.1.1 Sub-rede 'inicializa_modelo' do CENTRO_RECEPCAO

A sub-rede 'inicializa_modelo' está representada na Figura 9.3 (ver informação detalhada no anexo D.1.12). Esta modeliza a configuração da entidade.

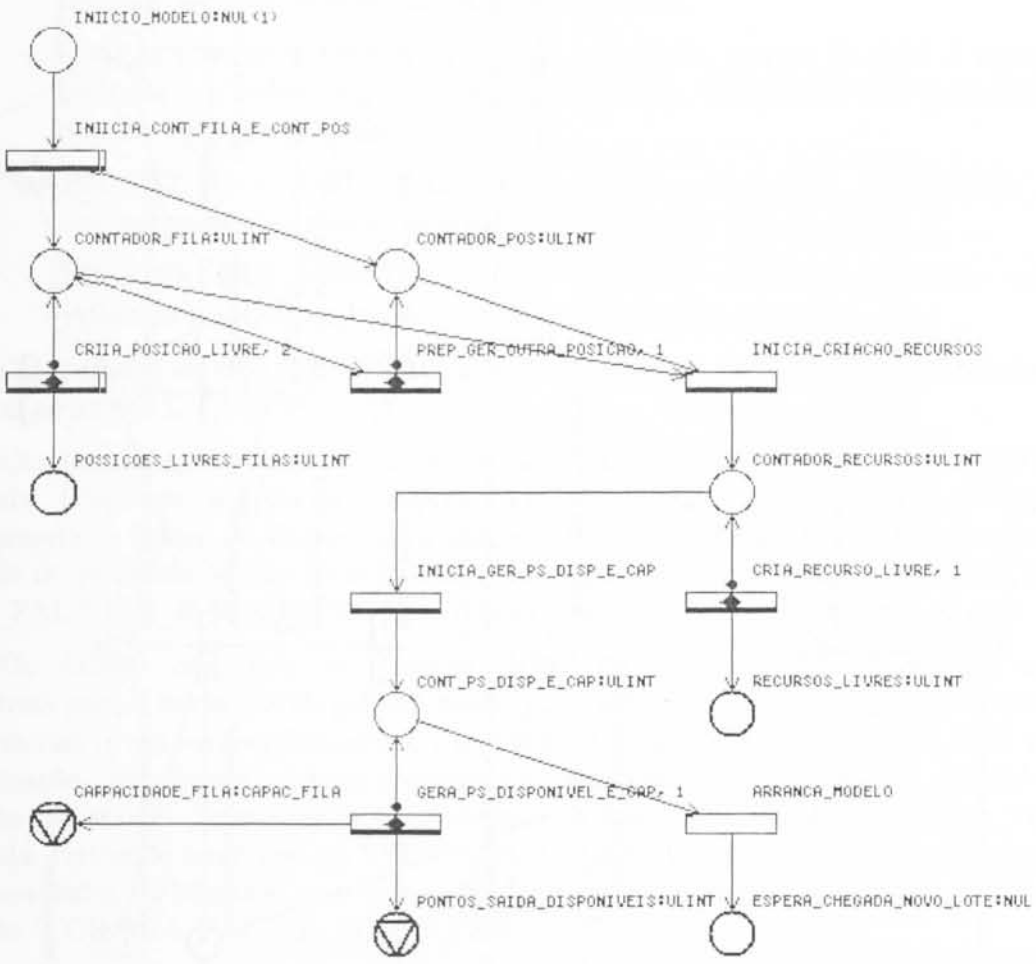


Figura 9.3 - Sub-rede 'inicializa_modelo' do CENTRO_RECEPCAO.

De acordo com os valores dos parâmetros 'capacidade_filas', 'fila1', 'filan' e 'num_recursos' (ver parâmetros da classe no anexo D.1.4), a RP representada realiza as acções seguintes:

- insere *tokens* na posição 'POSICOES_LIVRES_FILAS', representando as posições disponíveis nos *buffers* (em número igual a 'capacidade_filas' para todas as filas);

- insere *tokens* na posição 'RECURSOS_LIVRES', representando os recursos existentes para o transporte de paletes desde o cais até às filas;
- por cada fila, insere um *token* nas posições 'CAPACIDADE_FILA' e 'PONTOS_SAIDA_DISPONIVEIS', informando o exterior da capacidade da fila e da disponibilidade do ponto de saída associado.

9.2.2 Classe ARMAZEM_AUTOMATICO

A classe ARMAZEM_AUTOMATICO pretende modelizar uma entidade de um sistema de fabrico, com um número arbitrário de pontos de entrada e de saída de paletes. A entidade tem a funcionalidade de armazenar paletes, até ao limite da sua capacidade. Em todos os pontos de entrada e de saída existem *buffers* com uma determinada capacidade.

A especificação desta entidade pressupõe:

- a existência de um número arbitrário de pontos de carga, dotados de um *buffer* (fila de entrada FIFO), cuja capacidade é configurável e igual em todos os pontos;
- a existência de um número arbitrário de pontos de descarga, dotados de um *buffer* (fila de saída FIFO), cuja capacidade é configurável e igual em todos os pontos;
- o armazém tem uma determinada capacidade configurável;
- a entidade é dotada de um recurso designado por stocador, que permite manusear as paletes no interior do armazém e aceder a todas as células, a todos os pontos de carga e a todos os pontos de descarga;
- o tempo de uma viagem do stocador descarregado é configurável e descrito por uma FDP normal; o movimento do stocador descarregado pode ocorrer quando este acaba de descarregar uma palete numa célula do armazém, ou numa fila de saída, e se dirige em seguida a uma célula do armazém ou a uma fila de entrada, para carregar uma palete;
- o tempo de uma viagem do stocador carregado é configurável e descrito por uma FDP normal; o movimento do stocador carregado pode ocorrer quando este acaba de carregar uma palete numa célula do armazém, ou numa fila de entrada, e se dirige em seguida a uma célula do armazém ou a uma fila de saída, para descarregar uma palete;
- o tempo que demora a execução de uma operação de carga ou de descarga por parte do stocador é configurável e descrito por uma FDP normal;
- o tempo que demora a execução de uma operação de carga ou de descarga num AGV é configurável e descrito por uma FDP normal.

O anexo D.2 contém informação detalhada sobre a implementação da classe ARMAZEM_AUTOMATICO.

A página principal da RP da classe ARMAZEM_AUTOMATICO está representada na Figura 9.4 (ver informação detalhada no anexo D.2.11). O comportamento e configuração da entidade é definida pelos parâmetros apresentados no anexo D.2.4.

As posições de entrada definidas são:

- INICIO_OPERACAO_DESCARGA:ULINT – posição através da qual pode ser recebido um *token*, indicando a fila de entrada em que se iniciou a descarga de um AGV;
- INICIO_OPERACAO_CARGA:ULINT – posição através da qual pode ser recebido um *token*, indicando a fila de saída em que se iniciou a carga de um AGV;
- ERRO_TEMP_AGV:ULINT – posição através da qual a entidade é notificada da ocorrência de um erro num AGV, a executar uma operação de carga/descarga no ponto indicado pelo *token* recebido, implicando a interrupção da operação;
- ERRO_PERM_AGV:ULINT – posição através da qual a entidade é notificada da ocorrência de um erro permanente num AGV, a executar uma operação de carga/descarga no ponto indicado pelo *token*, implicando o cancelamento definitivo da operação;
- ERRO_TEMP_RESOLV:ULINT – posição através da qual a entidade é notificada da resolução de um erro num AGV, a executar uma operação de carga/descarga no ponto indicado pelo *token*, permitindo a continuação da execução da operação;
- PEDIDO_SAIDA:PALETE – posição que pode receber um *token* representando uma palete existente no armazém, que o stocador deve colocar numa das filas de saída, a pedido do exterior.

As posições de saída definidas são:

- FIM_OPERACAO_DESCARGA:ULINT – posição onde é depositado um *token*, indicando a fila de entrada em que uma operação de descarga terminou;
- FIM_OPERACAO_CARGA:ULINT – posição onde é depositado um *token*, indicando a fila de saída em que uma operação de carga terminou;
- PONTOS_SAIDA_DISPONIVEIS:ULINT – posição onde é depositado um *token*, indicando a fila de saída que ficou disponível para a realização de uma operação de carga;
- POSICOES_LIVRES_FILAS_ENTR:ULINT – posição onde pode ser depositado um *token*, indicando uma fila de entrada em que há uma posição disponível;
- POSICOES_LIVRES_FILAS_SAIDA:ULINT – posição onde pode ser depositado um *token*, indicando uma fila de saída em que há uma posição disponível;
- PALETES_ARMAZENADAS:PALETE – posição onde é depositado um *token*, sempre que uma palete seja armazenada numa das células do armazém;
- PAL_RETIDAS_SAIDA_ARMAZEM:PALETE – posição onde é depositado um *token*, sempre que uma palete for descarregada pelo stocador numa das filas de saída do armazém;
- PAL_ACABA_SAIR_ARMAZEM:PALETE – posição que recebe um *token* sempre que seja retirada uma palete da entidade;

- CAPACIDADE_FILA:CAPAC_FILA – posição através da qual é enviado ao exterior um *token* no arranque da simulação, informando da capacidade dos *buffers* das filas de saída;
- FILA_PALETE_INSERIDA:ULINT – posição onde são colocados *tokens* a informar o exterior sobre as filas de saída onde são inseridas paletes.

O objecto do tipo 'TEMPORIZADOR', que faz parte da RP da entidade, é utilizado para temporizar as operações de carga e de descarga de AGVs nas filas do armazém.

Os *tokens* da posição 'CELULAS_LIVRES' representam células do armazém disponíveis para armazenarem paletes.

Quando a posição 'STOCADOR_LIVRE' contém um *token*, isso significa que o stocador está disponível para efectuar um movimento. Esta serve portanto para fazer a partilha desse recurso.

As transições 'LIMPA_ERRO_...' consomem *tokens* das respectivas posições de entrada, que não digam respeito a filas pertencentes ao armazém.

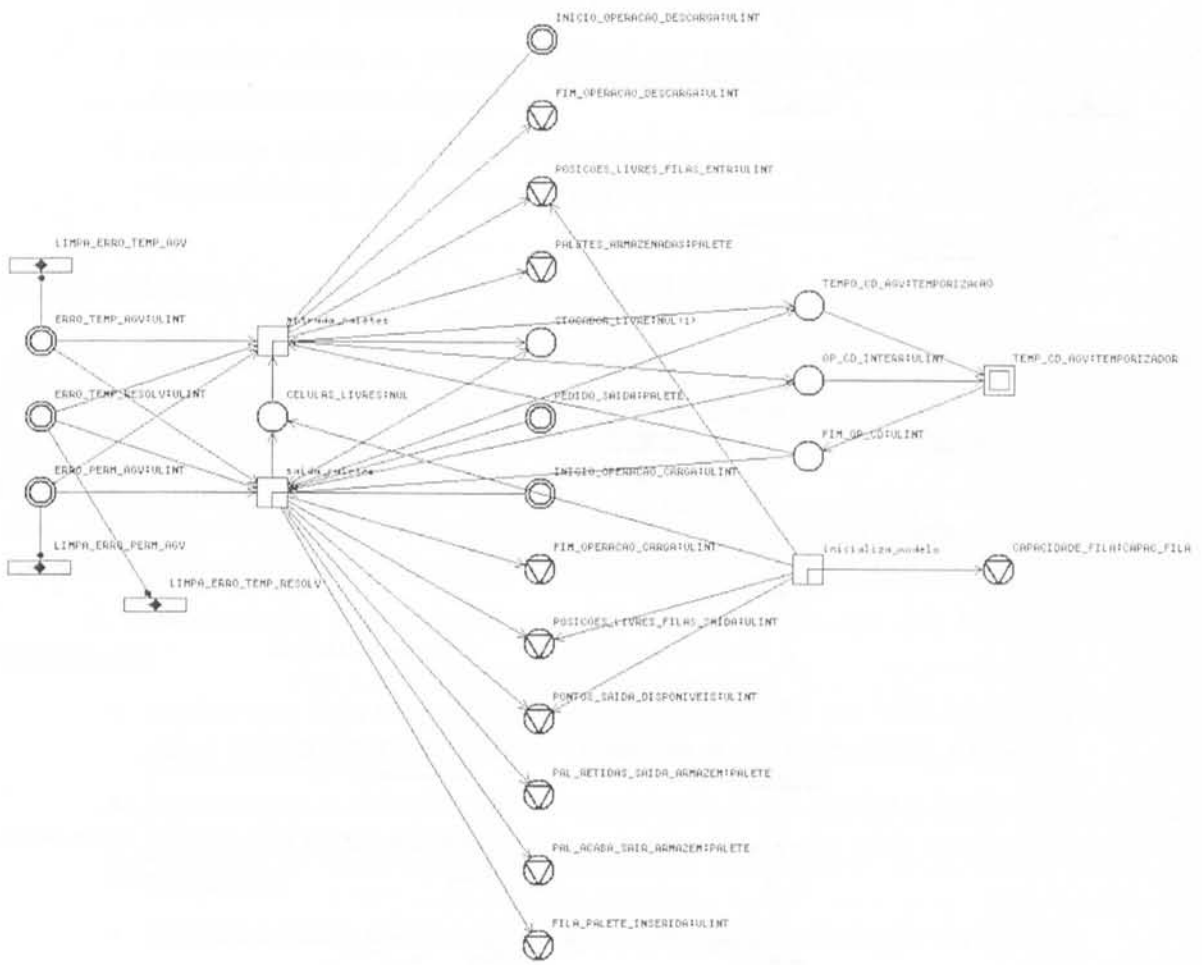


Figura 9.4 – RP da página principal da classe ARMAZEM_AUTOMATICO.

A RP' da página principal é constituída pelas sub-redes seguintes:

- inicializa_modelo,
- entrada_paletes e
- saida_paletes.

9.2.2.1 Sub-rede 'inicializa_modelo' da classe ARMAZEM_AUTOMATICO

A sub-rede 'inicializa_modelo' está representada na Figura 9.5 (ver informação detalhada no anexo D.2.13). Esta representa a configuração do modelo do armazém.

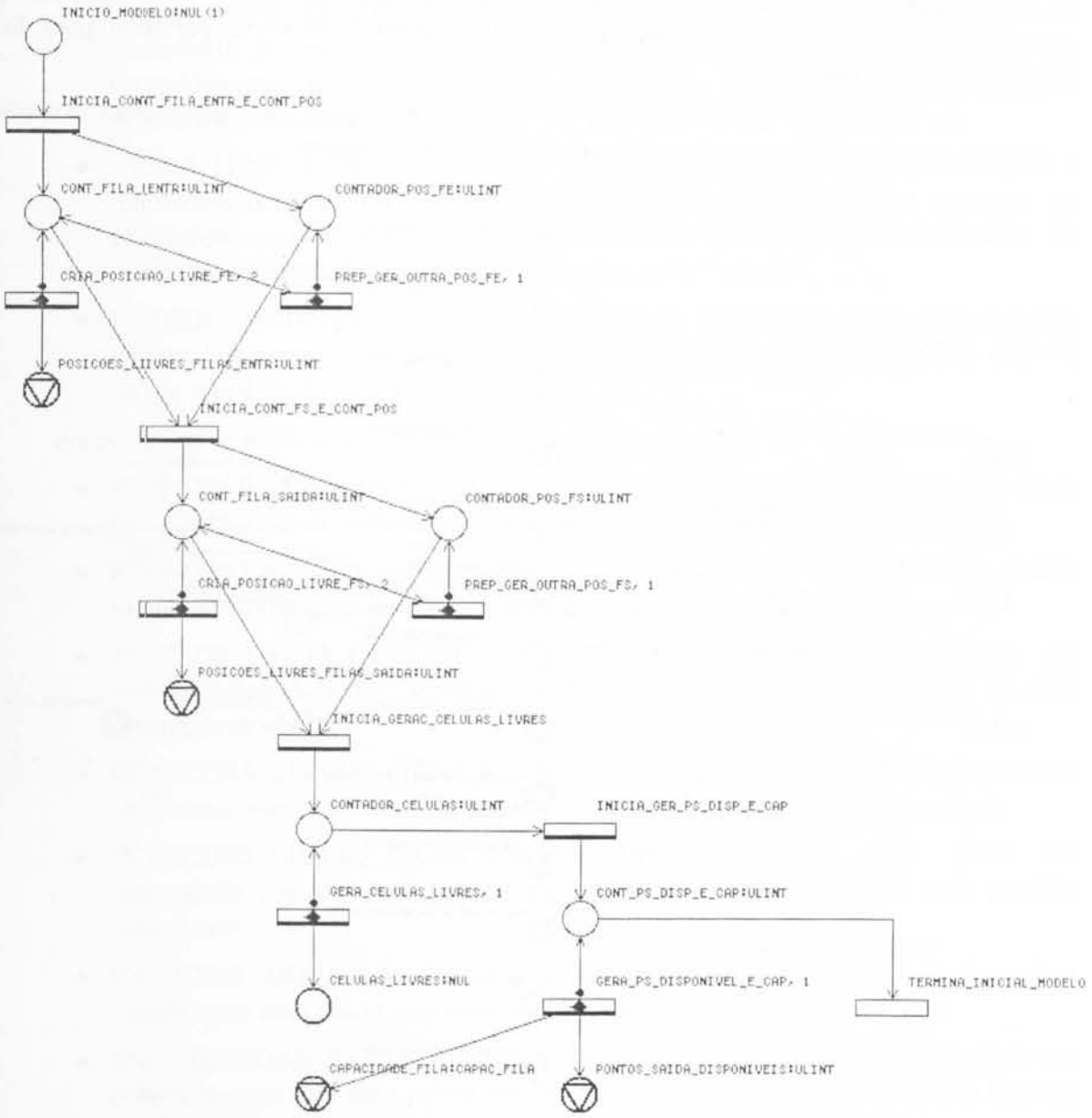


Figura 9.5 – Sub-rede 'inicializa_modelo' da classe ARMAZEM_AUTOMATICO.

Os parâmetros que influenciam a configuração são:

- fila_entr_1,
- fila_entr_n,
- fila_saida_1,
- fila_saida_n,
- capac_filas_entrada,
- capac_filas_saida e
- capac_armazem.

A configuração consiste em:

- depositar *tokens* na posição 'POSICOES_LIVRES_FILAS_ENTR', representando posições livres nos *buffers* das filas de entrada;
- depositar *tokens* na posição 'POSICOES_LIVRES_FILAS_SAIDA', representando posições livres nos *buffers* das filas de saída;
- depositar *tokens* na posição 'CELULAS_LIVRES', representando a capacidade de armazenamento disponível;
- depositar *tokens* na posição 'PONTOS_SAIDA_DISPONIVEIS', representando a disponibilidade dos pontos de carga para efectuar a saída de paletes.

9.2.2.2 Sub-rede 'entrada_paletes' da classe ARMAZEM_AUTOMATICO

A sub-rede 'entrada_paletes' está representada na Figura 9.6 (ver informação detalhada no anexo D.2.12). Esta representa a gestão de movimentos de descarga dos AGVs nos pontos de entrada do armazém, e a simulação de movimentos do stocador para transportar paletes desde os *buffers* de entrada até às células do armazém.

A gestão de movimentos de descarga dos AGVs é análoga à gestão de movimentos de carga dos AGVs, que foi descrita em 9.2.1 para a classe CENTRO_RECEPCAO.

A movimentação e armazenamento de uma palete de um dos *buffers* de entrada consiste em:

- esperar que haja uma célula disponível (um *token* em 'CELULAS_LIVRES') e que o stocador esteja disponível (um *token* em 'STOCADOR_LIVRE');
- esperar que o stocador se desloque desde a sua posição final, após o último movimento por si efectuado, até ao *buffer* de entrada, onde se encontra a palete a armazenar;
- carregar a palete no stocador e libertar uma posição do *buffer* de entrada;
- esperar que o stocador se movimente desde o *buffer* de entrada até à célula do armazém;
- esperar que o *stocador* descarregue a palete na célula do armazém;

- informar o exterior sobre o armazenamento da palete, através da deposição de um *token* em 'PALETES_ARMAZENADAS', e libertar o recurso stocador para a realização de outros movimentos, através da deposição de um *token* em 'STOCADOR_LIVRE'.

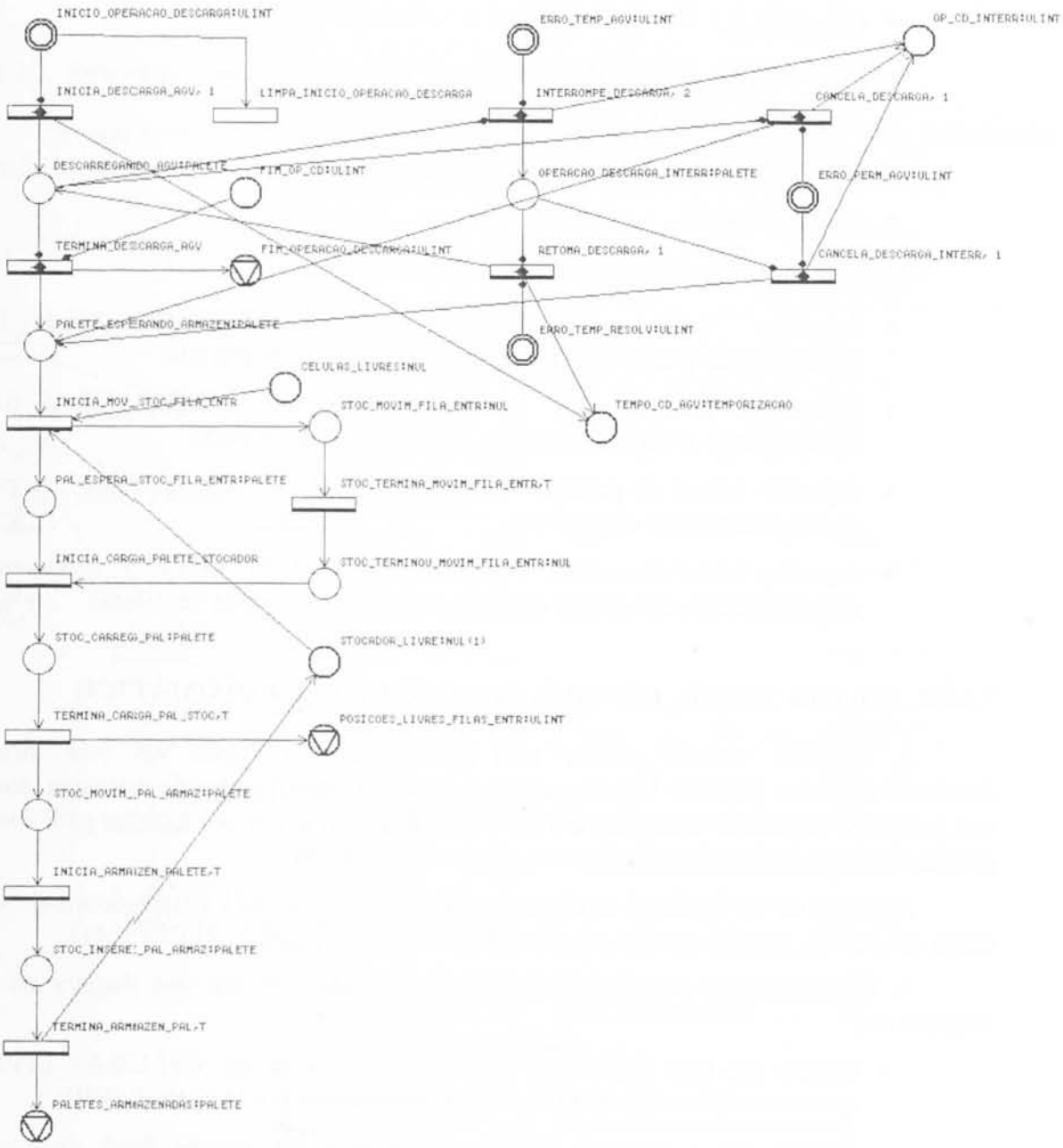


Figura 9.6 – Sub-rede 'entrada_paletes' da classe ARMAZEM_AUTOMATICO.

9.2.2.3 Sub-rede 'saida_paletes' da classe ARMAZEM_AUTOMATICO

A sub-rede 'saida_paletes' está representada na Figura 9.7 (ver informação detalhada no anexo D.2.14). Esta representa a simulação de movimentos do stocador para transportar

paletes desde as células do armazém até aos *buffers* de saída, e a gestão de movimentos de carga dos AGVs nos pontos de saída do armazém.

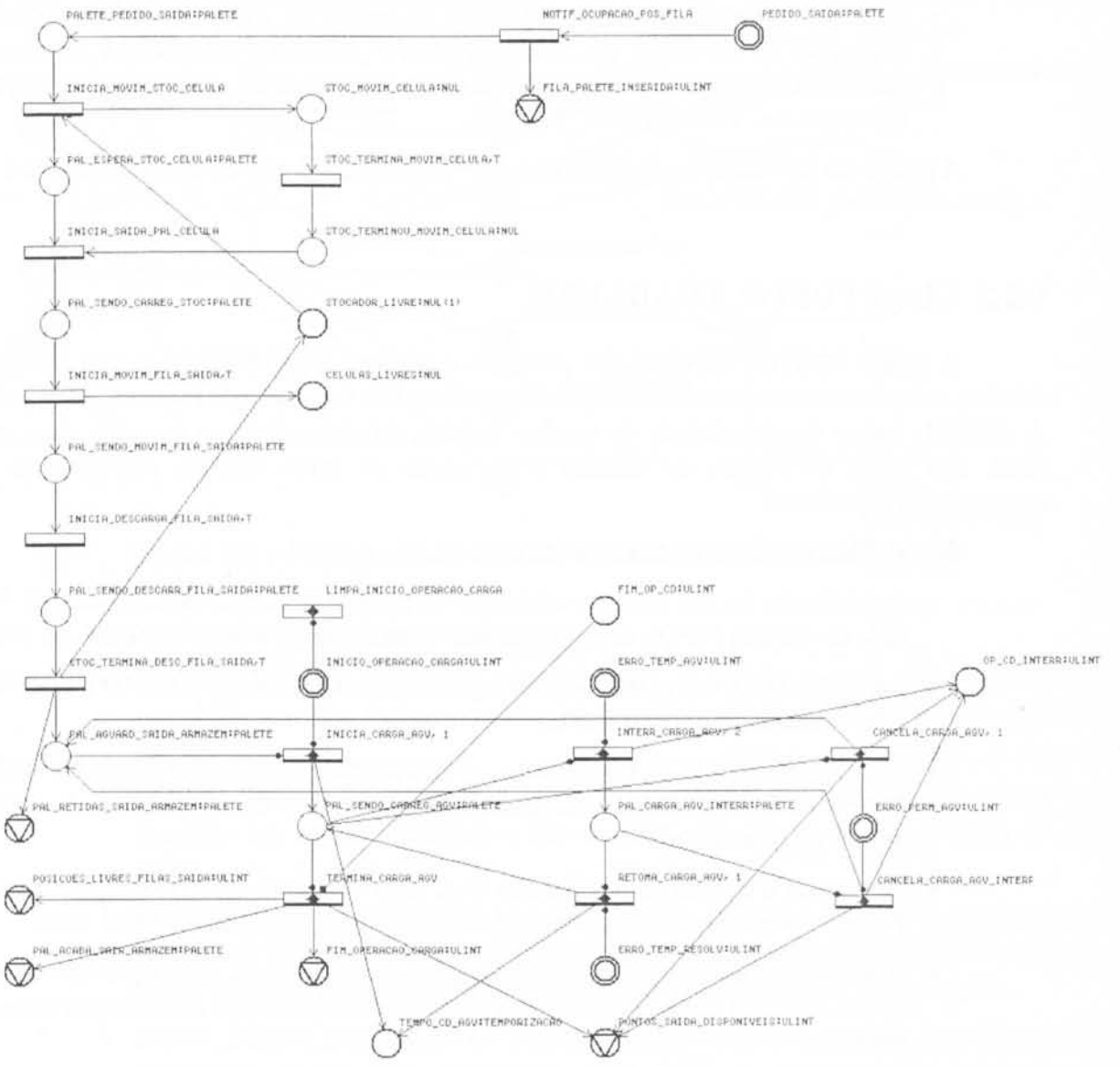


Figura 9.7 – Sub-rede 'saida_paletes' da classe ARMAZEM_AUTOMATICO.

A movimentação de uma palete armazenada até um dos *buffers* de saída consiste em:

- esperar que haja um pedido para retirar uma palete do armazém (*token* em 'PEDIDO_SAIDA') e que o stocador esteja disponível (um *token* em 'STOCADOR_LIVRE');
- esperar que o stocador se desloque desde a sua posição final, após o último movimento por si efectuado, até à célula do armazém onde se encontra a palete armazenada;
- carregar a palete no stocador e libertar a célula onde se encontrava armazenada, depositando um *token* em 'CELULAS_LIBRES';

- esperar que o stocador se movimente desde a célula do armazém até ao *buffer* de saída;
- esperar que o *stocador* descarregue a palete no *buffer* de saída;
- informar o exterior sobre a deposição da palete no *buffer* de saída, através da deposição de um *token* em 'PAL_RETIDAS_SAIDA_ARMAZEM', e libertar o recurso stocador para a realização de outros movimentos, através da deposição de um *token* em 'STOCADOR_LIVRE'.

A gestão de movimentos de carga dos AGVs é análoga à que foi descrita em 9.2.1 para a classe CENTRO_RECEPCAO.

9.2.3 Classe POSTO_TRABALHO

A classe POSTO_TRABALHO pretende modelizar uma entidade de um sistema de fabrico, com um número arbitrário de pontos de entrada e com um ponto de saída de paletes. A entidade tem a funcionalidade de receber paletes, de processá-las e de gerar uma palete final. Em todos os pontos de entrada e no ponto de saída existem *buffers* com uma determinada capacidade.

A especificação desta entidade pressupõe:

- a existência de um número arbitrário de pontos de carga, dotados de um *buffer* (fila de entrada FIFO), cuja capacidade é configurável e igual em todos os pontos;
- a existência de um ponto de descarga, dotado de um *buffer* (fila de saída FIFO), cuja capacidade é configurável;
- o processamento só se pode iniciar, se houver pelo menos uma palete em todos os *buffers* de entrada; o início do processamento implica a remoção de uma palete de cada *buffer* de entrada;
- a entidade é dotada de um recurso (e.g. uma máquina), que permite manusear as paletes no interior do armazém e efectuar operações produtivas sobre elas;
- o tempo que demora a execução por parte do recurso, de uma operação de carga ou de descarga de uma palete num *buffer*, é configurável e descrito por uma FDP normal;
- o *download* das paletes dos *buffers* de entrada no recurso ocorre em simultâneo para todas as paletes de entrada, ou seja, o tempo anterior só é contabilizado uma vez em cada *download*, mesmo que haja mais do que uma palete de entrada;
- o tempo que demora o processamento de um conjunto de paletes (tantas quanto o número de *buffers* de entrada existentes) é configurável e descrito por uma FDP normal.

O anexo D.3 contém informação detalhada sobre a implementação da classe POSTO_TRABALHO.

A página principal da RP da classe POSTO_TRABALHO está representada na Figura 9.8 (ver informação detalhada no anexo D.3.10). O comportamento e configuração da entidade é definida pelos parâmetros apresentados no anexo D.3.4.

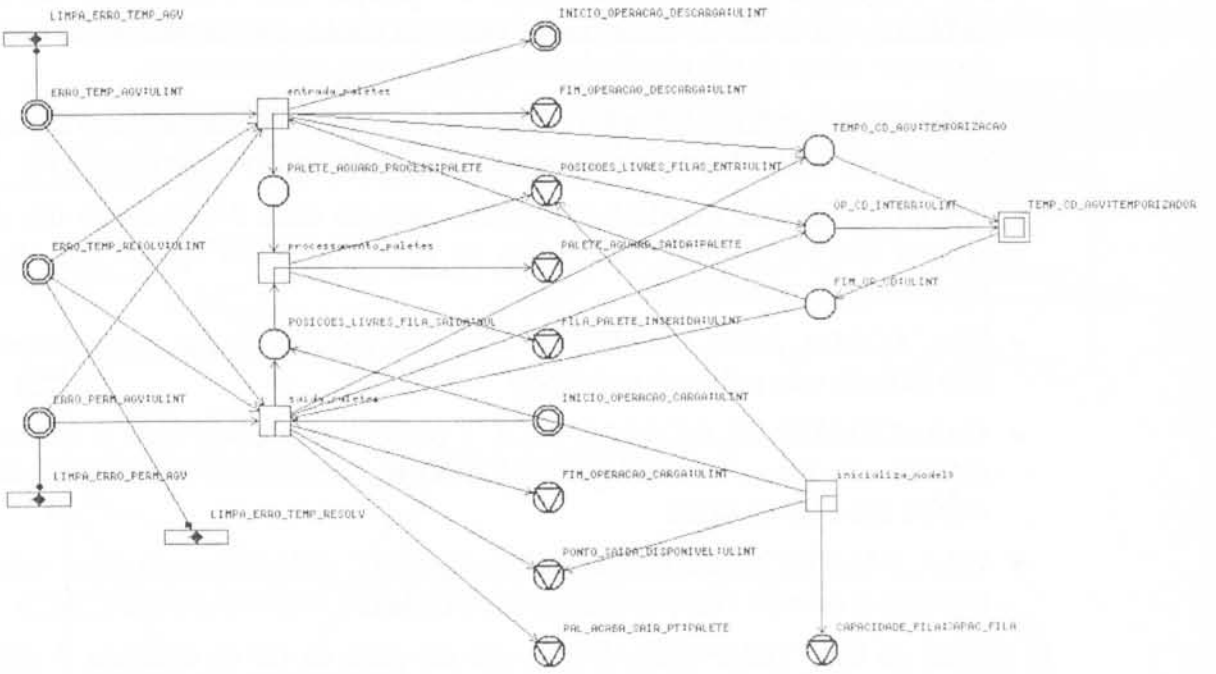


Figura 9.8 – Página principal da RP da classe POSTO_TRABALHO.

As posições de entrada definidas são:

- INICIO_OPERACAO_DESCARGA:ULINT – posição através da qual pode ser recebido um *token*, indicando a fila de entrada em que se iniciou a descarga de um AGV;
- INICIO_OPERACAO_CARGA:ULINT – posição através da qual pode ser recebido um *token*, indicando a fila de saída em que se iniciou a carga de um AGV;
- ERRO_TEMP_AGV:ULINT – posição através da qual a entidade é notificada da ocorrência de um erro num AGV, a executar uma operação de carga/descarga no ponto indicado pelo *token* recebido, implicando a interrupção da operação;
- ERRO_PERM_AGV:ULINT – posição através da qual a entidade é notificada da ocorrência de um erro permanente num AGV, a executar uma operação de carga/descarga no ponto indicado pelo *token*, implicando o cancelamento definitivo da operação;
- ERRO_TEMP_RESOLV:ULINT – posição através da qual a entidade é notificada da resolução de um erro num AGV, a executar uma operação de carga/descarga no ponto indicado pelo *token*, permitindo a continuação da execução da operação.

As posições de saída definidas são:

- FIM_OPERACAO_DESCARGA:ULINT – posição onde é depositado um *token*, indicando a fila de entrada em que uma operação de descarga terminou;
- FIM_OPERACAO_CARGA:ULINT – posição onde é depositado um *token*, indicando a fila de saída em que uma operação de carga terminou;

- P'ONTO_SAIDA_DISPONIVEL:ULINT – posição onde é depositado um *token*, indicando que a fila de saída ficou disponível para a realização de uma operação de carga;
- P'OSICOES_LIVRES_FILAS_ENTR:ULINT – posição onde pode ser depositado um *token*, indicando uma fila de entrada em que há uma posição disponível;
- P'ALETE_AGUARD_SAIDA:PALETE – posição onde é depositado um *token*, sempre que for inserida uma palete no *buffer* de saída, após o fim de uma operação de processamento;
- P'AL_ACABA_SAIR_PT:PALETE – posição que recebe um *token* sempre que seja retirada uma palete da entidade;
- CAPACIDADE_FILA:CAPAC_FILA – posição através da qual é enviado ao exterior um *token* no arranque da simulação, informando da capacidade dos *buffers* das filas de saída;
- FILA_PALETE_INSERIDA:ULINT – posição onde são colocados *tokens* a informar o exterior sobre as filas de saída onde são inseridas paletes.

O objecto do tipo 'TEMPORIZADOR', que faz parte da RP da entidade, é utilizado para temporizar as operações de carga e de descarga de AGVs nas filas da entidade.

As transições 'LIMPA_ERRO_...' consomem *tokens* das respectivas posições de entrada, que não digam respeito a filas pertencentes à entidade.

A RP da página principal é constituída pelas sub-redes seguintes:

- inicializa_modelo,
- entrada_paletes,
- processamento_paletes e
- saida_paletes.

9.2.3.1 Sub-rede 'inicializa_modelo' da classe POSTO_TRABALHO

A sub-rede 'inicializa_modelo' está representada na Figura 9.9 (ver informação detalhada no anexo D.3.12). Esta representa a configuração do modelo.

Os parâmetros que influenciam a configuração são:

- fila_entr_1,
- fila_entr_n,
- fila_saida,
- capac_filas_entrada e
- capac_filas_saida.

A configuração consiste em:

- depositar *tokens* na posição 'POSICOES_LIVRES_FILAS_ENTR', representando posições livres nos *buffers* das filas de entrada;

- depositar *tokens* na posição 'POSICOES_LIVRES_FILA_SAIDA', representando posições livres no *buffer* da fila de saída;
- depositar um *token* na posição 'PONTO_SAIDA_DISPONIVEL', representando a disponibilidade do ponto de carga para efectuar a saída de uma palete.

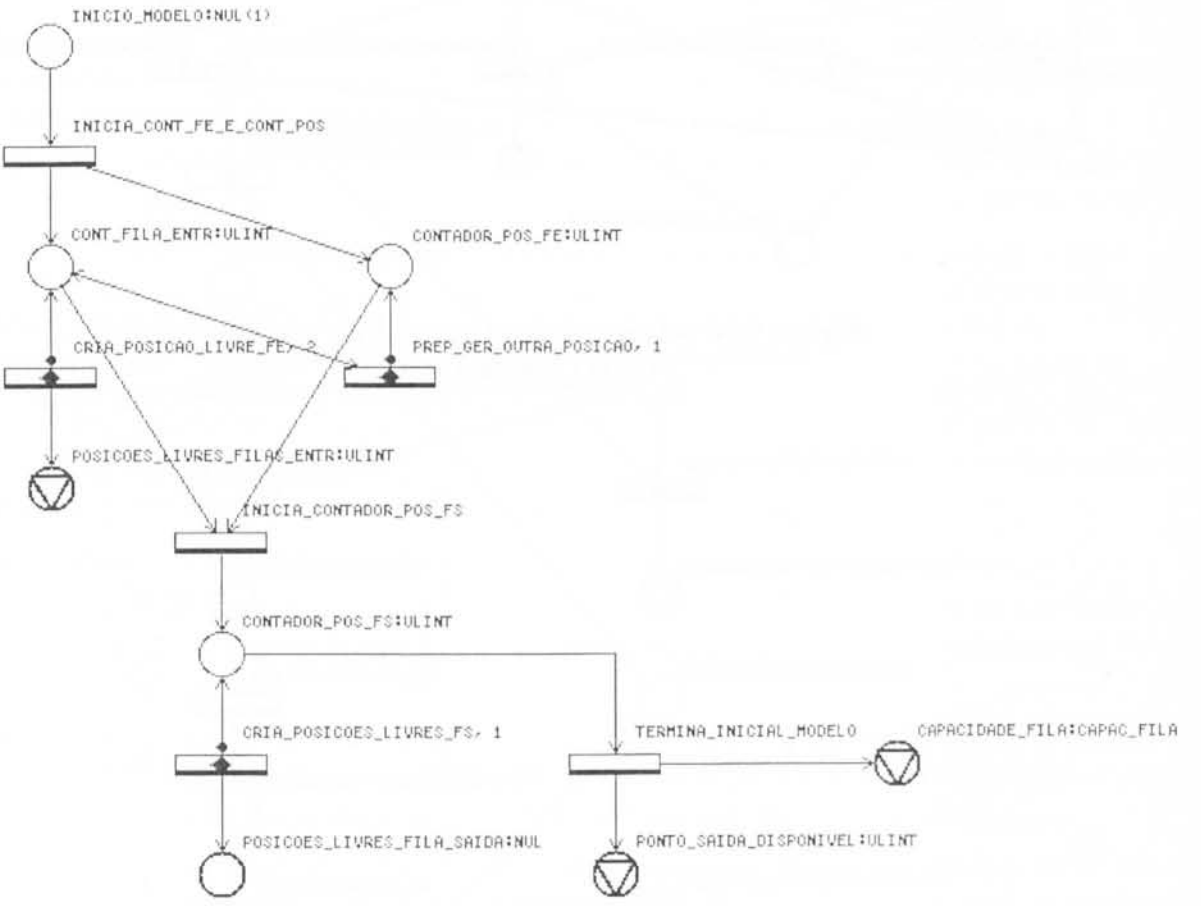


Figura 9.9 – Sub-rede 'inicializa_modelo' da classe POSTO_TRABALHO.

9.2.3.2 Sub-rede 'entrada_paletes' da classe POSTO_TRABALHO

A sub-rede 'entrada_paletes' está representada na Figura 9.10 (ver informação detalhada no anexo D.3.11). Esta representa a gestão de movimentos de descarga dos AGVs nos pontos de entrada, que é análoga à gestão de movimentos de carga dos AGVs, que foi descrita em 9.2.1 para a classe CENTRO_RECEPCAO.

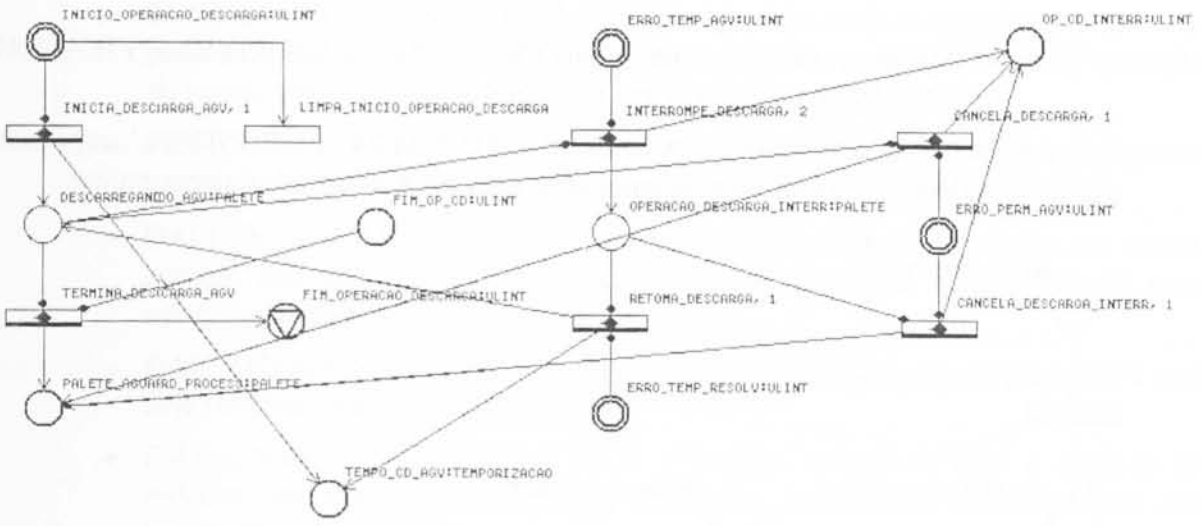


Figura 9.10 – Sub-rede 'entrada_paletes' da classe POSTO_TRABALHO.

9.2.3.3 Sub-rede 'processamento_paletes' da classe POSTO_TRABALHO

A sub-rede 'processamento_paletes' está representada na (ver informação detalhada no anexo D.3.13). Esta modeliza o processamento efectuado sobre as paletes retiradas dos *buffers* de entrada.

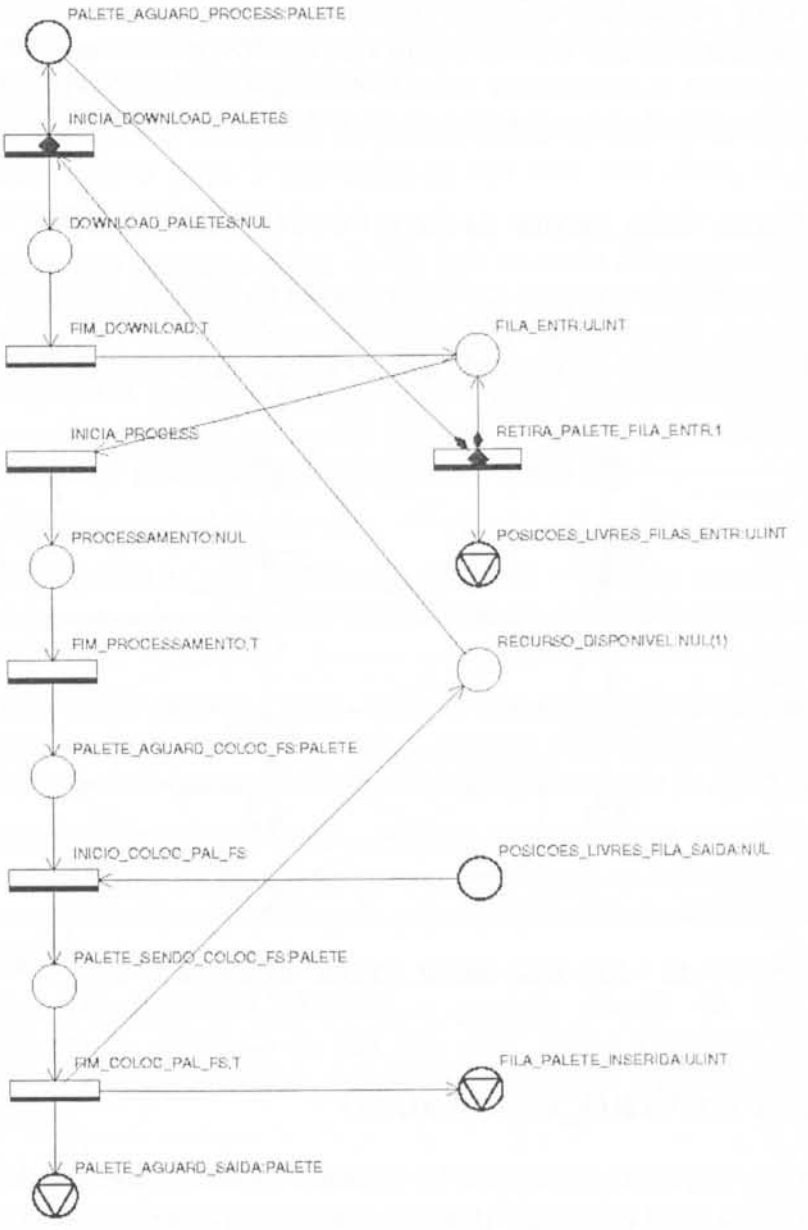


Figura 9.11 – Sub-rede 'processamento_paletes' da classe 'POSTO_TRABALHO'.

Uma operação de processamento consiste em:

- esperar que o recurso esteja livre, e que haja pelo menos uma palete em cada *buffer* de entrada (que haja um *token* relativo a cada fila de entrada, na posição 'PALETE_AGUARD_PROCESS');

- efectuar o *download* de uma palete de cada *buffer* de entrada, libertando uma posição em cada *buffer*;
- esperar pela realização das operações produtivas realizadas pela entidade (e.g. maquinação das peças, montagem, verificação da qualidade, etc);
- esperar que haja uma posição livre no *buffer* de saída, para ser colocada aí a palete resultante das operações de processamento;
- fazer o *download* da palete resultante das operações de processamento no *buffer* de saída, libertar o recurso para efectuar uma nova sequência de processamento e informar o exterior da colocação de uma palete na fila de saída, através da deposição de um *token* na posição 'PALETE_AGUARD_SAIDA'.

9.2.3.4 Sub-rede 'saida_paletes' da classe POSTO_TRABALHO

A sub-rede 'saida_paletes' está representada na Figura 9.12 (ver informação detalhada no anexo D.3.11). Esta representa a gestão de movimentos de carga dos AGVs no ponto de saída, que é análoga à que foi descrita em 9.2.1 para a classe CENTRO_RECEPCAO.

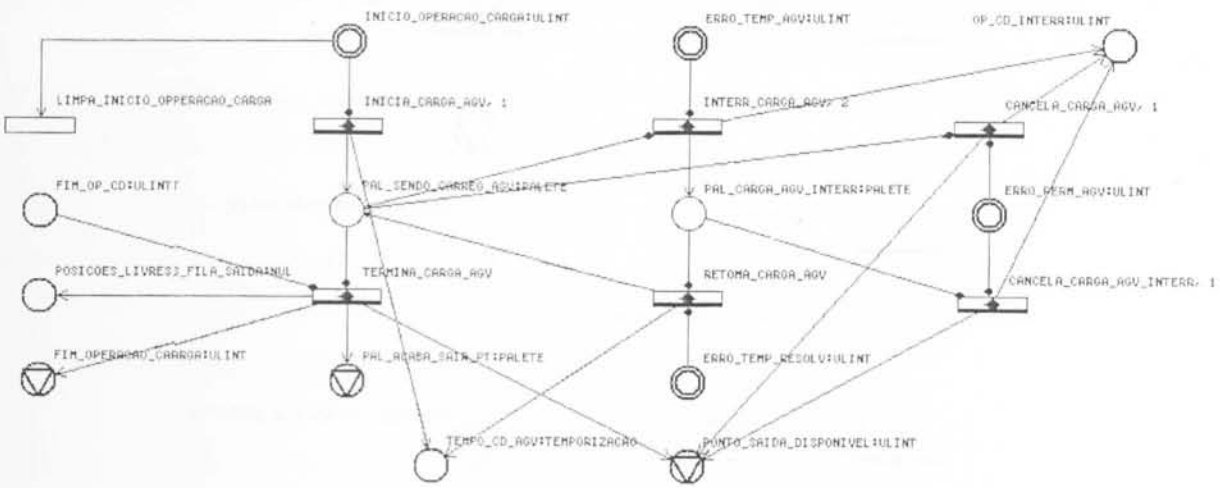


Figura 9.12 – Sub-rede 'saida_paletes' da classe POSTO_TRABALHO.

9.2.4 Classe CENTRO_EXPEDICAO

A classe CENTRO_EXPEDICAO pretende modelizar uma entidade de um sistema de fabrico, por onde saem paletes do sistema. Admite-se que as paletes são transportadas desde os pontos de descarga de AGVs, onde existem *buffers* com uma determinada capacidade, até a um cais de expedição, e que as paletes são expedidas em lotes com um número fixo de unidades.

A especificação desta entidade pressupõe:

- a existência de um número arbitrário de pontos de descarga, dotados de um *buffer* (fila de entrada FIFO), cuja capacidade é configurável e igual para todos os pontos;

- o tempo que demora a execução de uma operação de descarga de um AGV é configurável e descrito por uma FDP normal;
- o número de recursos disponíveis para o transporte das paletes, desde os pontos de descarga até ao cais, é configurável;
- o tempo que um recurso demora a transportar uma palete, desde um ponto de descarga até ao cais, é configurável e descrito por uma FDP normal;
- o tempo que medeia entre a expedição de dois lotes consecutivos é configurável e descrito por uma FDP exponencial;
- só é permitido o transporte de paletes desde os *buffers* de entrada até ao cais, quando houver lugar à expedição de um lote; fora disso, as paletes terão de permanecer nos *buffers*;
- o tempo que demora a carga de um lote no meio de transporte responsável pela sua expedição é configurável e descrito por uma FDP normal.

O anexo D.4 contém informação detalhada sobre a implementação da classe CENTRO_EXPEDICAO. A página principal da RP desta classe está representada na Figura 9.13 (ver informação detalhada no anexo D.4.10). O comportamento e configuração da entidade é definida pelos parâmetros apresentados no anexo D.4.4.

As posições de entrada definidas são:

- INICIO_OPERACAO_DESCARGA:ULINT – posição através da qual pode ser recebido um *token*, indicando a fila de entrada em que se iniciou a descarga de um AGV;
- ERRO_TEMP_AGV:ULINT – posição através da qual a entidade é notificada da ocorrência de um erro num AGV, a executar uma operação de descarga no ponto indicado pelo *token* recebido, implicando a interrupção da operação;
- ERRO_PERM_AGV:ULINT – posição através da qual a entidade é notificada da ocorrência de um erro permanente num AGV, a executar uma operação de descarga no ponto indicado pelo *token*, implicando o cancelamento definitivo da operação;
- ERRO_TEMP_RESOLV:ULINT – posição através da qual a entidade é notificada da resolução de um erro num AGV, a executar uma operação de descarga no ponto indicado pelo *token*, permitindo a continuação da execução da operação.

As posições de saída definidas são:

- FIM_OPERACAO_DESCARGA:ULINT – posição onde é depositado um *token*, indicando a fila de entrada em que uma operação de descarga terminou;
- POSICOES_LIVRES_FILAS:ULINT – posição onde pode ser depositado um *token*, indicando uma fila de entrada em que há uma posição disponível.

Os objectos do tipo 'TEMPORIZADOR', que fazem parte da RP da entidade, estão descritos no anexo D.4.10.1.

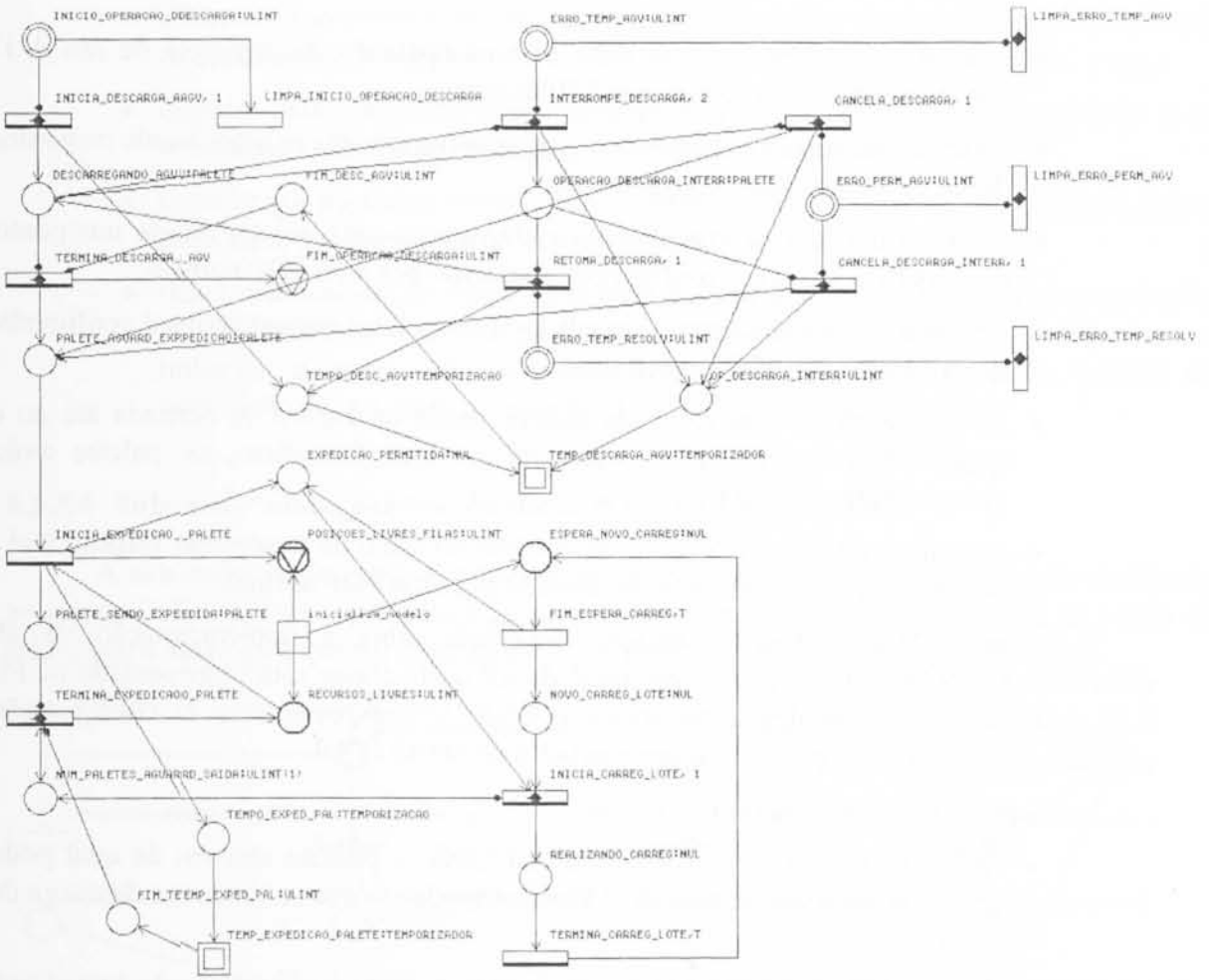


Figura 9.13 – RP da página principal da classe CENTRO_EXPEDICAO.

A partee superior da RP da página principal representa a gestão de movimentos de descarga dos AGVs nos pontos de entrada, que é análoga à gestão de movimentos de carga dos AGVs, quue foi descrita em 9.2.1 para a classe CENTRO_RECEPCAO.

A partee inferior representa a gestão da expedição dos lotes de paletes.

Quandco é decidida a expedição de um lote de paletes, é disparada a transição 'FIM_ESPERA_CARREG', sendo colocado um *token* em 'EXPEDICAO_PERMITIDA'. Enquanto estaa posição estiver marcada, as transições 'INICIA_EXPEDICAO_PALETE' e 'TERMINA_EXPEDICAO_PALETE' vão sendo disparadas alternadamente, desde que os *buffers* de entrada contenham paletes aí depositadas. O *token* existente na posição 'NUM_PALETTES_AGUARD_SAIDA' vai sendo actualizado com o número de paletes existentes no ccais de expedição.

Quandco o número de paletes aí existentes atingir o número de paletes de um lote, a transição 'INIICIA_CARREG_LOTE' dispara. Esta impede a expedição de mais paletes, através da remoção do *token* existente em 'EXPEDICAO_PERMITIDA', subtrai ao valor guardado no *ttoken* da posição 'NUM_:PALETES_AGUARD_SAIDA' o número de paletes despachadas no lote, e configura o tempo de disparo da transição 'TERMINA_CARREG_LOTE'. Este tempo simula o tempo de carga do lote no meio de transporte resjponsável pela sua expedição. A expedição termina com o disparo da transição

'TERMINA_CARREG_LOTE', sendo planeada a expedição de um novo lote, através do tempo de disparo da transição 'FIM_ESPERA_CARREG'.

9.2.4.1 Sub-rede 'inicializa_modelo' da classe CENTRO_EXPEDICAO

A sub-rede 'inicializa_modelo' está representada na Figura 9.14 (ver informação detalhada no anexo D.4.11). Esta representa a configuração do modelo.

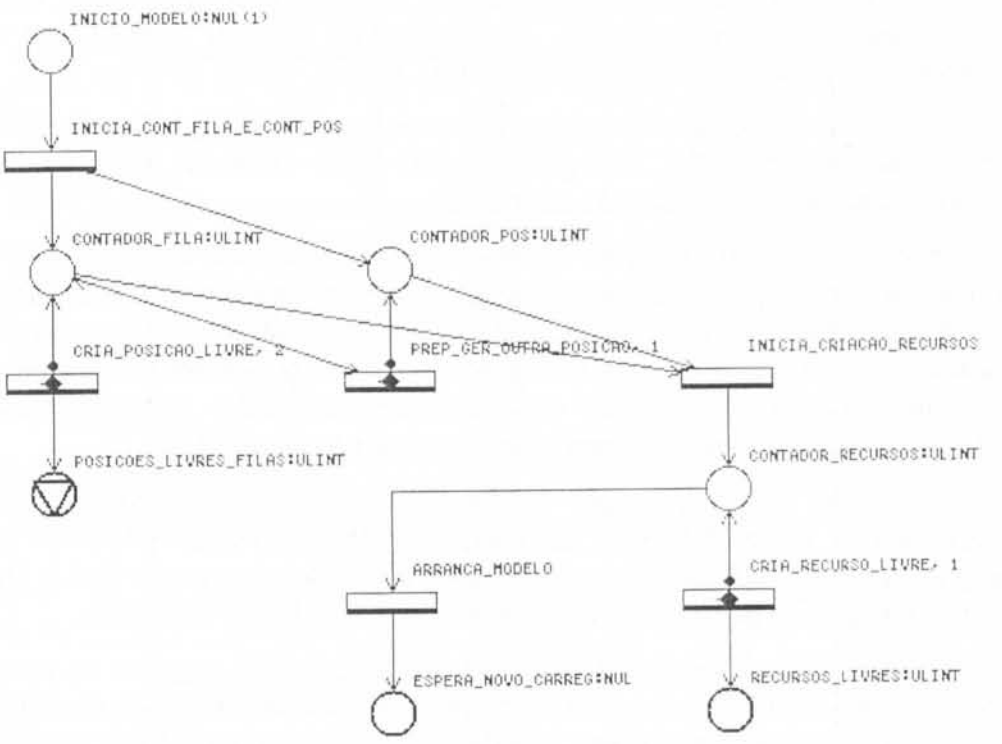


Figura 9.14 – Sub-rede 'inicializa_modelo' da classe CENTRO_EXPEDICAO.

Os parâmetros que influenciam a configuração são:

- fila1,
- filan,
- capacidade_filas e
- num_recursos.

A configuração consiste em:

- depositar *tokens* na posição 'POSICOES_LIVRES_FILAS', representando posições livres nos *buffers* das filas de entrada;
- depositar *tokens* na posição 'RECURSOS_LIVRES', representando recursos disponíveis para o transporte de paletes, desde os *buffers* de entrada até ao cais de expedição;

- depositar um *token* na posição 'ESPERA_NOVO_CARREG', e planear a expedição do primeiro lote, através da configuração do tempo de disparo da transição 'FIM_ESPERA_CARREG' da página principal.

9.3 Modelização do sistema de fabrico (Classe SISTEMA_MANUFACTURA)

Com base na definição das classes dos diferentes tipos de equipamentos que constituem o sistema de fabrico ilustrado pela Figura 9.1 da página 170, construiu-se um modelo global do comportamento do sistema de fabrico, servido por um SAGVs, representado pela classe SISTEMA_MANUFACTURA.

O modelo foi construído tomando como base as informações de encaminhamento apresentadas na Tabela 9.1. O significado das siglas utilizadas pode ser esclarecido no glossário ou na descrição efectuada em 9.1.

Pode-se observar que, para qualquer célula do sistema de fabrico, há sempre dois destinos alternativos para uma palete, que seja necessário encaminhar. A escolha de um ou de outro destino baseia-se nas probabilidades definidas. Note-se que com as probabilidades indicadas na tabela, uma palete que saia do CM4 é sempre encaminhada para a CM, e uma palete que saia do AAPA é sempre encaminhada para o CEPA. As probabilidades indicadas são as que foram utilizadas nas simulações descritas no capítulo 10.

A RP da página principal da RP da classe SISTEMA_MANUFACTURA está representada na Figura 9.15 (ver informação detalhada no anexo D.5). Os parâmetros de configuração de um objecto desta classe estão apresentados no anexo D.5.4. Quase todos os eles dizem respeito à definição das probabilidades indicadas na Tabela 9.1.

A lista de posições de entrada é um subconjunto das posições de entrada da classe POSTO_TRABALHO, contudo dizem respeito a todas as entidades da planta fabril. Por exemplo, quando a posição de entrada 'INICIO_OPERACAO_CARGA' recebe um *token* do exterior, todas as posições de entrada com o mesmo nome das entidades representadas por objectos do tipo 'ARMAZEM_AUTOMATICO', 'POSTO_TRABALHO' e 'CENTRO_EXPEDICAO' recebem uma cópia desse *token*, uma vez que essa posição de entrada está ligada às posições de entrada de todos esses objectos. O mesmo se passa em relação a todas as outras posições de entrada. A lista completa das posições de entrada é apresentada no anexo D.5.2.

No caso das posições de saída, o processo é inverso. Por exemplo, todos os *tokens* a informarem sobre o fim de operações de carga, de todos os objectos do sistema de fabrico onde possam ocorrer operações de carga, são canalizados para o exterior através de uma única posição de saída: 'FIM_OPERACAO_CARGA'. A lista completa de posições de saída é apresentada no anexo D.5.3.

Tabela 9.1 – Definição do caminho tecnológico das paletes do sistema de fabrico modelizado.

Origem	Destino	Probabilidade
CRMP	AAMP	0.90
	CM1	0.10
AAMP	CM1	0.66
	CM2	0.33
CM1	PM1	0.50
	PM2	0.50
CM2	PM3	0.50
	PM4	0.50
PM1	PM3	0.50
	CM3	0.50
PM2	PM4	0.50
	CM3	0.50
PM3	PM5	0.50
	PM6	0.50
PM4	PM5	0.50
	PM6	0.50
PM5	CM4	0.50
	CM	0.50
PM6	CM4	0.50
	CM	0.50
CM3	PM5	0.50
	PM6	0.50
CM4	CM	1.00
	CM2	0.00
CM	CQ	0.75
	AAPA	0.25
CQ	AAPA	0.90
	CEPA	0.10
AAPA	CQ	0.00
	CEPA	1.00

A Figura 9.16 ilustra os objectos instanciados no modelo do sistema de fabrico. Observa-se que cada célula da planta fabril foi representada por um objecto de uma classe, correspondente ao tipo de equipamento em causa, e de acordo com a identificação feita em 9.1.

Para todos as células de fabrico que contêm pontos de saída de paletes (todas, excepto o CEPA), foi necessário modelizar o encaminhamento dessas paletes, de acordo com as probabilidades especificadas na Tabela 9.1. Por isso, a RP da Figura 9.15 é constituída por várias sub-redes (denominadas 'Controlo_...').

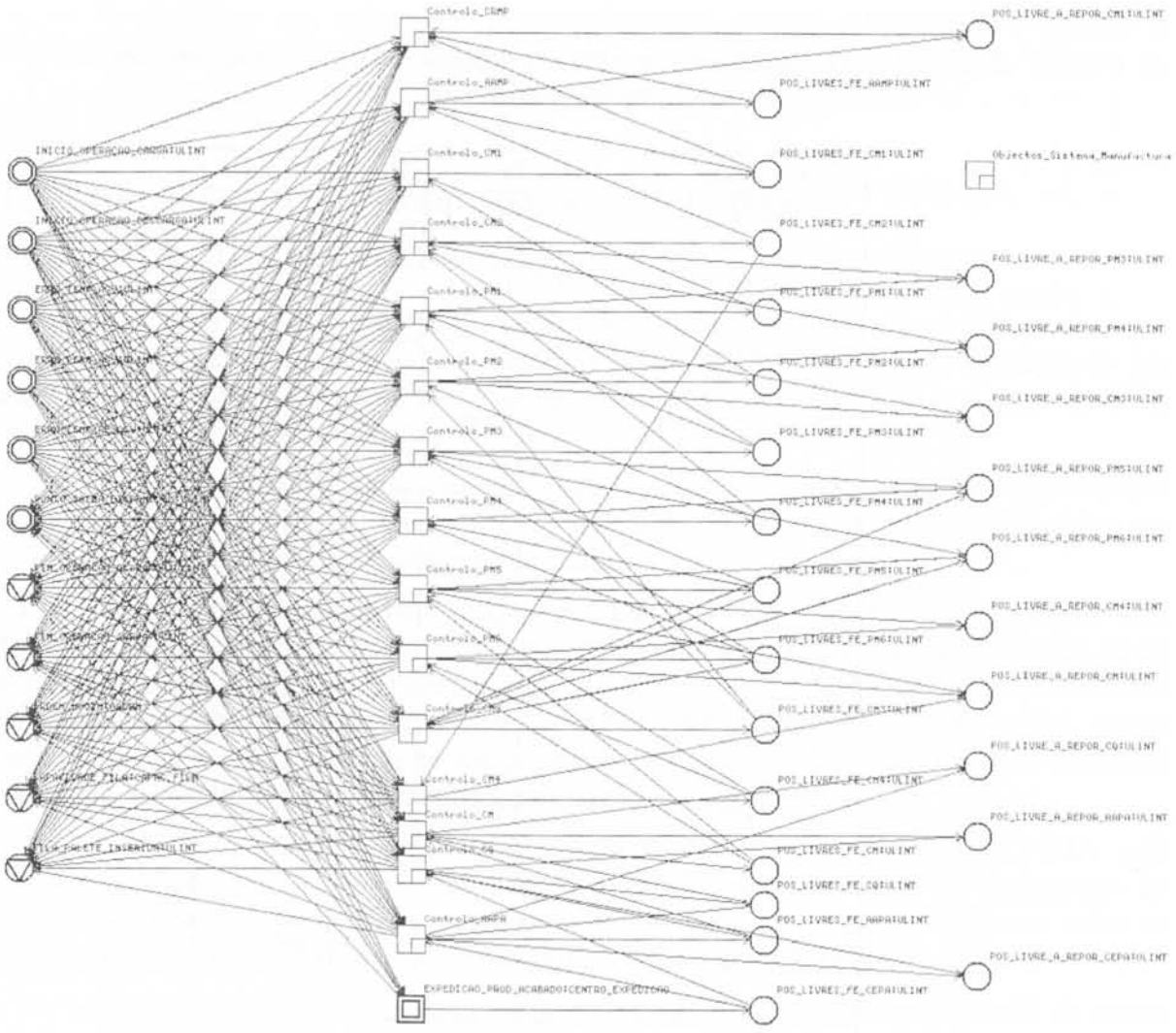


Figura 9.15 – RP da página principal da classe SISTEMA_MANUFACTURA.

O processamento associado a cada uma dessas sub-redes é muito parecido em todas elas, sofrendo apenas pequenas variações devidas às diferentes entidades envolvidas. Por essa razão, na secção seguinte é descrita a implementação de apenas uma delas. Para as outras, apenas serão apresentadas as figuras com as RP correspondentes.



Figura 9.16 – Objectos instanciados na RP da classe SISTEMA_MANUFACTURA.

9.3.1 Descrição da sub-rede 'Controlo_CRMP' da classe SISTEMA_MANUFACTURA

A RP que controla a célula CRMP está apresentada na Figura 9.17. Esta evidencia a integração do objecto 'RECEPCAO_MAT_PRIMAS' no modelo do sistema de fabrico e a sua interacção com a sub-rede 'encam_pal_CRMP_AAMP_ou_CM1', que modeliza o encaminhamento das paletes do CRMP para o AAMP ou para o CM1. Esta sub-rede está apresentada na Figura 9.18 (ver informação detalhada no anexo D.5.18).

Se houver pelo menos uma posição livre nas filas de entrada do AAMP e na fila de entrada de CM1, a transição 'DECIDE_CRMP_AAMP_OU_CM1' decide o destino da próxima palete que sair do CRMP, com base na probabilidade 'p_CRMP_AAMP_CM1' (ver parâmetros da classe SISTEMA_MANUFACTURA no anexo D.5.4). Nos outros casos, as transições 'DECIDE_CRMP_AAMP' e 'DECIDE_CRMP_CM1' permitem encaminhar a próxima palete para o AAMP ou para o CM1, respectivamente, se num dos destinos possíveis não existirem posições livres nas filas de entrada. O *token* depositado em 'DESTINO_PROX_PAL_CRMP' indica a fila de entrada (do AAMP ou do CM1) para onde será encaminhada a próxima palete que sair do CRMP.

As transições 'REPOE_POS_LIVRE_...' permitem repor na posição 'POS_LIVRES_FE_...' respectiva o *token* correspondente à posição de uma fila de entrada não utilizada, na sequência da decisão resultante do disparo da transição 'DECIDE_CRMP_AAMP_OU_CM1'.

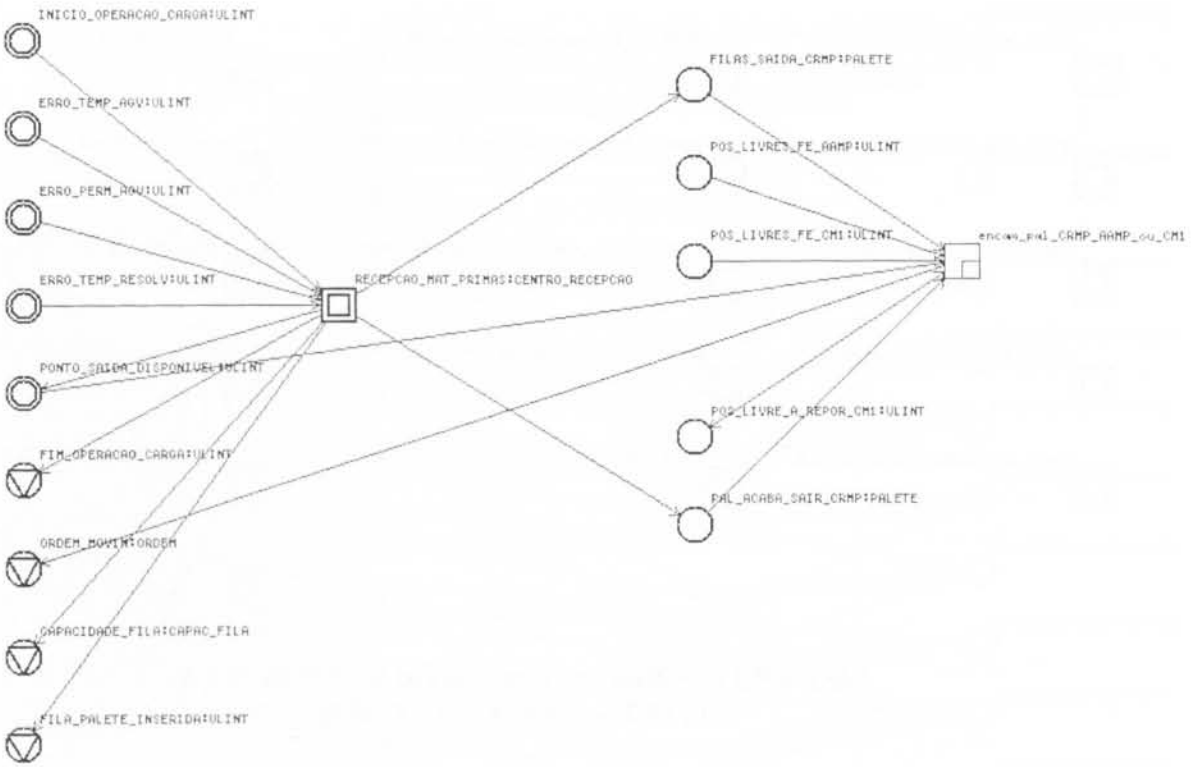


Figura 9.17 – Sub-rede 'Controlo_CRMP' da classe SISTEMA_MANUFACTURA.

Quando houver pelo menos uma palete numa das filas de saída do CRMP, representada por um *token* em 'FILAS_SAIDA_CRMP', e se o ponto de saída, correspondente ao destino decidido para a próxima palete a sair do CRMP, estiver disponível, a transição 'ENVIA_ORDEM_SAIDA_CRMP' é disparada. Esta deposita um *token* em 'ORDEM_MOVIM' e em 'PAL_RETIDAS_SAIDA_CRMP'. O primeiro desencadeia o lançamento de uma tarefa de movimentação no SGM (camada superior de gestão do SAGVs). O segundo poderá vir a ser utilizado para lançar uma nova ordem de movimentação, através do disparo da transição 'ENVIA_ORDEM_PAL_RET_SAIDA_CRMP', se primeira tarefa for cancelada pelo SAGVs (e.g. devido a um erro no AGV).

Quando a palete sair do CRMP (após o fim da carga do AGV), a posição 'PAL_ACABA_SAIR_CRMP' recebe um *token* correspondente a essa palete, provocando o disparo da transição 'LIMPA_PAL_RETIDA_SAIDA_CRMP'. Esta consome o *token* da posição 'PAL_RETIDAS_SAIDA_CRMP', que representava o facto da palete estar retida numa fila de saída do CRMP.

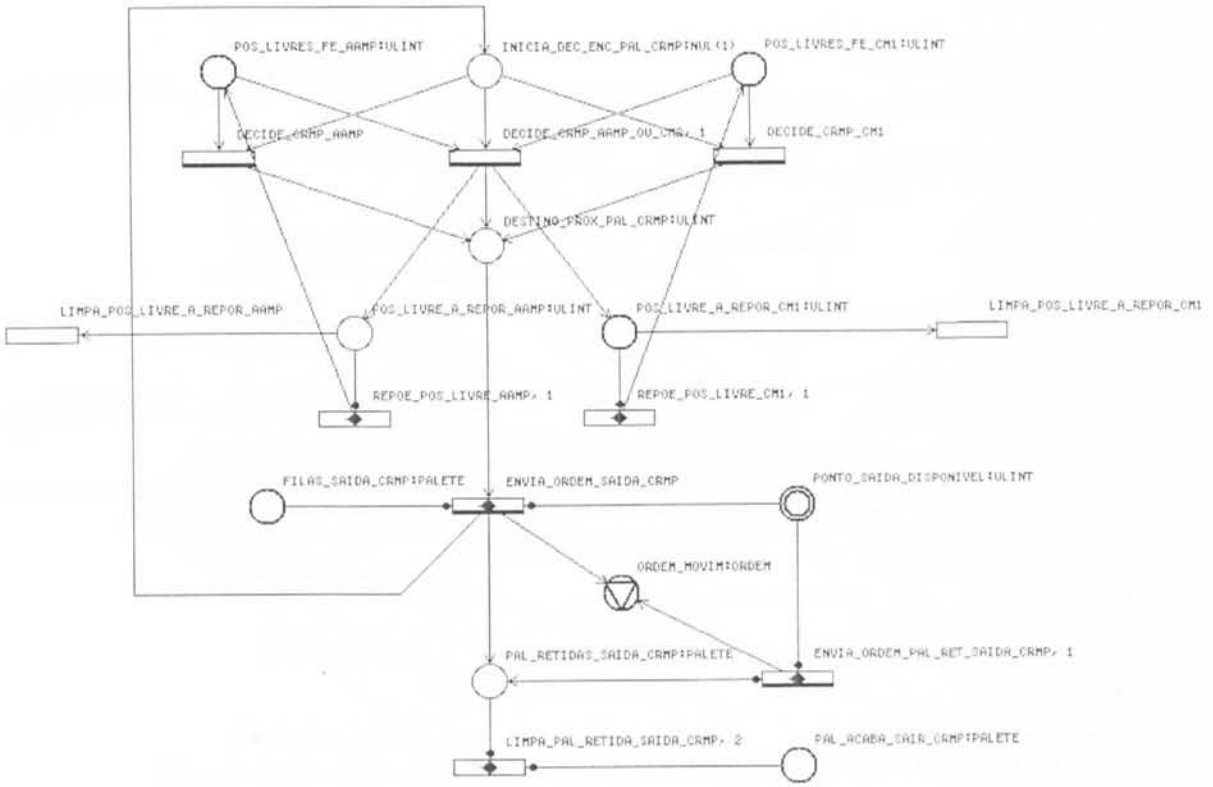


Figura 9.18 – Sub-rede 'encam_pal_CRMP_AAMP_ou_CM1 da classe SISTEMA_MANUFACTURA.

9.3.2 Figuras com as restantes sub-redes da classe SISTEMA_MANUFACTURA

Nesta secção são apresentadas as figuras das outras sub-redes denominadas 'Controlo_...'.

9.3.2.1 Sub-rede 'Controlo_AAMP' da classe SISTEMA_MANUFACTURA

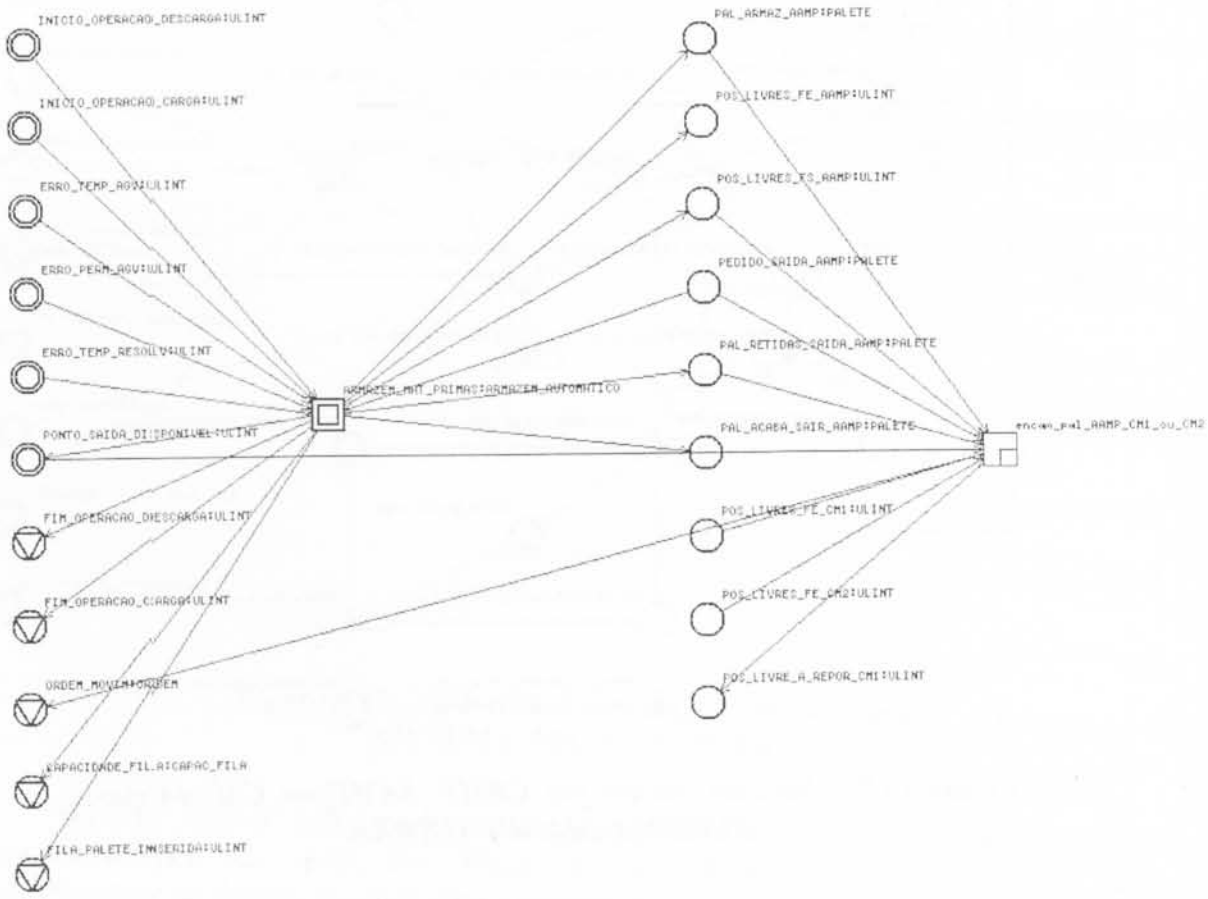


Figura 9.19 – Sub-rede 'Controlo_AAMP' da classe SISTEMA MANUFACTURA.

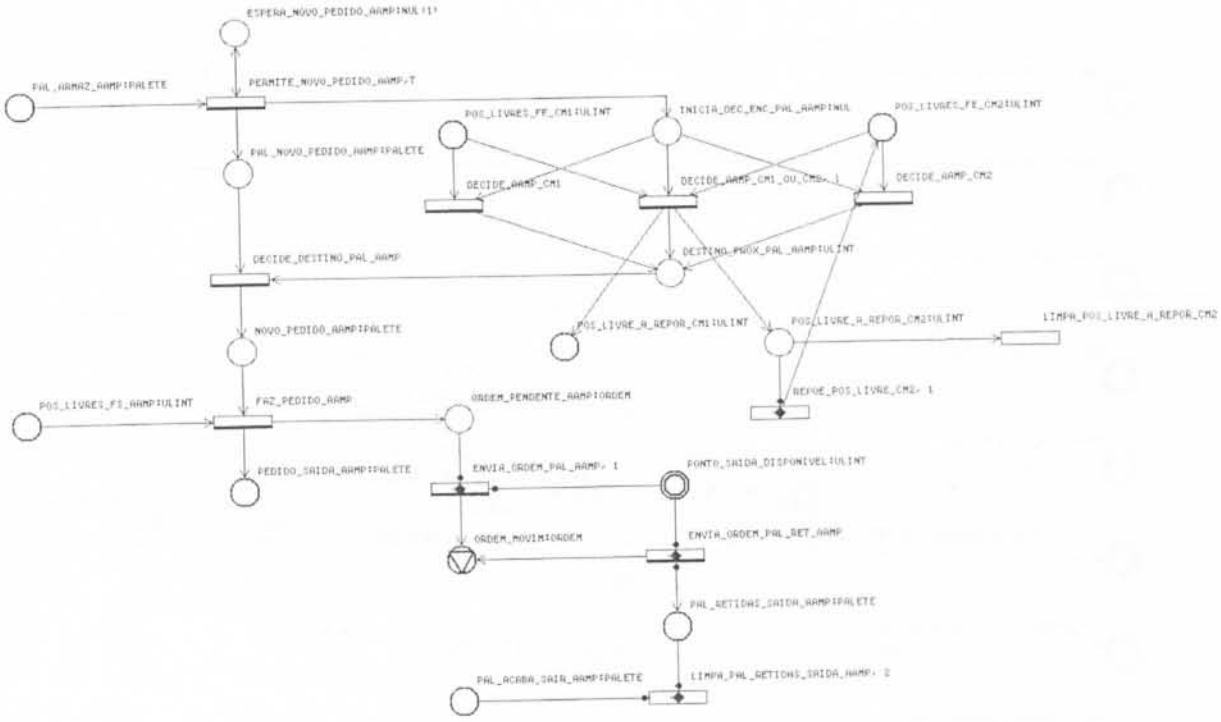


Figura 9.20 – Sub-rede 'encam_pal_AAMP_CM1_ou_CM2' da classe SISTEMA MANUFACTURA.

9.3.2.2 Sub-rede 'Controlo_CM1' da classe SISTEMA_MANUFACTURA

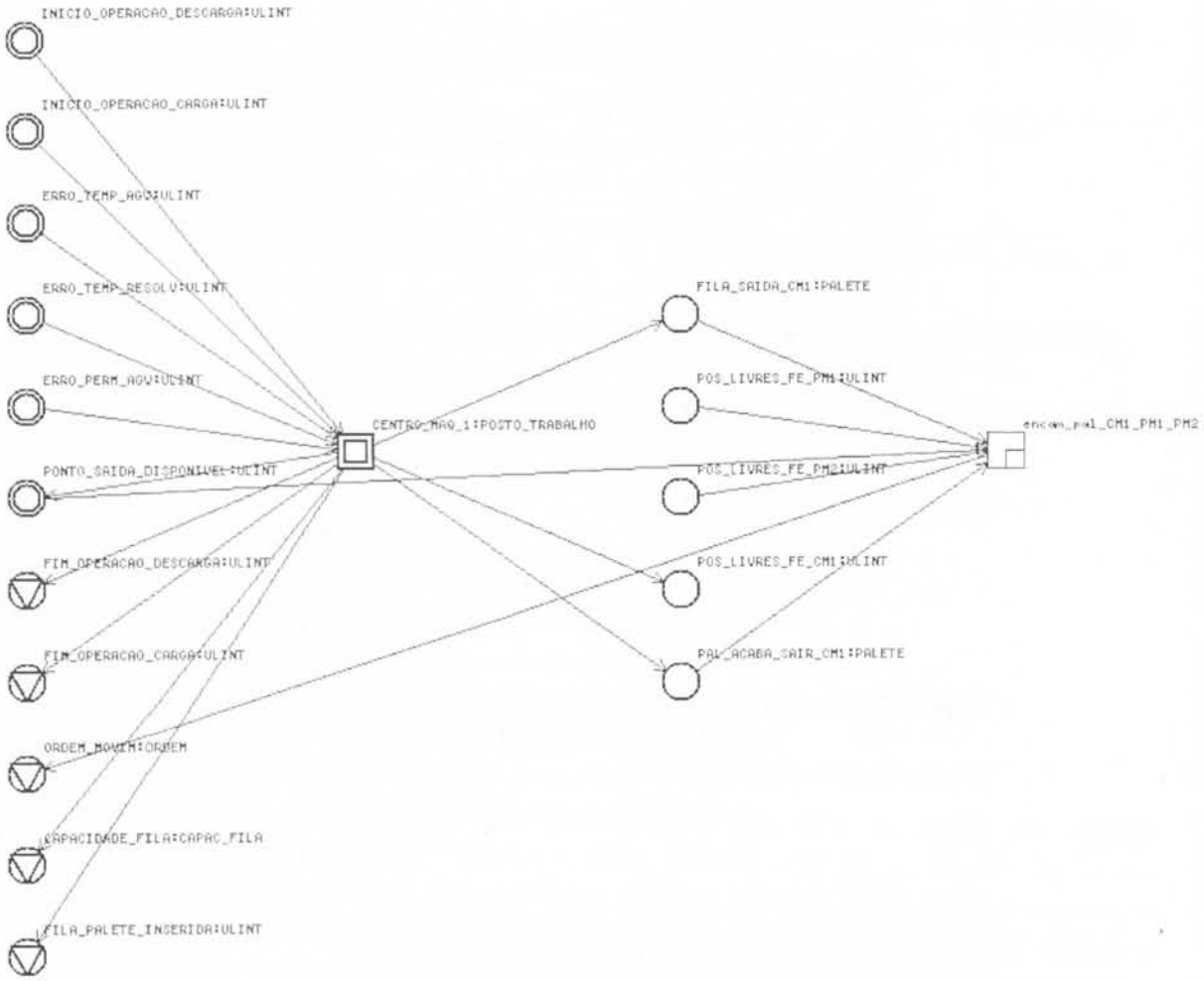


Figura 9.21 – Sub-rede 'Controlo_CM1' da classe SISTEMA_MANUFACTURA.

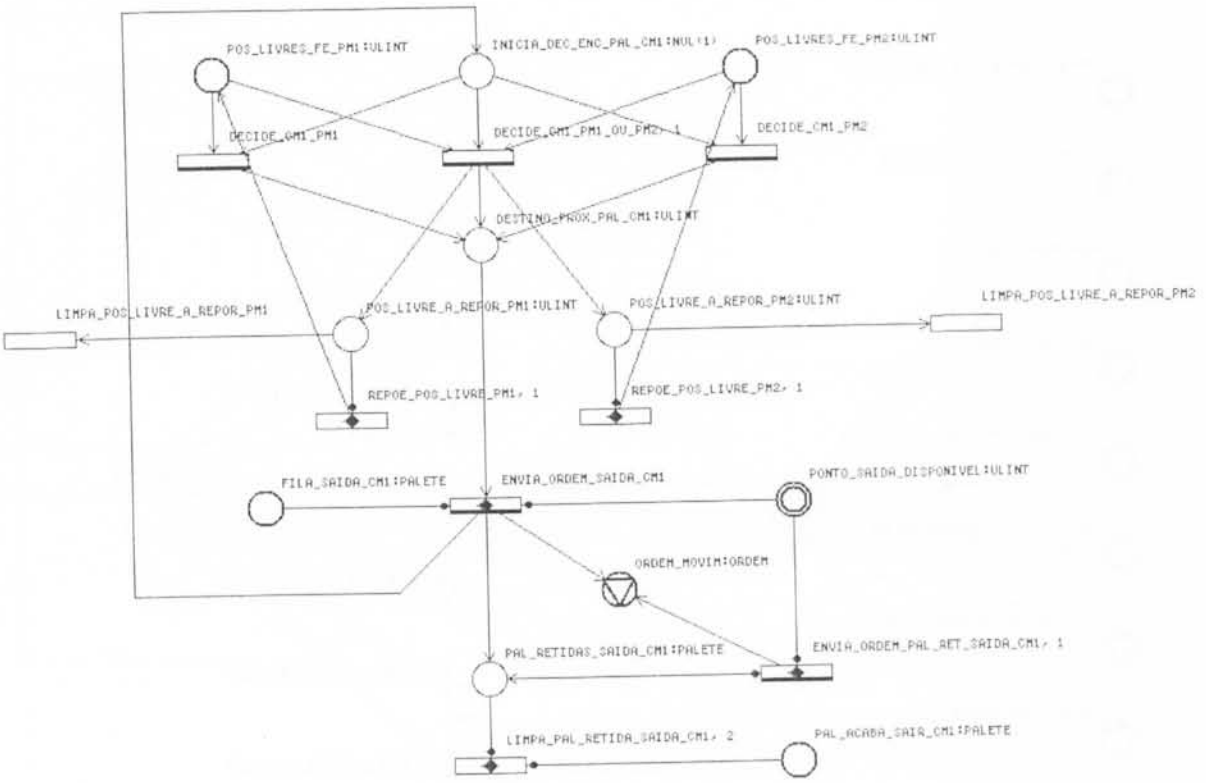


Figura 9.22 – Sub-rede 'encam_pal_CM1_PM1_PM2' da classe SISTEMA_MANUFACTURA.

BIBLIOTECA F.E. PORTO

9.3.2.3 Sub-rede 'Controlo_CM2' da classe SISTEMA_MANUFACTURA

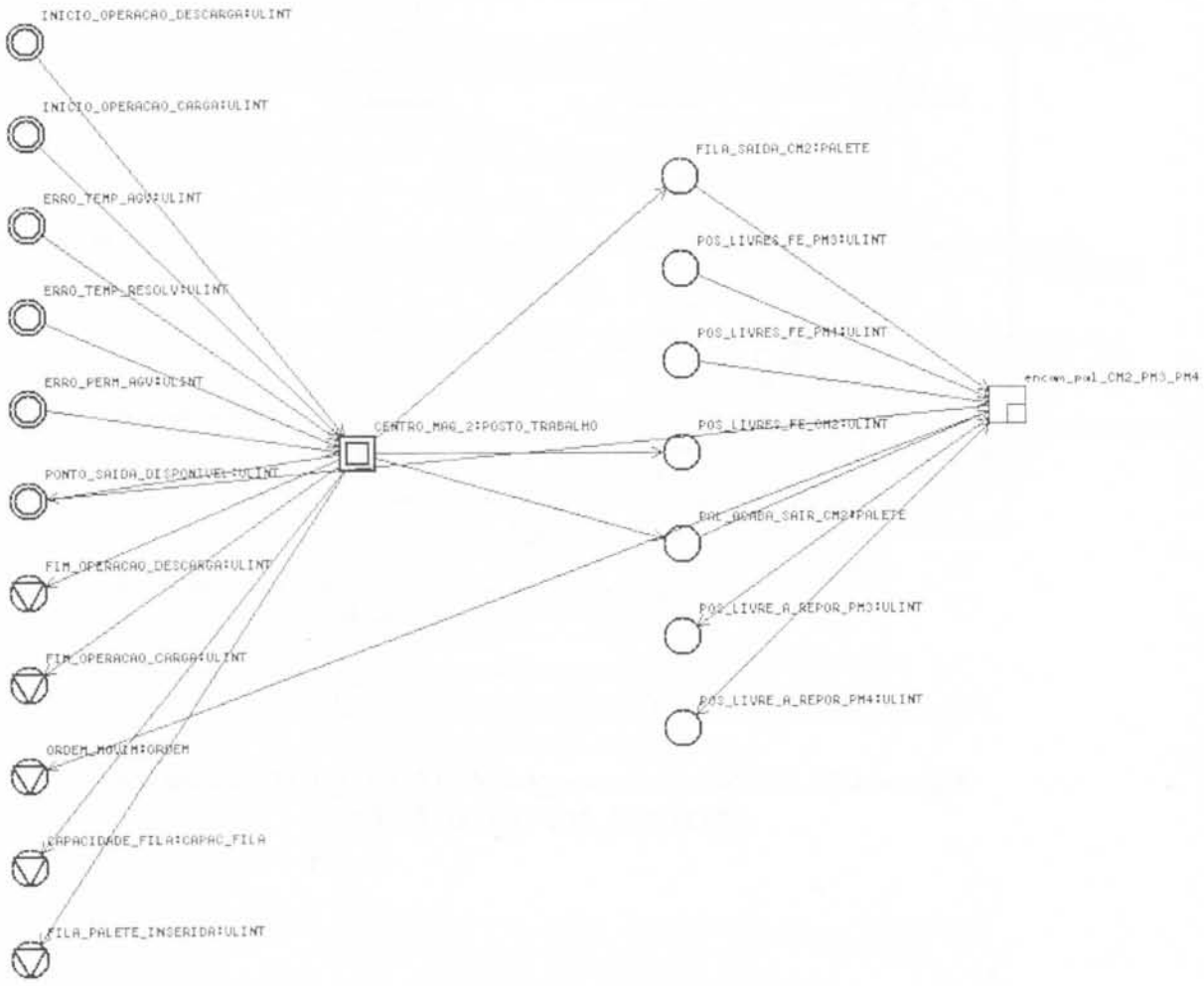


Figura 9.23 – Sub-rede 'Controlo_CM2' da classe SISTEMA_MANUFACTURA.

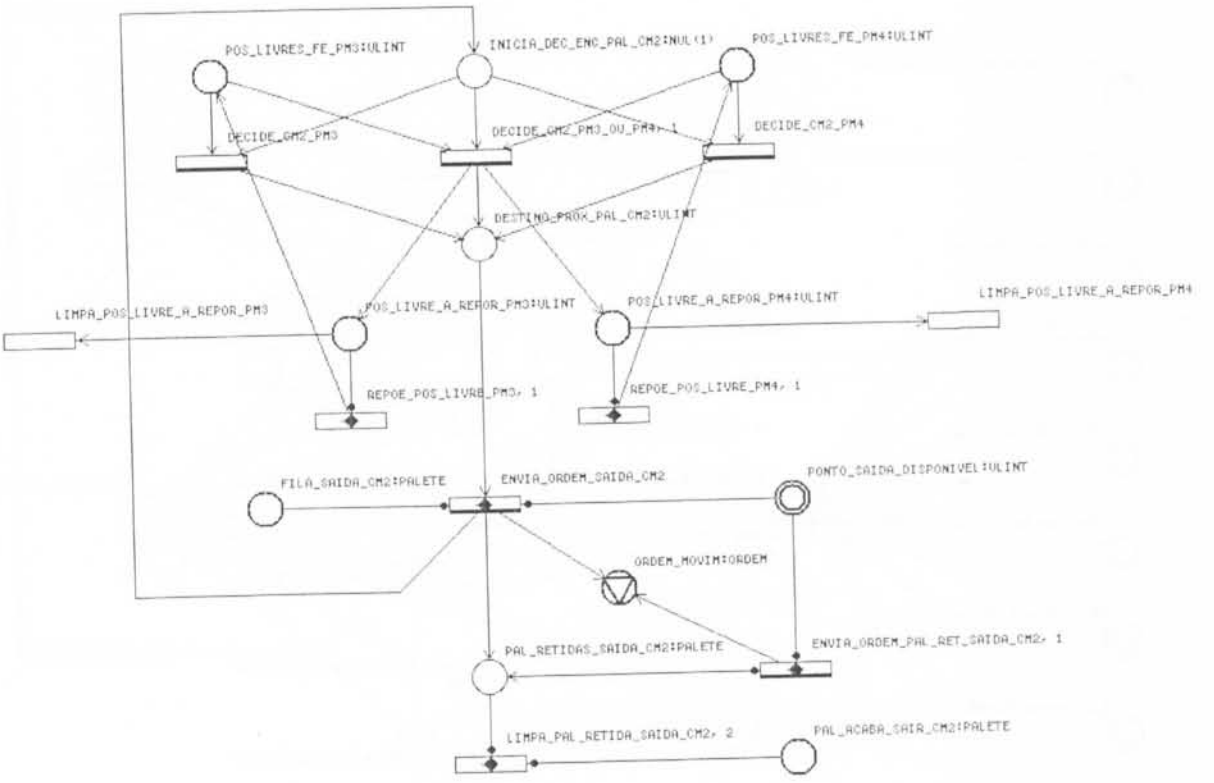


Figura 9.24 – Sub-rede 'encam_pal_CM2_PM3_PM4' da classe SISTEMA_MANUFACTURA.

9.3.2.4 Sub-rede 'Controlo_PM1' da classe SISTEMA_MANUFACTURA

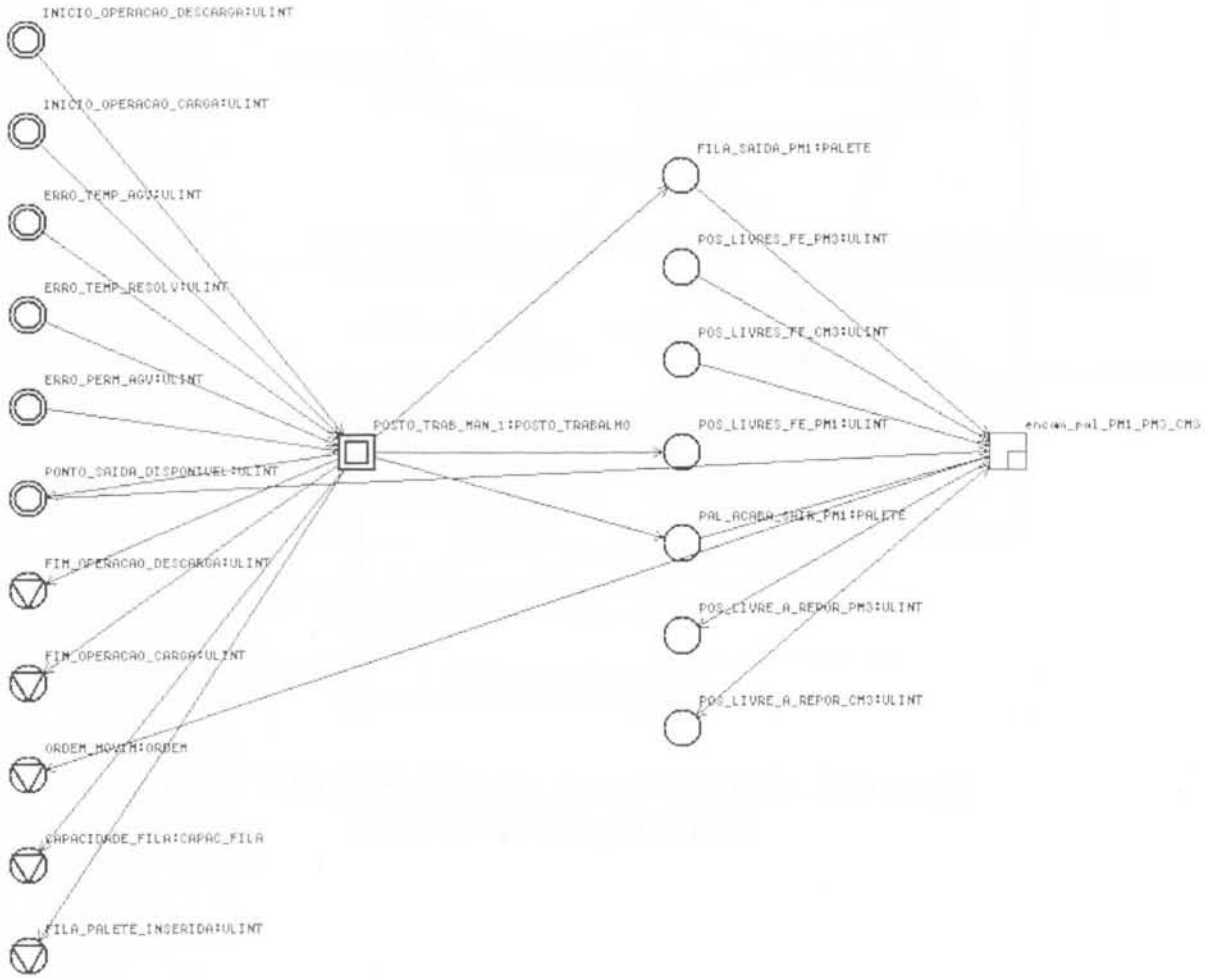


Figura 9.25 – Sub-rede 'Controlo_PM1' da classe SISTEMA_MANUFACTURA.

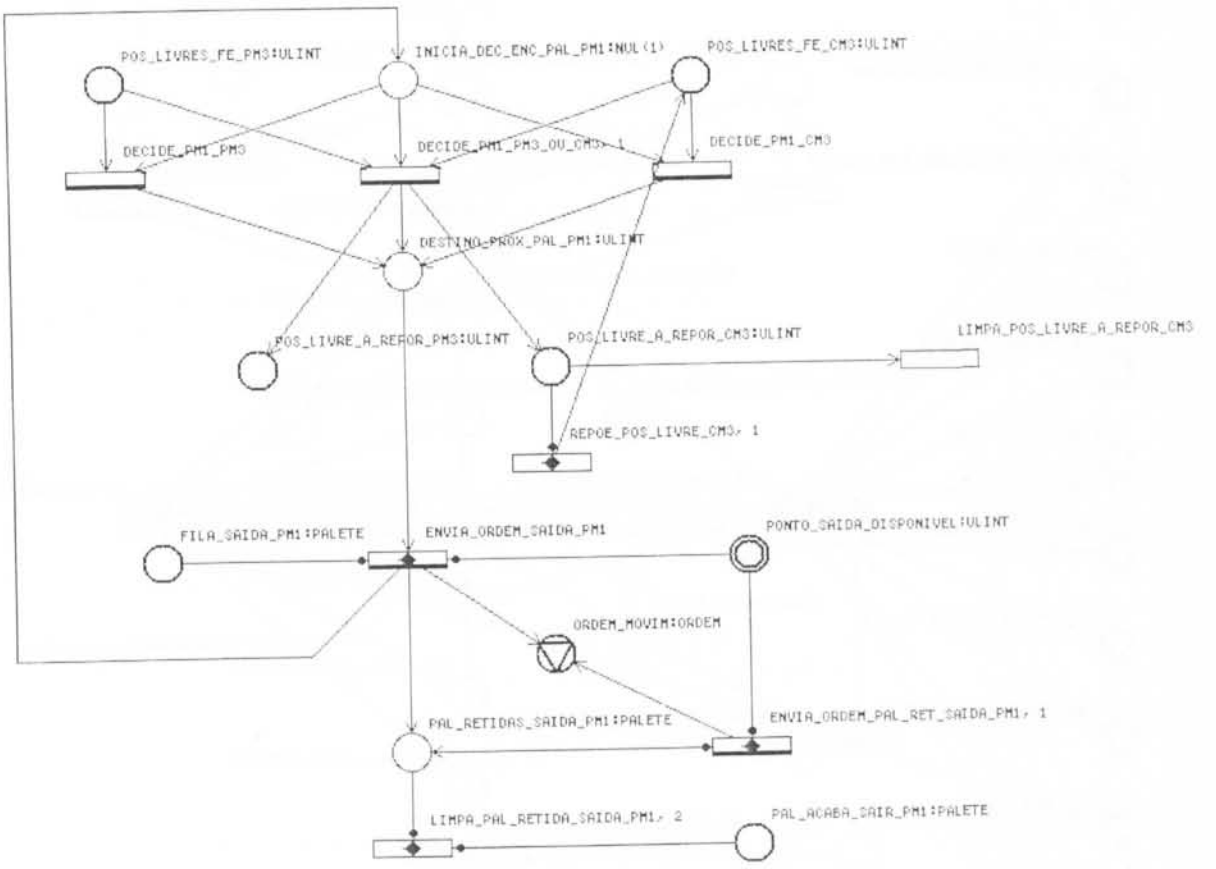


Figura 9.26 – Sub-rede 'encam_pal_PM1_PM3_CM3' da classe SISTEMA_MANUFACTURA.

9.3.2.5 Sub-rede 'Controlo_PM2' da classe SISTEMA_MANUFACTURA

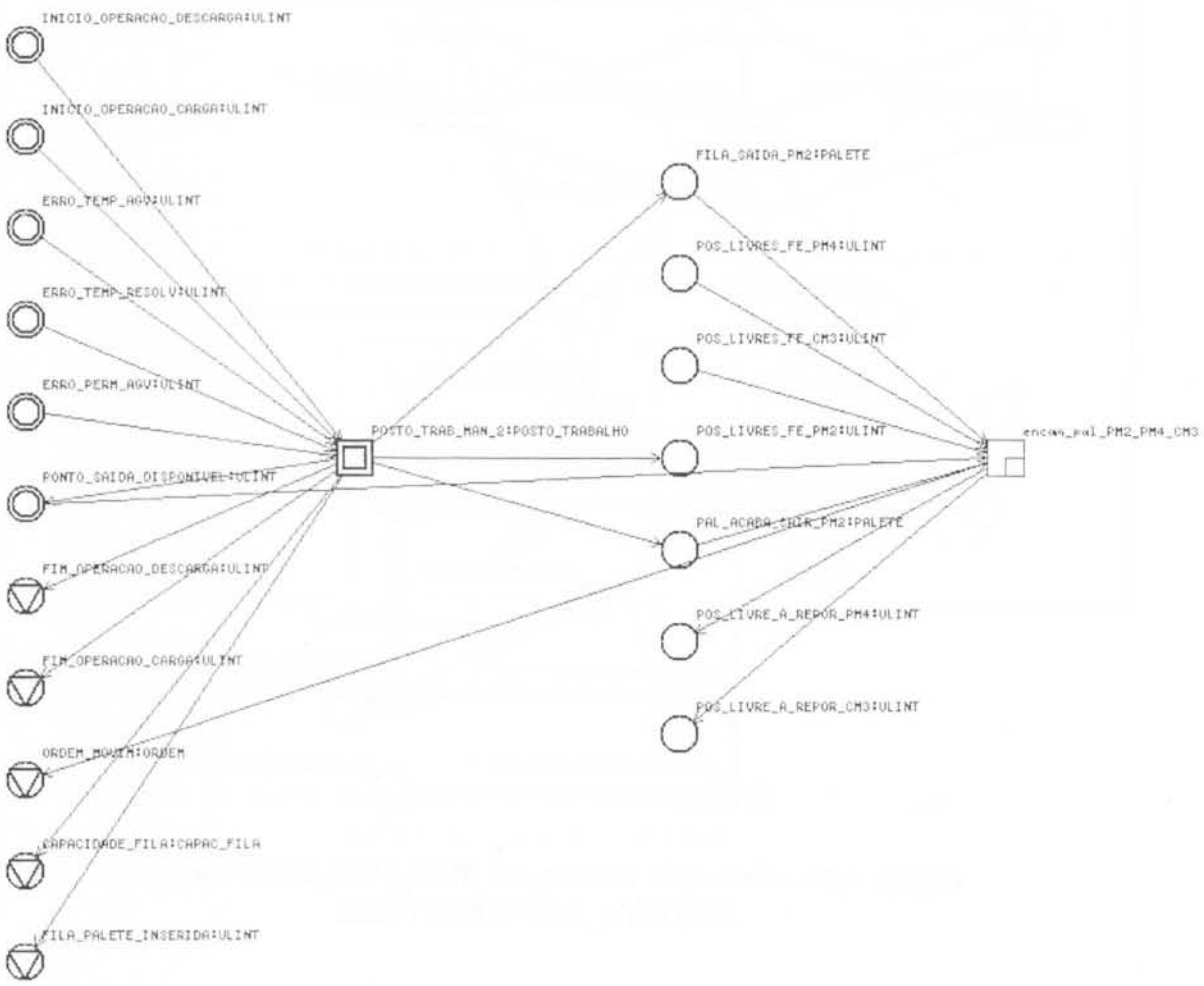


Figura 9.27 – Sub-rede 'Controlo_PM2' da classe SISTEMA_MANUFACTURA.

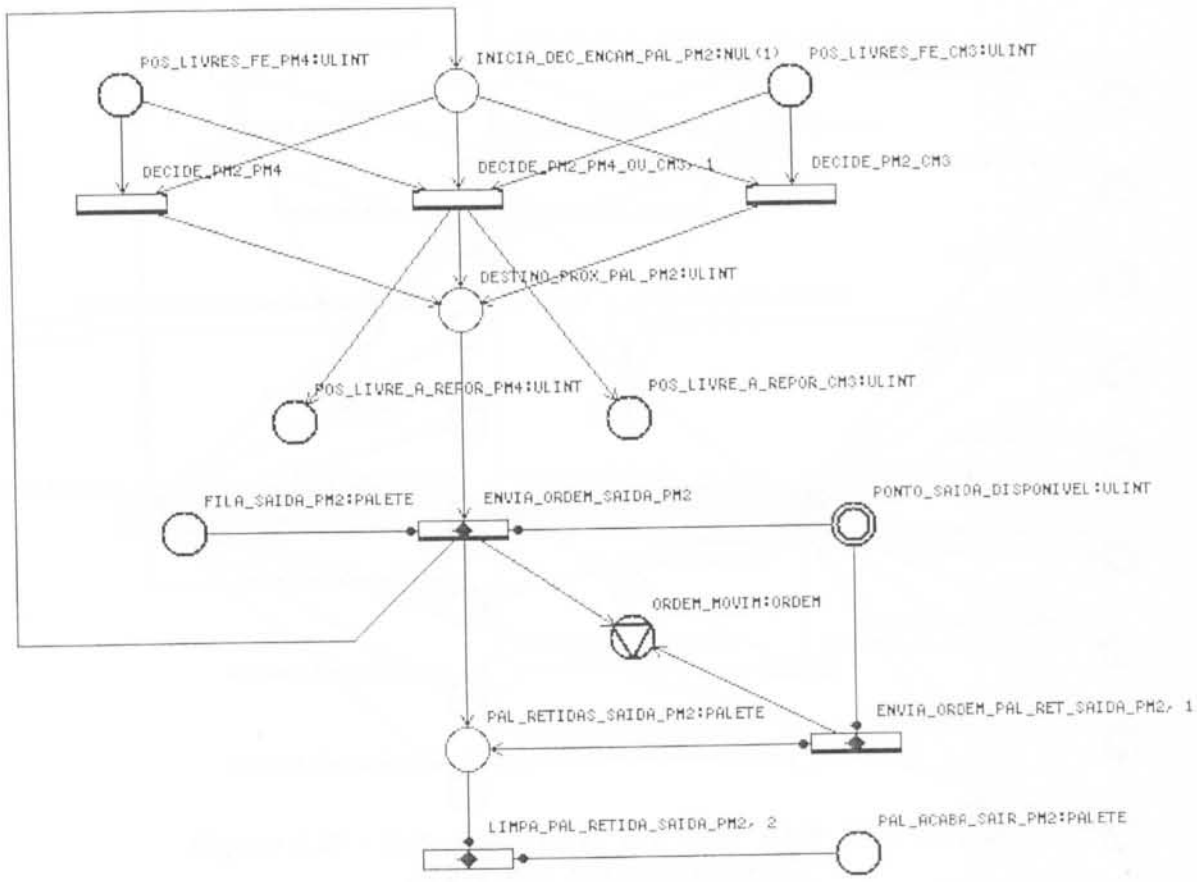


Figura 9.28 – Sub-rede 'encam_pal_PM2_PM4_CM3' da classe SISTEMA_MANUFACTURA.

9.3.2.6 Sub-rede 'Controlo_PM3' da classe SISTEMA_MANUFACTURA

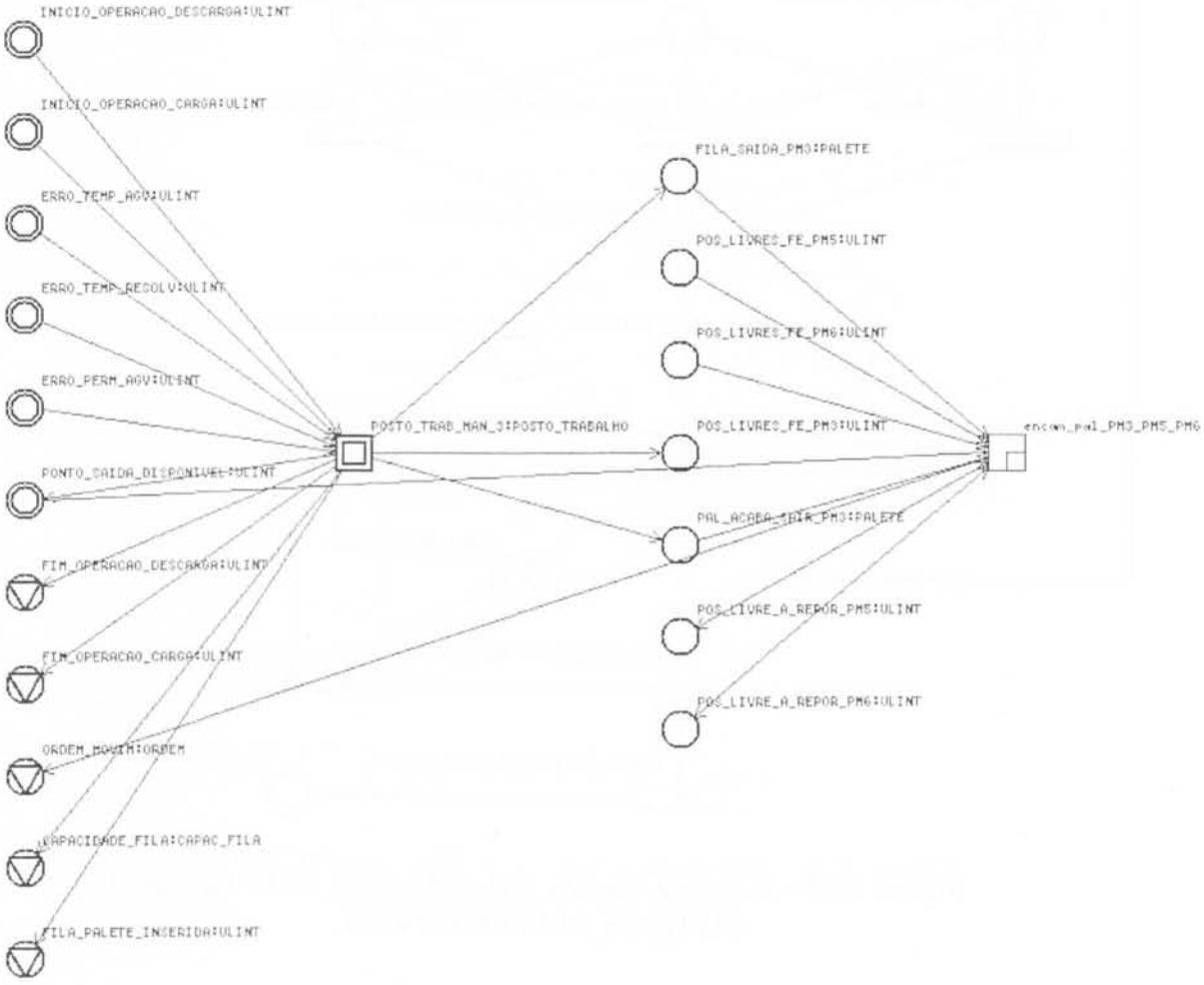


Figura 9.29 – Sub-rede 'Controlo_PM3' da classe SISTEMA_MANUFACTURA.

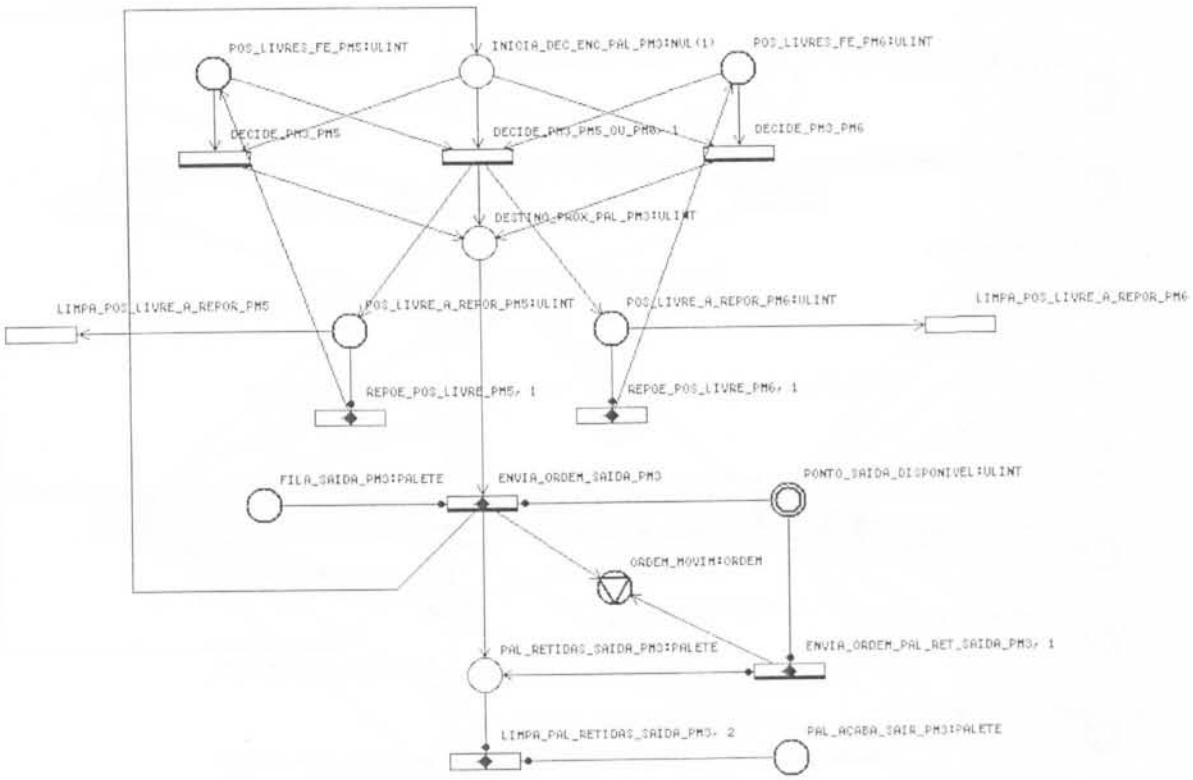


Figura 9.30 – Sub-rede 'encam_pal_PM3_PM5_PM6' da classe SISTEMA_MANUFACTURA.

9.3.2.7 Sub-rede 'Controlo_PM4' da classe SISTEMA_MANUFACTURA

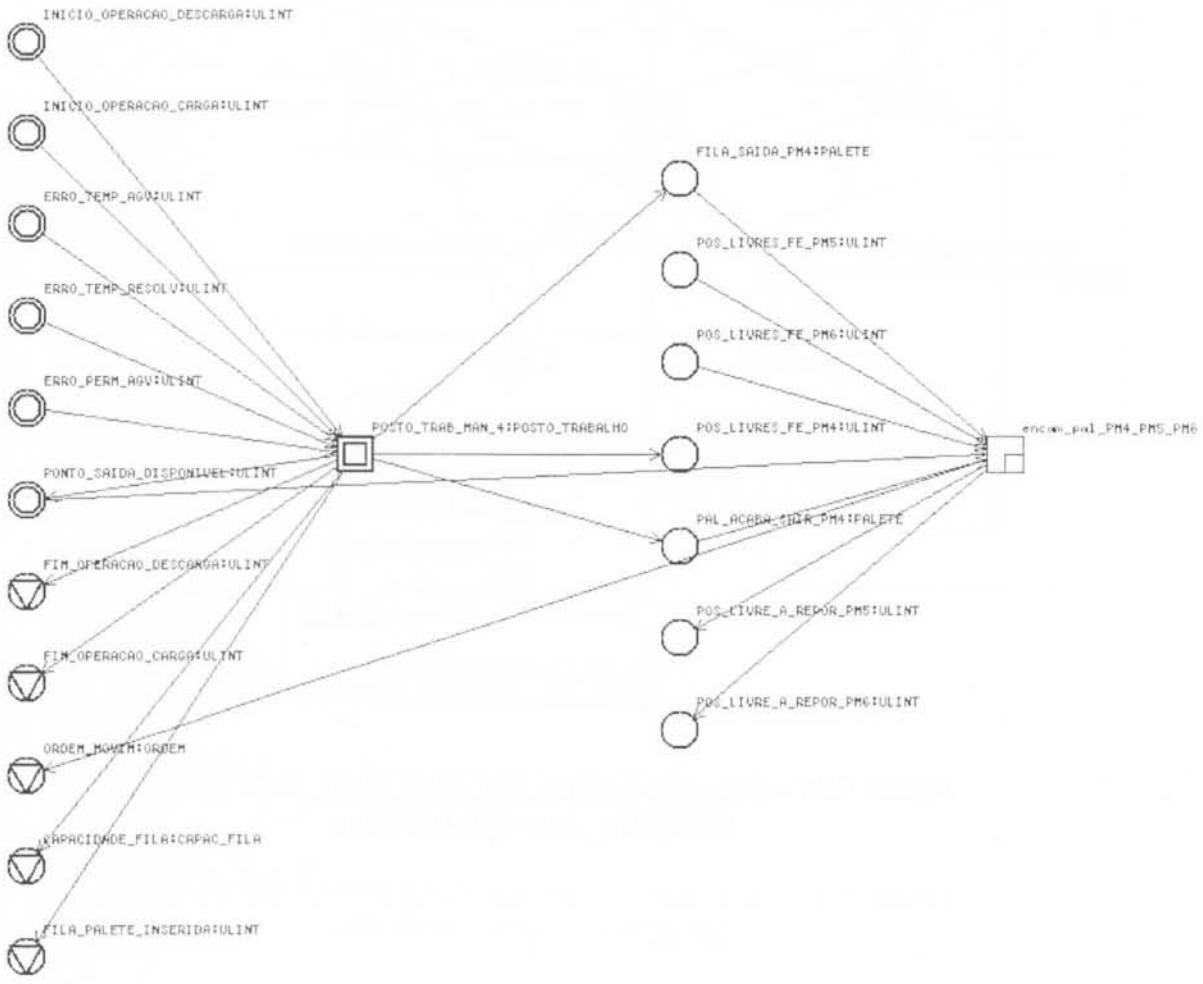


Figura 9.31 – Sub-rede 'Controlo_PM4' da classe SISTEMA_MANUFACTURA.

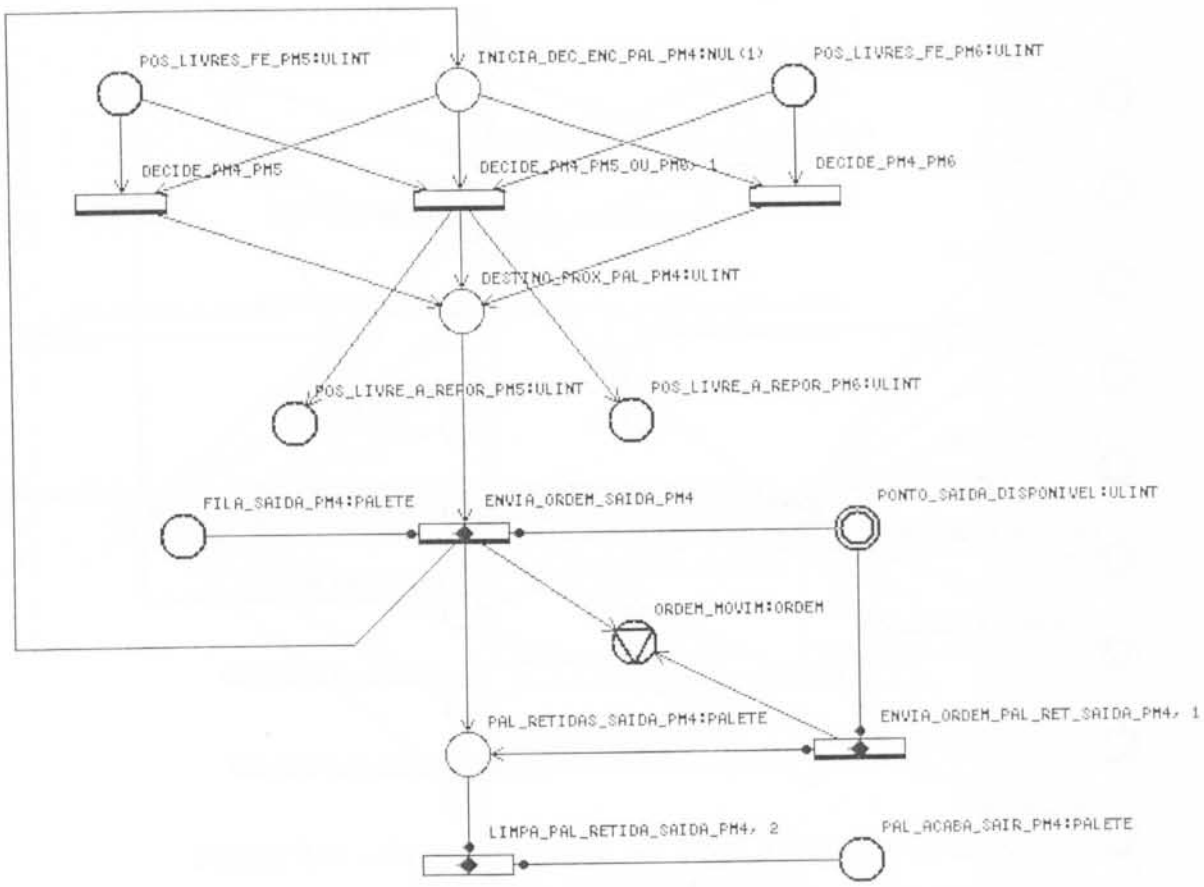


Figura 9.32 – Sub-rede 'encam_pal_PM4_PM5_PM6' da classe SISTEMA_MANUFACTURA.

9.3.2.8 Sub-rede 'Controlo_PM5' da classe SISTEMA_MANUFACTURA

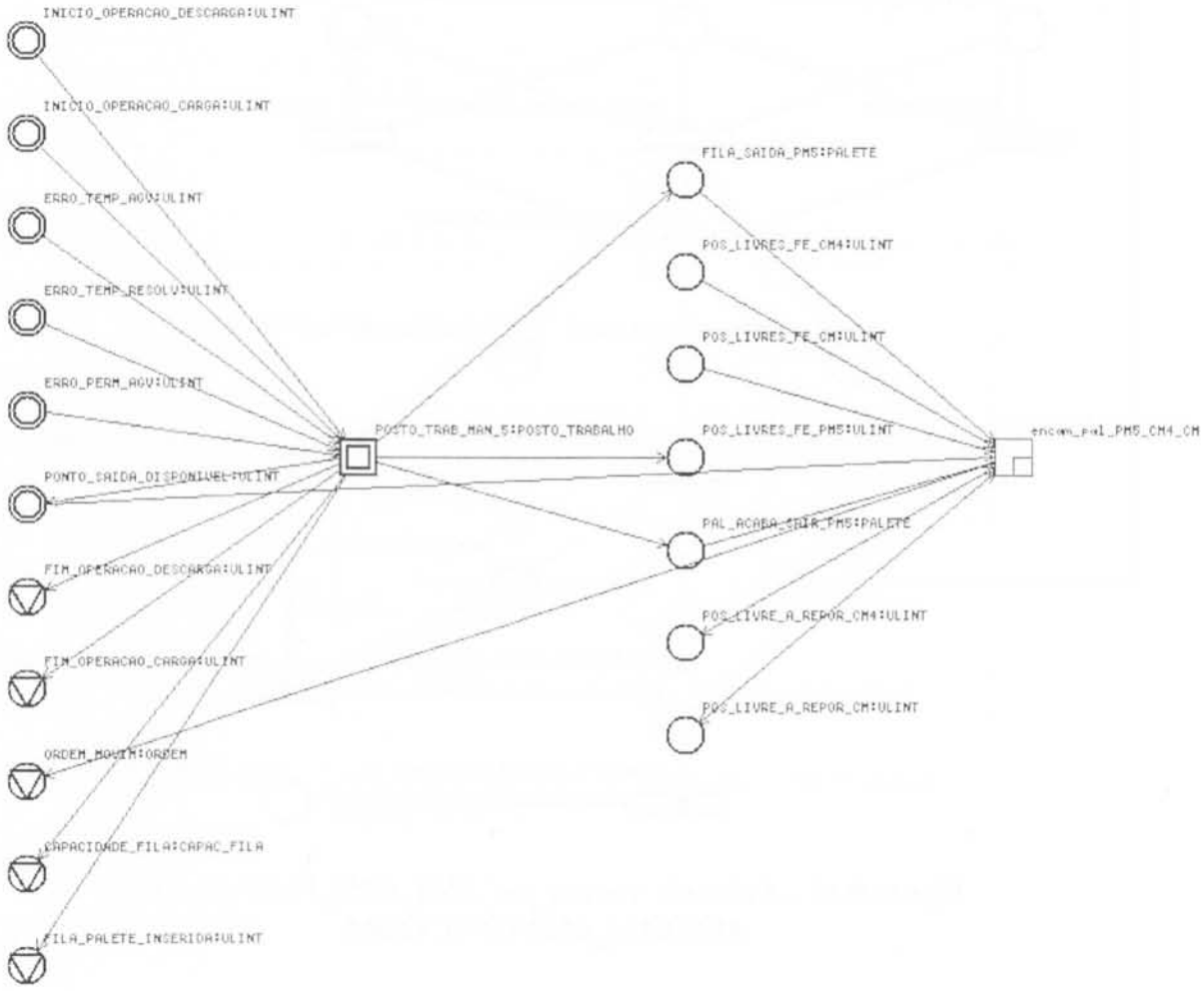


Figura 9.33 – Sub-rede 'Controlo_PM5' da classe SISTEMA_MANUFACTURA.

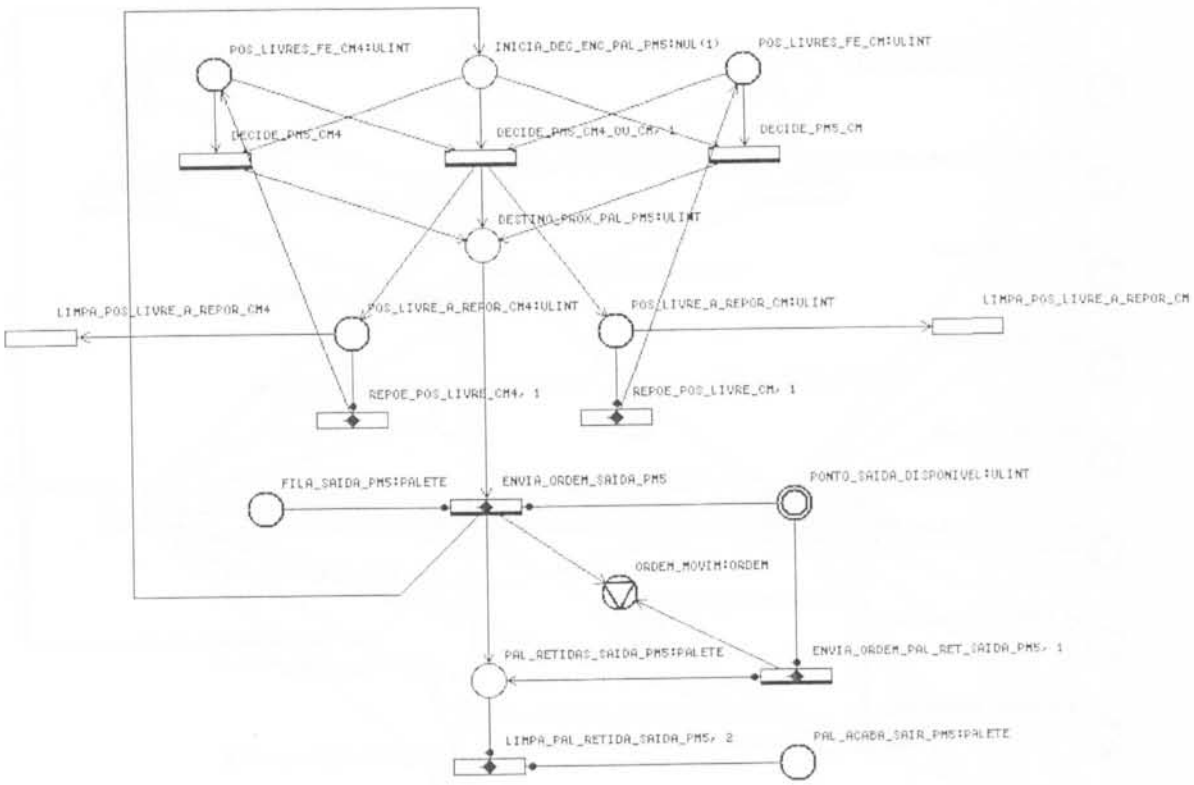


Figura 9.34 – Sub-rede 'encam_pal_PM5_CM4_CM' da classe SISTEMA_MANUFACTURA.

9.3.2.9 Sub-rede 'Controlo_PM6' da classe SISTEMA_MANUFACTURA

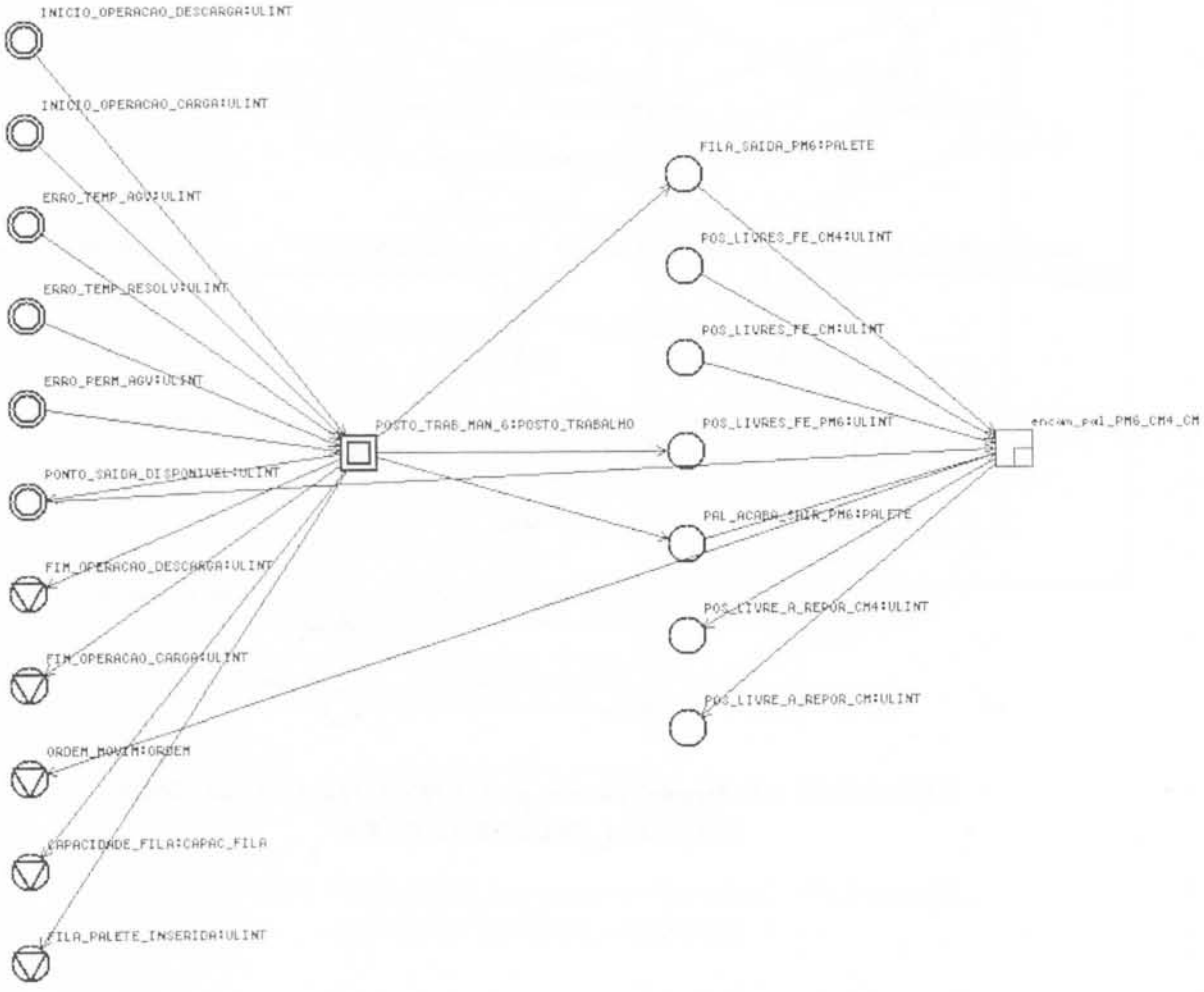


Figura 9.35 – Sub-rede 'Controlo_PM6' da classe SISTEMA_MANUFACTURA.

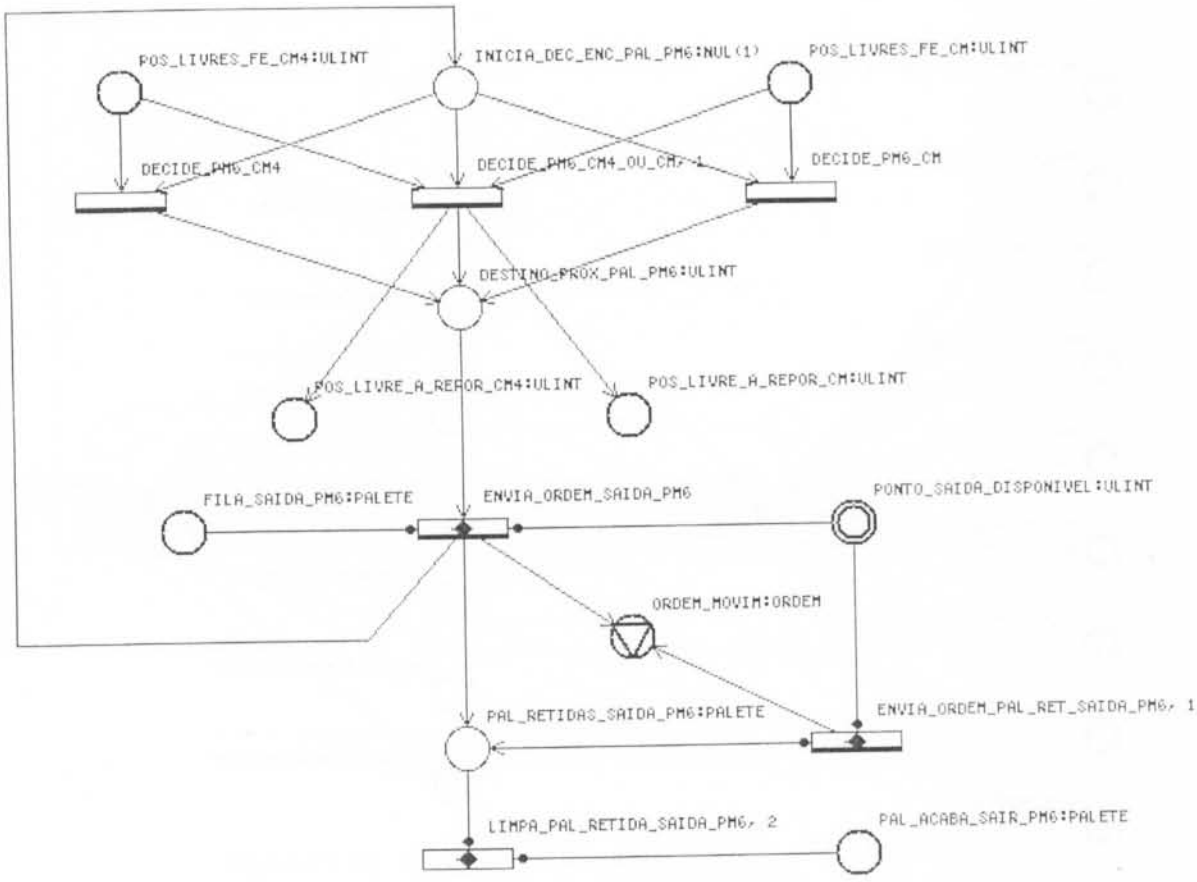


Figura 9.36 – Sub-rede 'encam_pal_PM6_CM4_CM' da classe SISTEMA_MANUFACTURA.

9.3.2.10 Sub-rede 'Controlo_CM3' da classe SISTEMA_MANUFACTURA

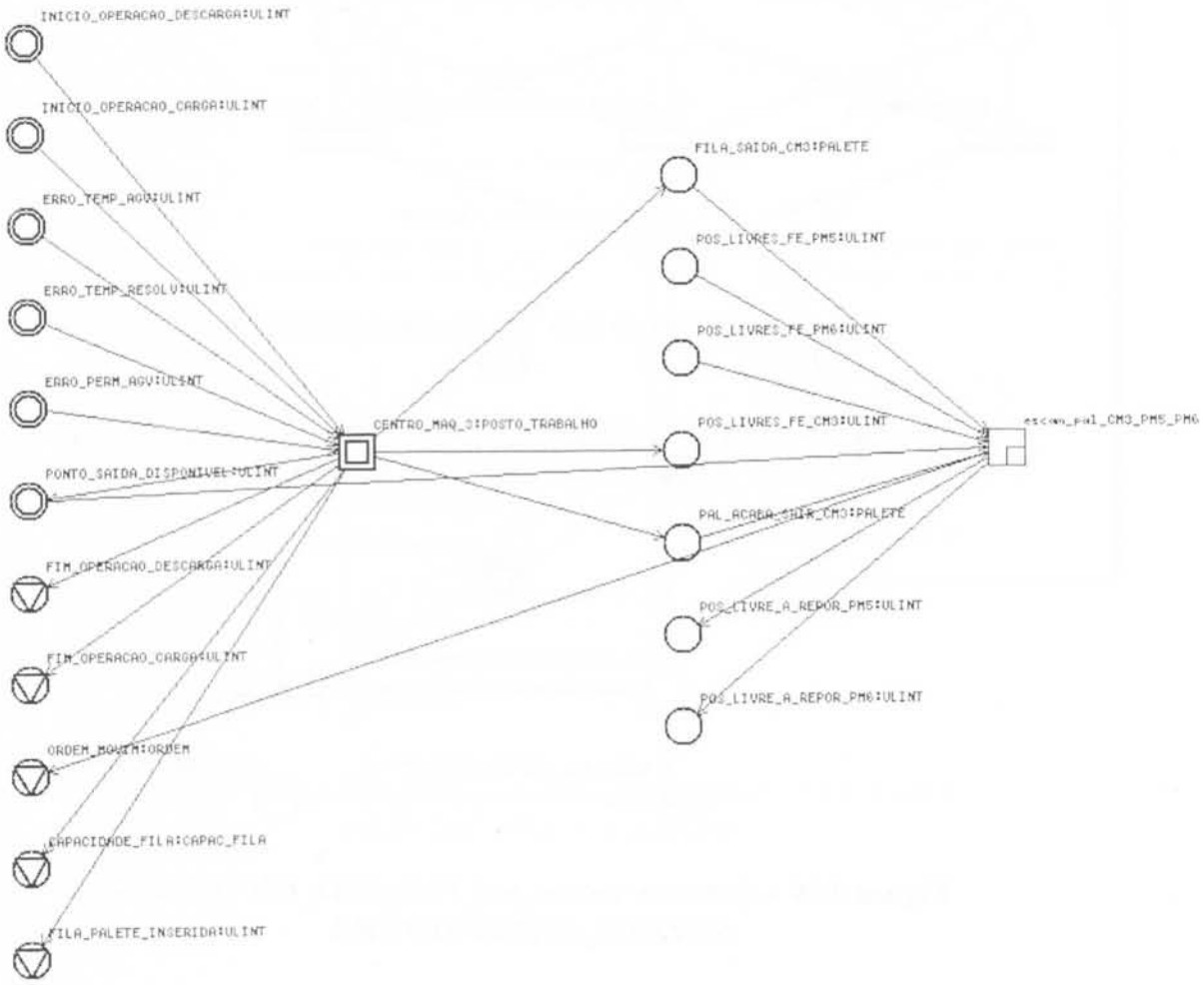


Figura 9.37 – Sub-rede 'Controlo_CM3' da classe SISTEMA_MANUFACTURA.

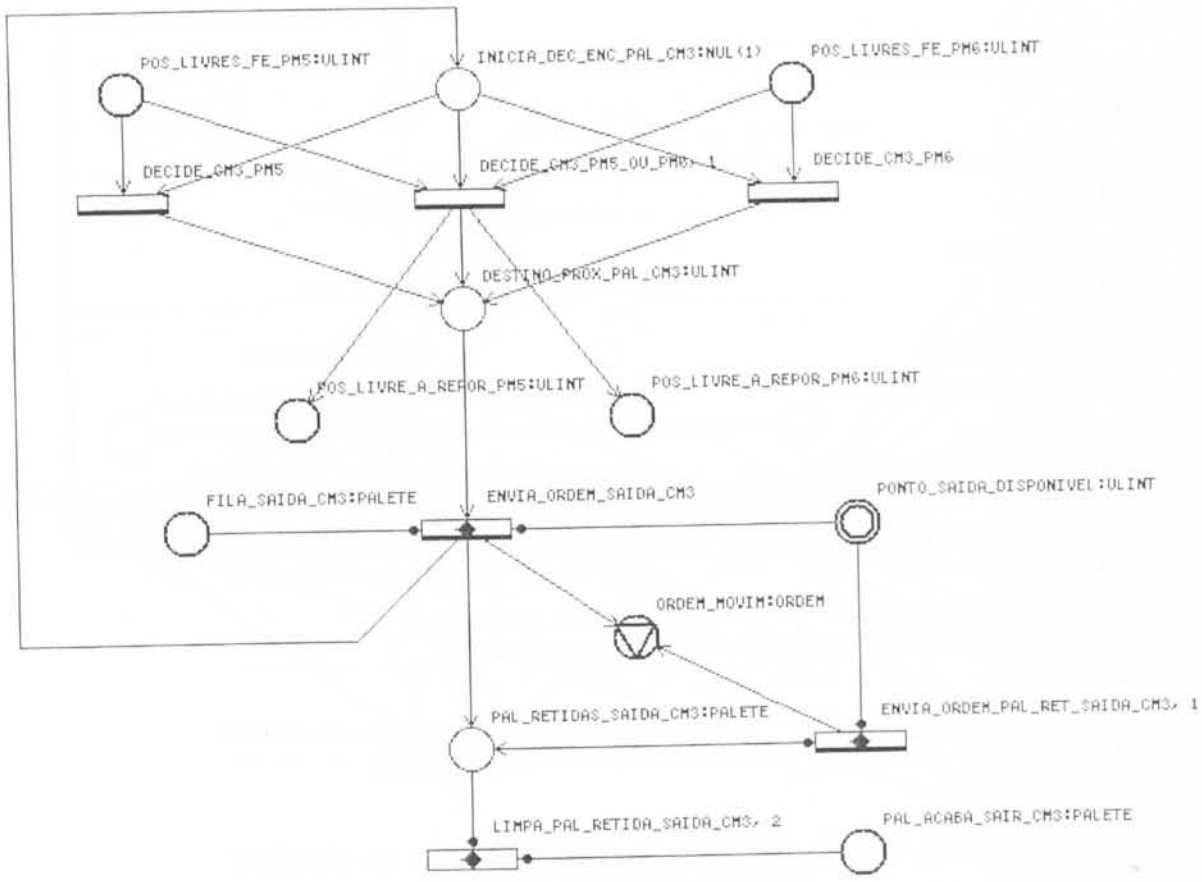


Figura 9.38 – Sub-rede 'encam_pal_CM3_PM5_PM6' da classe SISTEMA_MANUFACTURA.

9.3.2.11 Sub-rede 'Controlo_CM4' da classe SISTEMA_MANUFACTURA

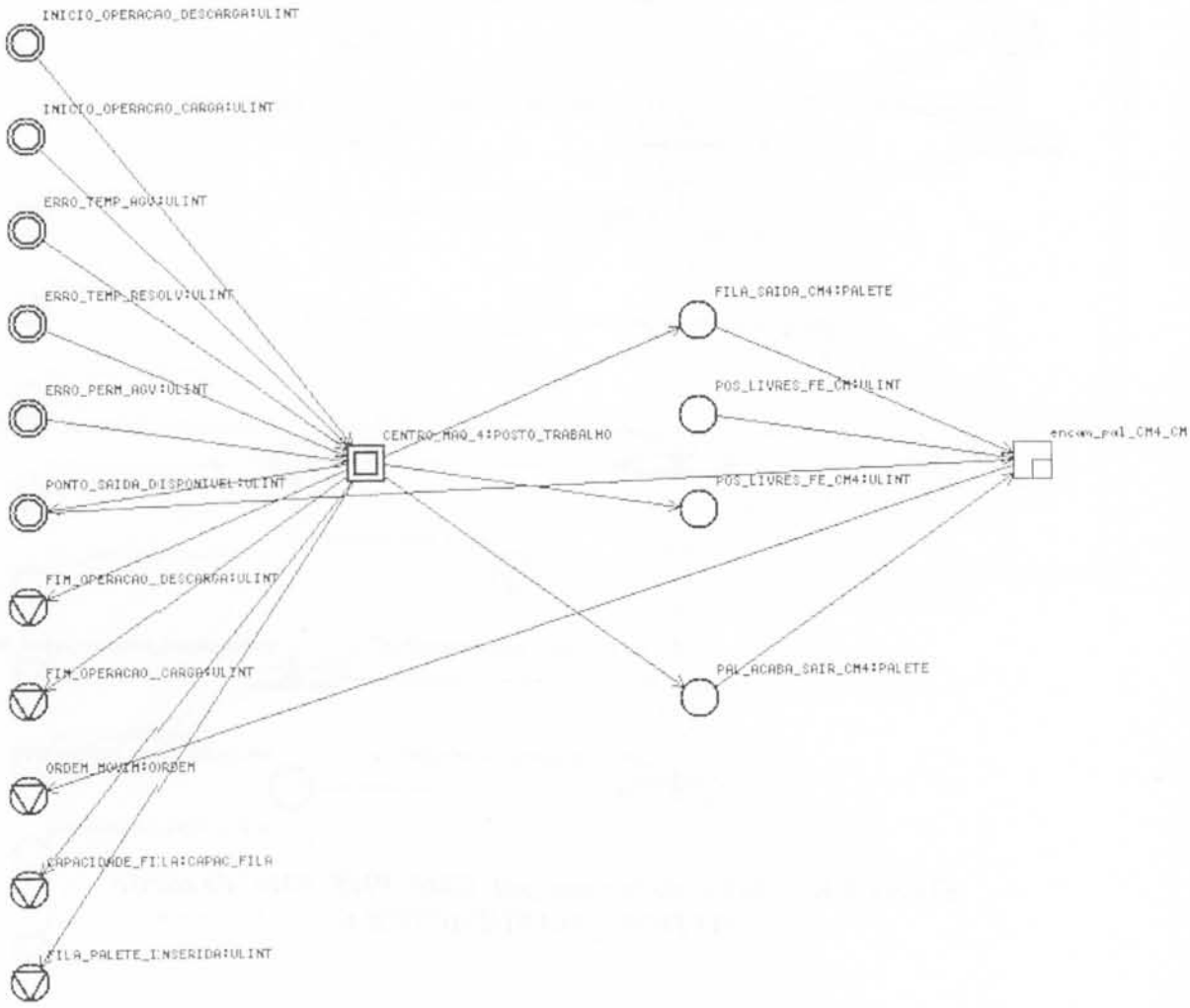


Figura 9.39 – Sub-rede 'Controlo_CM4' da classe SISTEMA_MANUFACTURA.

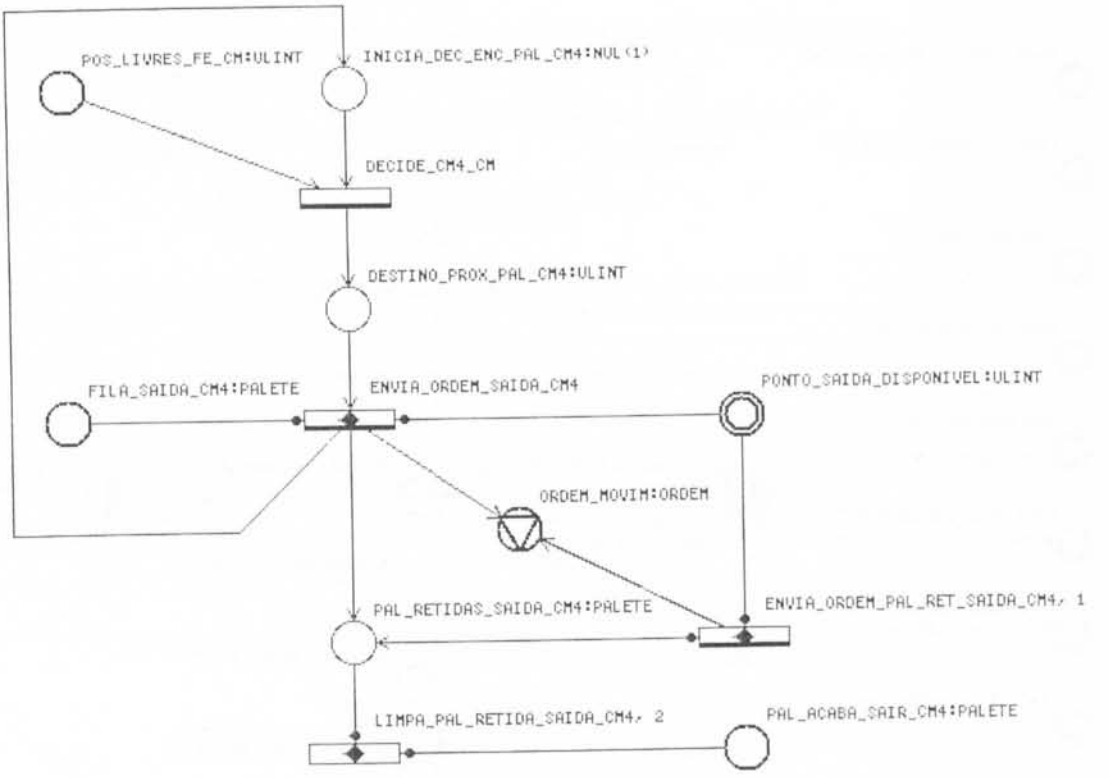


Figura 9.40 – Sub-rede 'encam_pal_CM4_CM' da classe SISTEMA_MANUFACTURA.

9.3.2.12 Sub-rede 'Controlo_CM' da classe SISTEMA_MANUFACTURA

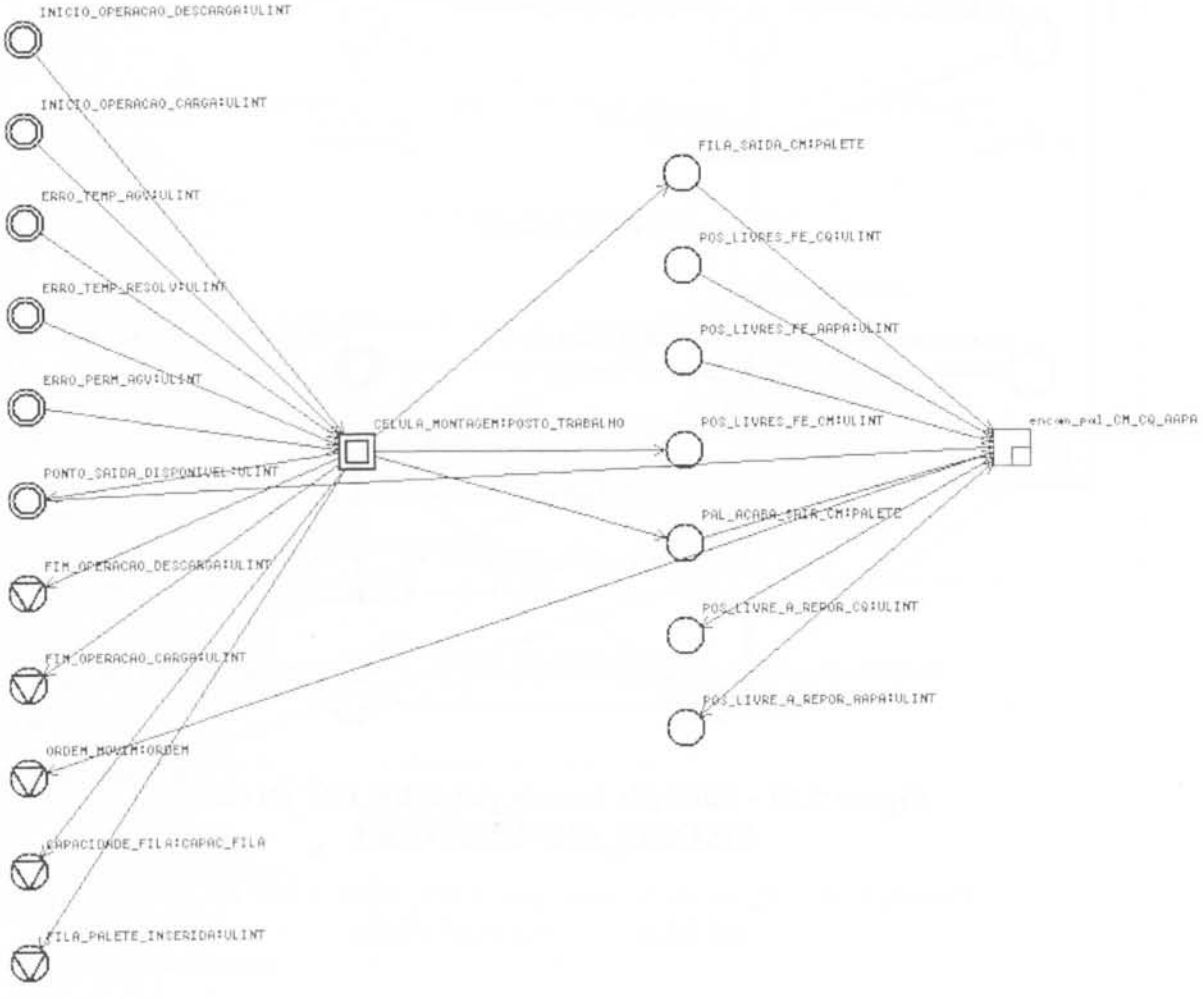


Figura 9.41 – Sub-rede 'Controlo_CM' da classe SISTEMA_MANUFACTURA.

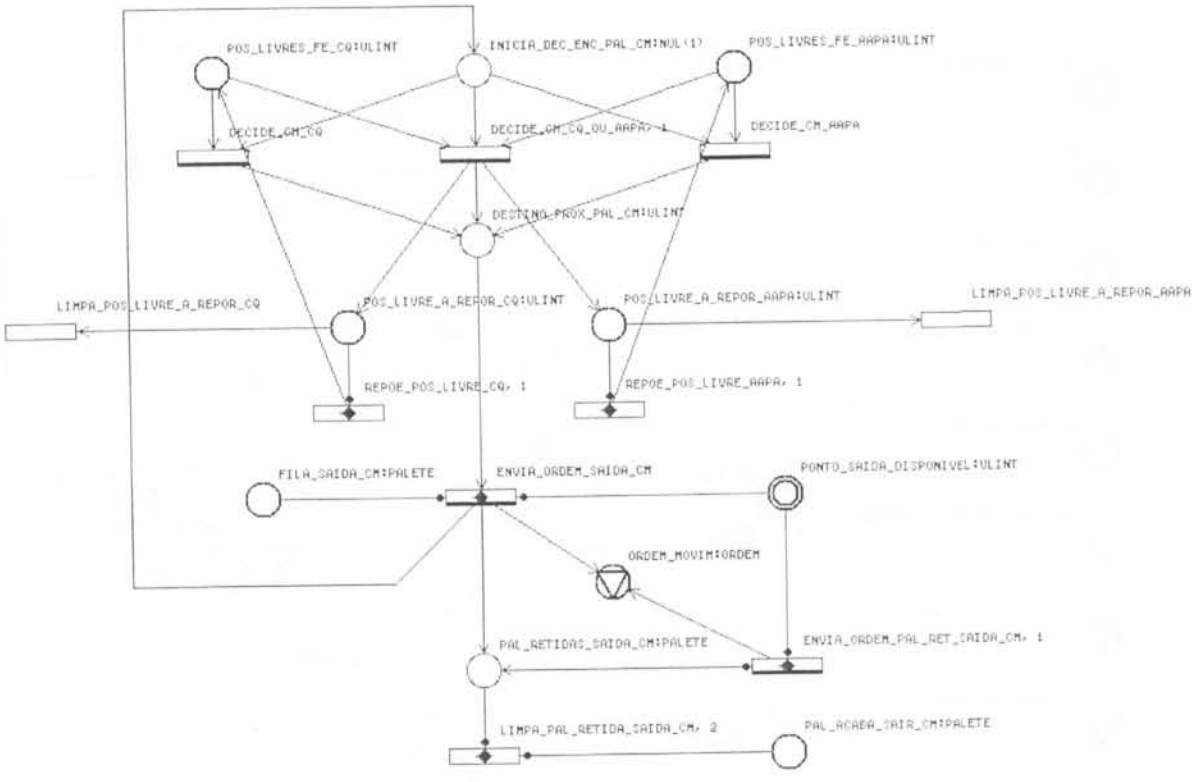


Figura 9.42 – Sub-rede 'encam_pal_CM_CQ_AAPA' da classe SISTEMA_MANUFACTURA.

9.3.2.13 Sub-rede 'Controlo_CQ' da classe SISTEMA_MANUFACTURA

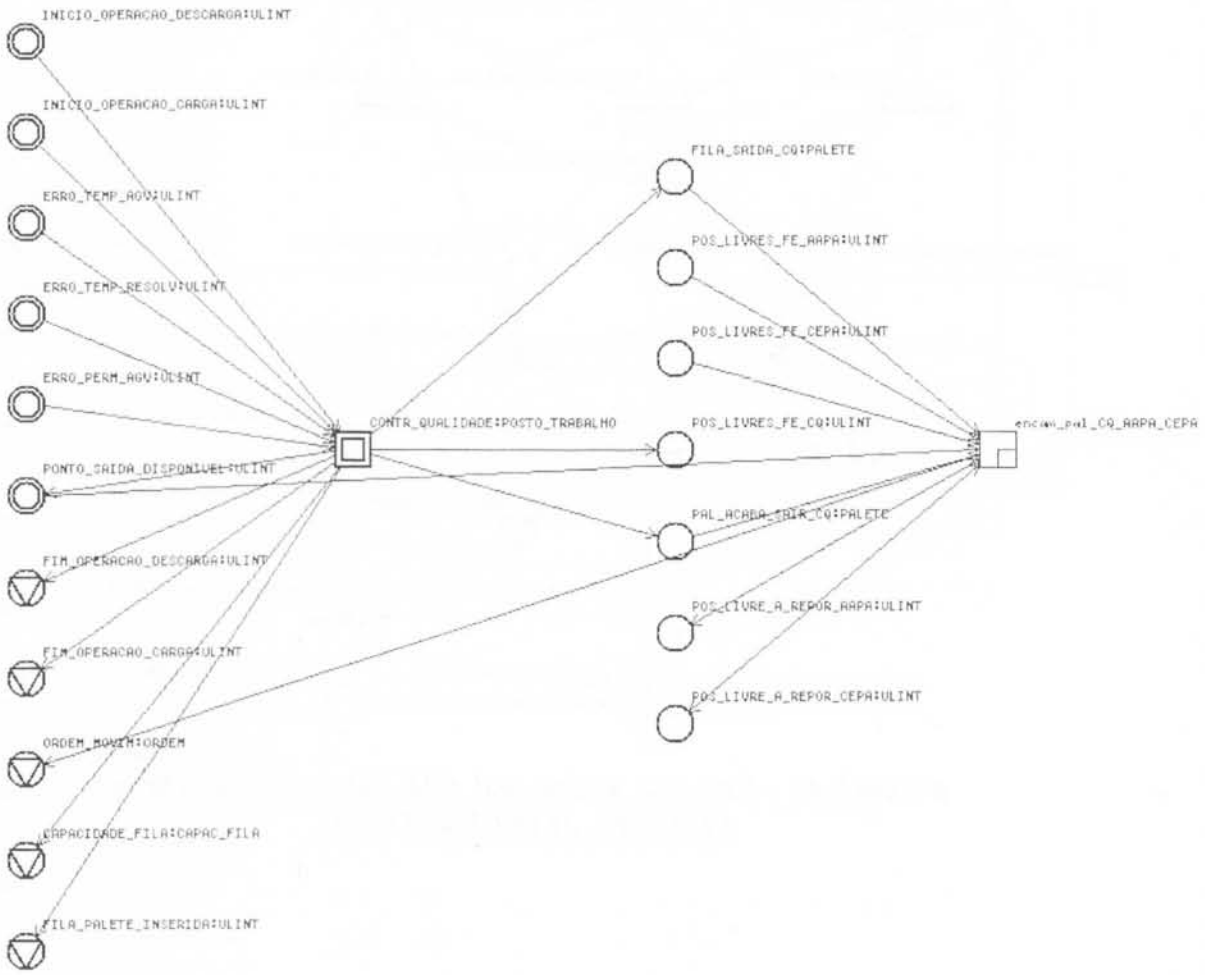


Figura 9.43 – Sub-rede 'Controlo_CQ' da classe SISTEMA_MANUFACTURA.

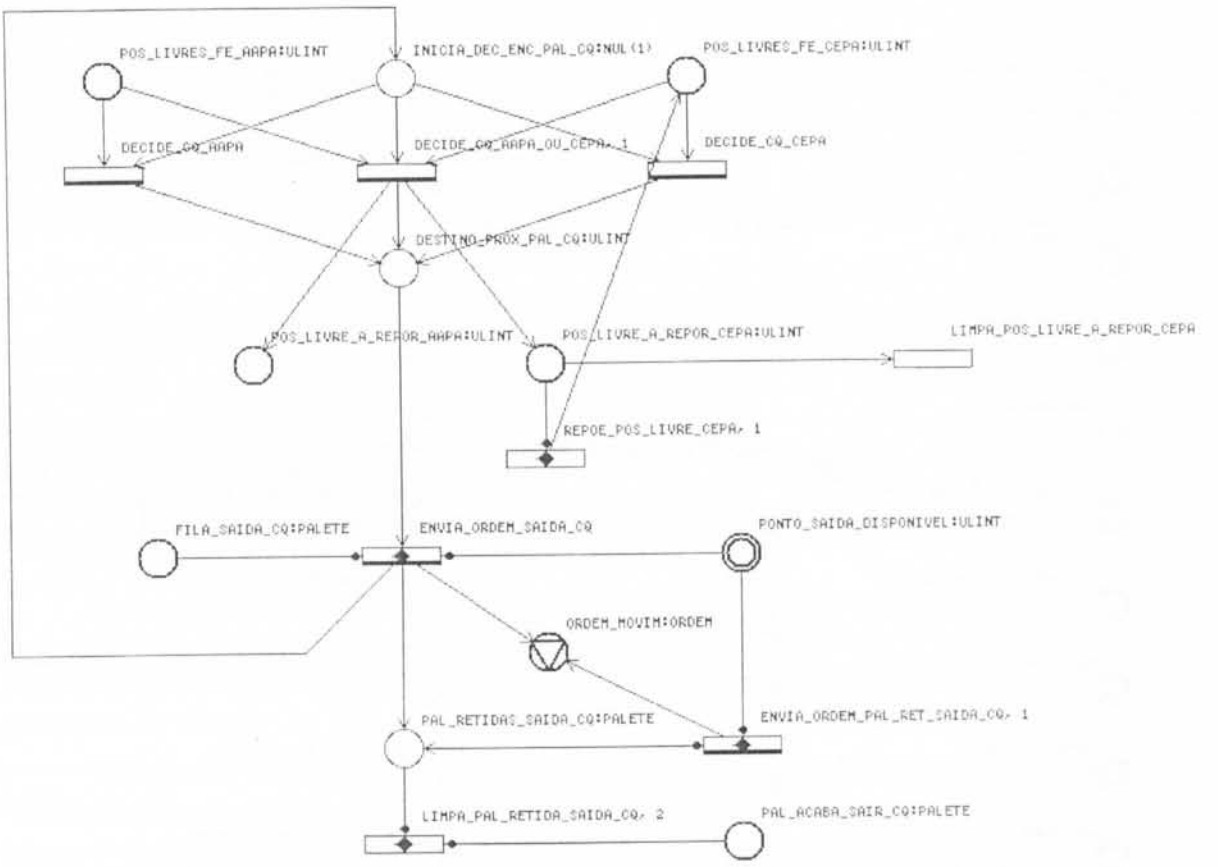


Figura 9.44 – Sub-rede 'encam_pal_CQ_AAPA_CEPA' da classe SISTEMA_MANUFACTURA.

9.3.2.14 Sub-rede 'Controlo_AAPA' da classe SISTEMA_MANUFACTURA

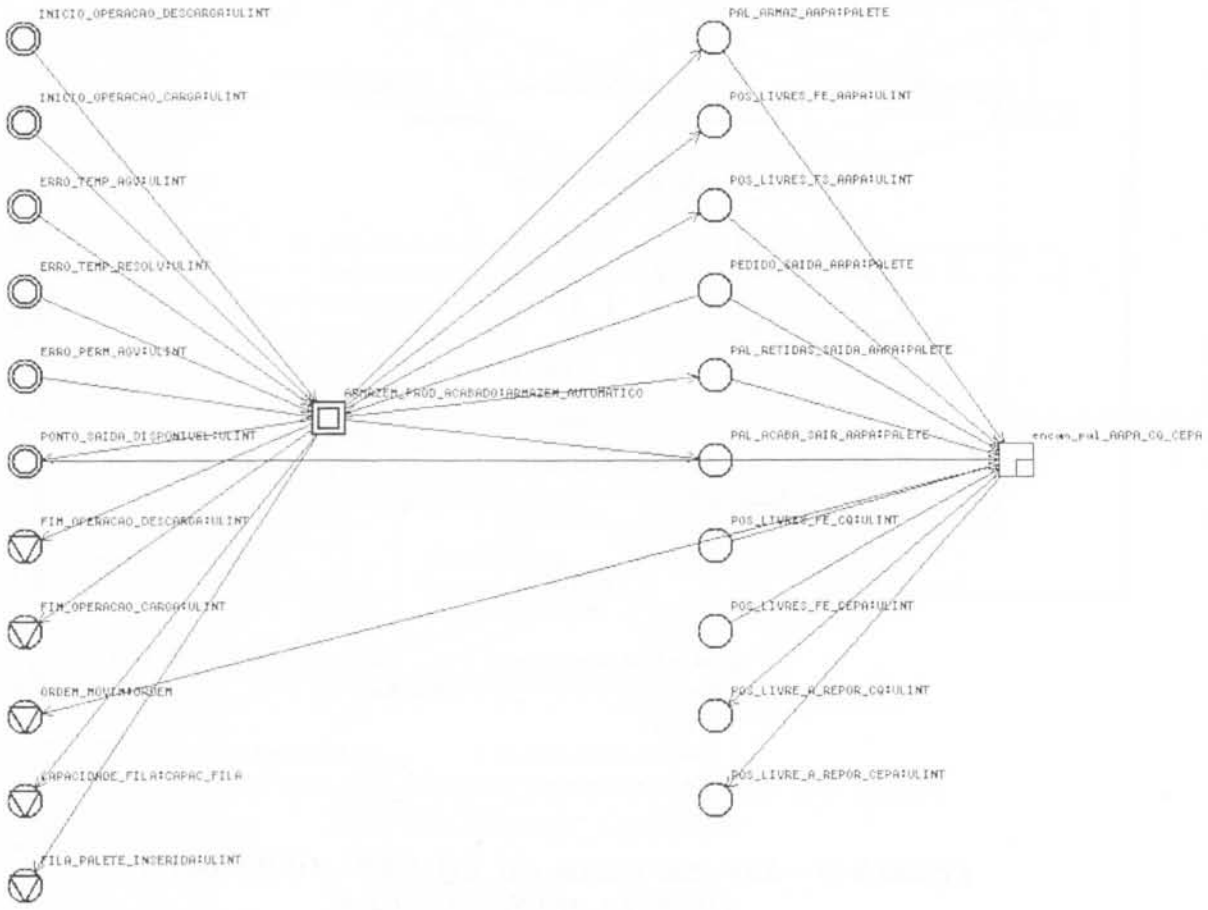


Figura 9.45 – Sub-rede 'Controlo_AAPA' da classe SISTEMA_MANUFACTURA.

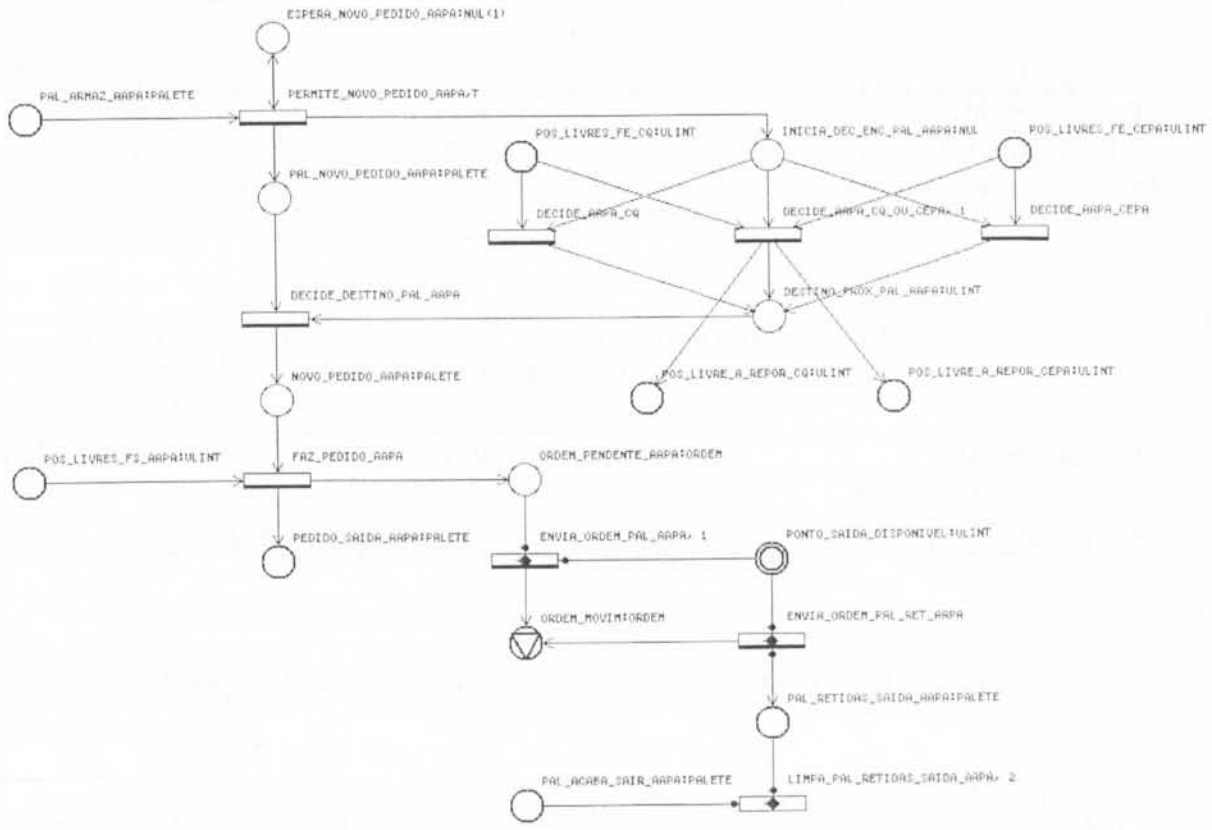


Figura 9.46 – Sub-rede 'encam_pal_AAPA_CQ_CEPA' da classe SISTEMA_MANUFACTURA.

9.4 Construção do modelo global do SAGVs

Instanciando um objecto de cada uma das três classe apresentadas no capítulo 9 (SGM, SGAGV e SPAGV) e um objecto da classe SISTEMA_MANUFACTURA, obtém-se o modelo final do SAGVs, representado na Figura 8.1, da página 117.

As tabelas seguintes evidenciam a correspondência entre posições de entrada e de saída na comunicação entre esses quatro grandes blocos, que tornam possível a existência dos fluxos de informação descritos em 2.2.

Tabela 9.2 – Ligação entre o objecto da classe SGM e o objecto da classe SISTEMA_MANUFACTURA.

Posição SGM	Posição SISTEMA_MANUFACTURA
NOVA_TAREFA	ORDEM_MOVIM
PONTO_CARGA_DISPONIVEL	PONTO_SAIDA_DISPONIVEL

Tabela 9.3 – Ligação entre o objecto da classe SGM e o objecto da classe SGAGV.

Posição SGM	Posição SGAGV
MENS_RECEB_SGAGV	MENS_ENV_SGM
MENS_ENV_SGAGV	MENS_RECEB_SGM

Tabela 9.4 – Ligação entre o objecto da classe SGAGV e o objecto da classe SISTEMA_MANUFACTURA.

Posição SGAGV	Posição SISTEMA_MANUFACTURA
CAPACIDADE_FILA	CAPACIDADE_FILA
FILA_PALETE_INSERIDA	FILA_PALETE_INSERIDA

Tabela 9.5 – Ligação entre o objecto da classe SGAGV e o objecto da classe SPAGV.

Posição SGAGV	Posição SPAGV
MENS_RECEB_SPAGV	MENS_ENV_SGAGV
MENS_ENV_SPAGV	MENS_RECEB_SGAGV

Tabela 9.6 – Ligação entre o objecto da classe SPAGV e o objecto da classe SISTEMA_MANUFACTURA.

Posição SPAGV	Posição SISTEMA_MANUFACTURA
FIM_OPERACAO_CARGA	FIM_OPERACAO_CARGA
FIM_OPERACAO_DESCARGA	FIM_OPERACAO_DESCARGA
INICIO_OP_CARGA	INICIO_OPERACAO_CARGA
INICIO_OP_DESCARGA	INICIO_OPERACAO_DESCARGA
OP_TRANSF_CARGA_INTERR	ERRO_TEMP_AGV
OP_TRANSF_CARGA_RETOM	ERRO_TEMP_RESOLV
OP_TRANSF_CARGA_CANCEL	ERRO_PERM_AGV

O anexo E contém informação detalhada sobre a implementação da classe SISTEMA_DE_AGVS. As acções iniciais e as acções finais permitem gerir a eventual utilização de um ficheiro de log, para registar todas as mensagens canalizadas pela função 'prn_string()', ao longo de todo o modelo. O anexo G apresenta excertos de um ficheiro de log, gerado durante uma das simulações referidas no capítulo 10.

10. Apresentação e análise de resultados das simulações

Este capítulo começa por apresentar o *layout* do SAGVs e a metodologia seguida na realização de simulações do sistema de fabrico modelizado pela RP estocástica e colorida, representada pela Figura 8.1 da página 117. Na secção 10.3 são apresentados e analisados os principais resultados obtidos nas simulações..

10.1 Configuração do sistema de fabrico utilizado nas simulações

O sistema de fabrico representado na Figura 9.1 da página 170 não mostra o *layout* de trajectórias do SAGVs. A Figura 10.2 mostra a mesma planta, mas já incluindo o *layout* de trajectórias dos AGVs. O sentido de circulação normal dos AGVs é contrário ao sentido dos ponteiros do relógio.

De acordo com a numeração dos nós aí representada e com a sintaxe definida pela Tabela 3.6 (página 26), o ficheiro de configuração do *layout* de AGVs é o que consta no anexo F.3. Este ficheiro inclui a definição dos nós fictícios 201 a 232, utilizados para gerir cruzamentos e áreas partilhadas por mais do que uma trajectória. Para tornar o desenho da planta fabril mais legível, a Figura 10.2 não representa esses nós fictícios.

A Figura 10.1 exemplifica a definição do nó fictício 222, utilizado para gerir o cruzamento existente entre as trajectórias (132, 21) e (20, 133), permitindo porém que os movimentos (132, 133) e (20, 21) possam ocorrer em simultâneo.

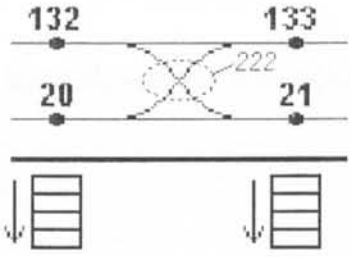


Figura 10.1 – Exemplificação da definição de um nó fictício no layout do SAGVs simulado.

O ficheiro de configuração da posição dos pontos de carga/descarga, em relação aos respectivos nós do *layout* é apresentado no anexo F.4. Este segue a sintaxe definida pela Tabela 7.14 e o raciocínio exposto em 7.2.3.3, para a definição da posição relativa dos pontos de carga/descarga.

O ficheiro de configuração com as probabilidades de introdução e de posicionamento de um AGV em cada nó do *layout*, é apresentado no anexo F.5. Este segue a sintaxe definida na Tabela 8.1.

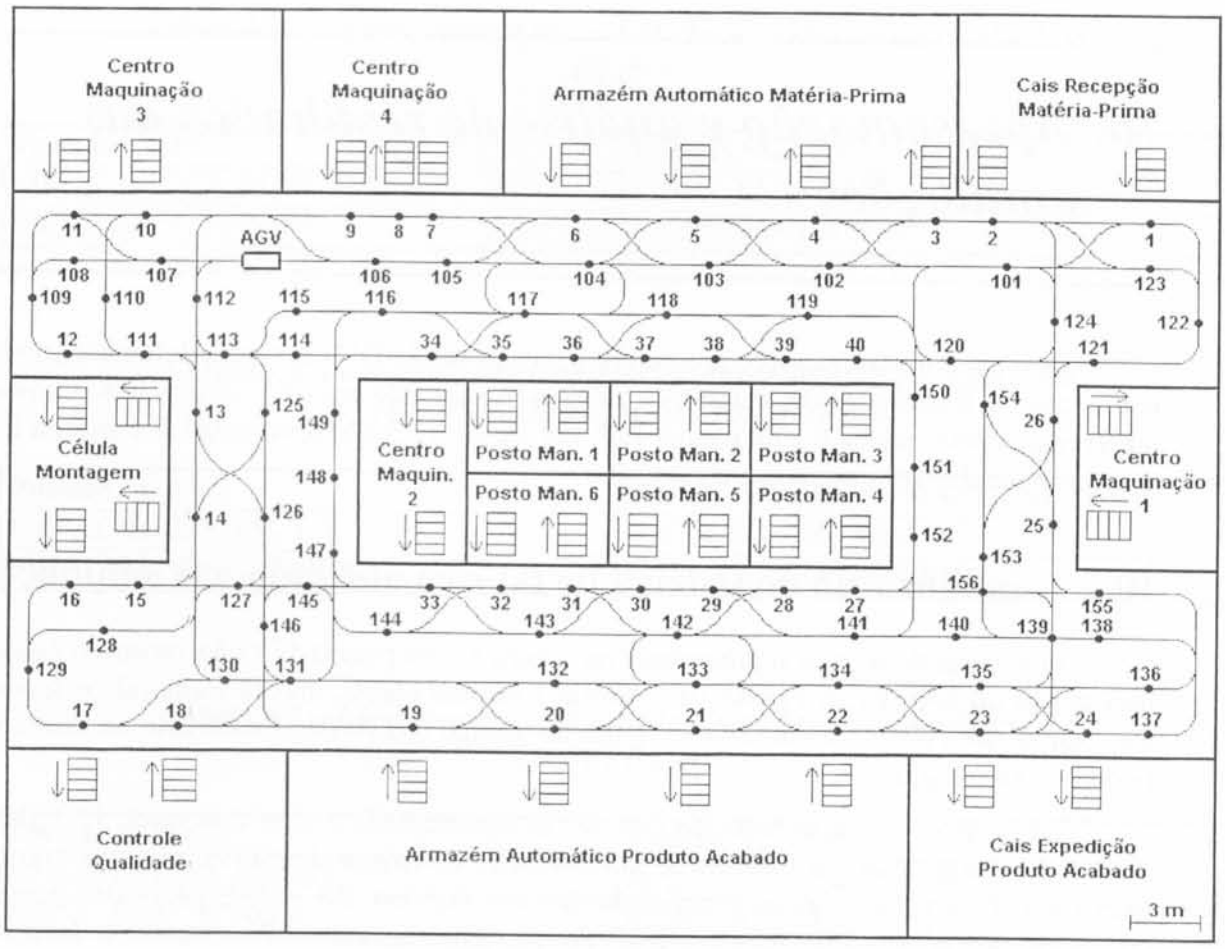


Figura 10.2 – Planta do layout do SAGVs simulado.

O ficheiro de configuração dos parâmetros do modelo do SAGVs é apresentado no anexo F.1. O ficheiro de configuração dos parâmetros do modelo do sistema de fabrico (objecto FABRICA da classe SISTEMA_MANUFACTURA) é apresentado no anexo F.2

10.2 Metodologia utilizada na realização das simulações

Definida a configuração do sistema de fabrico dotado de um SAGVs, realizaram-se nove simulações da fábrica ao longo de um dia de trabalho (86400 s). Para cada uma das simulações utilizou-se uma combinação diferente de regras de despacho no escalonador do SAGVs, variando os parâmetros 'regra_ADCT', 'regra_ADV' e 'reaf_dinam' do SGAGV (ver descrição dos parâmetros do SGAGV no anexo C.3.4).

A Tabela 10.1 indica as regras de despacho utilizadas em cada uma das simulações efectuadas. Ao longo do capítulo, as regras de despacho e as respectivas simulações são designadas pelas siglas aí indicadas entre parênteses. Nas simulações 1 e 2 utilizou-se o algoritmo húngaro no escalonamento das tarefas. No primeiro caso utilizou-se também o mecanismo de reafectação dinâmica.

Tabela 10.1 – Regras de afectação utilizadas nas simulações.

Simulação Nº	Regra de despacho para a afectação desencadeada por um veículo	Regra de despacho para a afectação desencadeada por um centro de trabalho	Reafectação dinâmica
1	Tempo Mais Curto de Viagem (TMCV)	Veículo Mais Próximo (VMP)	Sim
2	Tempo Mais Curto de Viagem (TMCV)		Não
3	Máximo Tamanho da Fila de Saída (MTFS)		
4	Mínimo Espaço Livre na Fila de Saída (MELFS)		
5	Primeiro a Chegar Primeiro a ser Servido, Modificada (PCPSM)		
6	Tempo Mais Curto de Viagem (TMCV)	Veículo Menos Utilizado (VMU)	
7	Máximo Tamanho da Fila de Saída (MTFS)		
8	Mínimo Espaço Livre na Fila de Saída (MELFS)		
9	Primeiro a Chegar Primeiro a ser Servido, Modificada (PCPSM)		

As três tabelas seguintes indicam os valores dos parâmetros de configuração dos objectos que modelizam as três camadas de gestão do SAGVs.

Como se pode ver, não foi permitido o lançamento de tarefas escalonadas pelo SGM. Assim, apenas se permitiu a ocorrência de tarefas do tipo 'NORMAL' e do tipo 'POSIC_ESPEC'. Também se procurou evitar o cancelamento de tarefas e a inversão do estado de alocação dos AGVs, atribuindo aos respectivos parâmetros um tempo médio muito superior ao tempo de simulação, e configurando 'p_alocado' com o valor zero.

Tabela 10.2 – Valores dos parâmetros do SGM utilizados nas simulações.

numero_agvs	5
vm_carga	10.0
dp_carga	2.0
p_alocado	0.0
p_esc_sgm	1.0
p_tar_prior	0.0
priorid_tar_max	10.0
p_cancel_1_tar	0.7
p_cancel_tar_exec	0.2
t_med_cancel_tar	259200.0
t_med_dis_en	1.0
t_med_en_dis	604800.0
t_med_mud_est_aloc	31104000.0

Tabela 10.3 – Valores dos parâmetros do SGAGV utilizados nas simulações.

numero_agvs	5
num_max_af	5
num_max_mov_avan	1
reaf_dinam	
regra_ADCT	
regra_ADV	
vel_curva	75.0
vel_frente	100.0
vel_tras	50.0

Tabela 10.4 – Valores dos parâmetros do SGAGV utilizados nas simulações.

numero_agvs	5
p_erro_manual	0.1
p_introd_or_pos	1.0
t_med_erro	10800.0
t_med_introducao	60.0
t_med_manual	172800.0
t_med_perm_erro	300.0
vel_curva	75.0
vel_frente	100.0
vel_tras	50.0

Os valores dos parâmetros que condicionam o encaminhamento das paletes no interior da fábrica (parâmetros 'p_...') e os pedidos de saída de paletes dos armazéns (parâmetros 't_entre_ped....'), estão indicados na Tabela 10.5. As probabilidades são as mesmas que estão indicadas na TTabela 9.1.

Tabela 10.5 – Valores dos parâmetros da fábrica utilizados nas simulações.

p_AAMP_CM1_CM2	0.7
p_AAPA_CQ_CEPA	0.0
p_CM1_PM1_PM2	0.5
p_CM2_PM3_PM4	0.5
p_CM3_PM5_PM6	0.5
p_CM_CQ_AAPA	0.8
p_CQ_AAPA_CEPA	0.1
p_CRMP_AAMP_CM1	0.9
p_PM1_PM3_CM3	0.5
p_PM2_PM4_CM3	0.5
p_PM3_PM5_PM6	0.5
p_PM4_PM5_PM6	0.5
p_PM5_CM4_CM	0.5
p_PM6_CM4_CM	0.5
t_entre_ped_AAMP	180.0
t_entre_ped_AAPA	60.0

Os valores dos parâmetros das células da planta fabril estão indicados nas tabelas seguintes.

Tabela 10.6 – Valores dos parâmetros do CRMP utilizados nas simulações.

capacidade_filas	4
dp_tcarga	0.5
dp_trecep_pal	7.7
fila1	1
filan	2
num_recursos	2
tam_lote_paletes	300
tm_entre_lotes	86400.0
tm_recep_pal	40.0
vm_tcarga	20.0

Tabela 10.7 – Valores dos parâmetros dos armazéns automáticos utilizados nas simulações.

Armazém	AAMP	AAPA
capac_armazem	100	100
capac_filas_entrada	4	2
capac_filas_saida	4	4
dp_tcd	0.5	0.5
dp_tcd_stoc	0.5	0.5
dp_tstoc_carreg	10.2	10.2
dp_tstoc_vazio	10.2	10.2
fila_entr_1	3	19
fila_entr_n	4	20
fila_saida_1	5	21
fila_saida_n	6	22
vm_tcd	20.0	20.0
vm_tcd_stoc	15.0	15.0
vm_tstoc_carreg	40.0	40.0
vm_tstoc_vazio	35.0	35.0

Tabela 10.8 – Valores dos parâmetros dos centros de maquinação, postos de trabalho manuais, célula de montagem e centro de controle de qualidade utilizados nas simulações.

Célula de fabrico	CM1	CM2	CM3	CM4	PM1	PM2	PM3	PM4	PM5	PM6	CM	CQ
capac_fila_saida	2	4	4	4	4	4	4	4	4	4	2	2
capac_filas_entrada	1	2	4	2	2	2	4	4	4	4	2	2
dp_tcd	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
dp_tdownload	5.1	5.1	2.6	5.1	0.5	0.5	0.5	0.5	0.5	0.5	2.6	2.6
dp_tprocess	30.6	15.3	15.3	15.3	61.2	61.2	45.9	45.9	61.2	61.2	30.6	10.2
fila_entr_1	26	34	10	7	35	37	39	27	29	31	12	17
fila_entr_n	26	34	10	8	35	37	39	27	29	31	14	17
fila_saida	25	33	11	9	36	38	40	28	30	32	16	18
vm_tcd	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
vm_tdownload d	30.0	25.0	20.0	30.0	5.0	5.0	5.0	5.0	5.0	5.0	30.0	10.0
vm_tprocess	240.0	300.0	270.0	360.0	300.0	300.0	240.0	240.0	360.0	360.0	300.0	120.0

Tabela 10.9 – Valores dos parâmetros do cais de expedição de produto acabado utilizados nas simulações.

capacidade_filas	2
dp_exped_pal	7.7
dp_tcarreg	459.2
dp_tdescarga	0.5
fila1	23
filan	24
num_recursos	2
tam_lote_paletes	50
tm_entre_carreg	86400.0
tm_exped_pal	40.0
vm_tcarreg	1800.0
vm_tdescarga	20.0

O sisignificado dos parâmetros indicados nas tabelas anteriores pode ser encontrado nos anexos C e e D.

10.3 Appresentação e discussão dos resultados obtidos nas simulaçções

Os ; objectos que modelizam as entidades do sistema de fabrico e o objecto representatativo do SGAGV fazem a compilação de estatísticas, que caracterizam o desempenhho do sistema ao longo da simulação. Estas permitem obter os resultados apresentaddos ao longo desta secção, referentes às nove simulações efectuadas. A configuração utilizada erem cada simulação foi apresentada nas duas secções anteriores.

Ao › longo da secção, a identificação das diferentes simulações é feita à custa das siglas apresentaddas na Tabela 10.1, referentes a regras de despacho, e à custa das siglas seguintes:

- RADV – Regra para a Afectação Desencadeada por um Veículo;

- RADCT – Regra para a Afectação Desencadeada por um Centro de Trabalho;
- REAFD – Reafectação Dinâmica.

10.3.1 Estatísticas referentes ao controlo do SAGVs

A Tabela 10.10 apresenta contagens relacionadas com o lançamento e a execução de tarefas. Como se pode observar, há diferenças significativas entre as várias simulações. O número de tarefas lançadas é superior ao número de tarefas executadas, porque no momento em que a simulação termina (t=86400 s), o SAGVs ainda tem tarefas em fila de espera e em execução.

As diferenças existentes na contagem de tarefas do tipo 'POSIC_ESPEC' não são relevantes do ponto de vista do desempenho do controlo do SAGVs, porque o lançamento destas resulta da ocorrência de conflitos na gestão de tráfego e de avarias nos AGVs, que não dependem directamente da política de escalonamento utilizada em cada caso. Porém, acabam por influenciar o número de tarefas do tipo 'NORMAL' executadas, porque sendo executadas mais tarefas de posicionamento, resta menos tempo para a execução de tarefas do tipo 'NORMAL'.

Contudo, as diferenças na contagem do número de tarefas do tipo 'NORMAL' lançadas e executadas não podem ser explicadas com base nesse factor. Por exemplo, na simulação 2 foram executadas mais tarefas do tipo 'POSIC_ESPEC' do que na simulação 7, mas apesar disso, na simulação 2 foram executadas mais tarefas do tipo 'NORMAL' do que na simulação 7 (mais 344 tarefas). Isto é um claro indício de que a política de escalonamento utilizada acaba por ser o factor mais importante nas diferenças verificadas entre as várias simulações.

O gráfico da Figura 10.3 evidencia as diferenças no número de tarefas do tipo 'NORMAL' executadas. Com base neste conjunto de resultados, a melhor política de escalonamento é (TMCV;VMP;Não), logo seguida de (TMCV;VMP;Sim) e de (TMCV;VMU;Não). Estas três combinações apresentam um desempenho claramente superior a todas as outras.

Este resultado permite obter duas conclusões:

- por um lado, o facto da primeira e da terceira melhor combinação terem desempenhos muito próximos, permite concluir que a escolha da regra de despacho utilizada na afectação desencadeada por um centro de trabalho não é muito relevante; isto é compreensível, porque na maior parte dos casos há mais tarefas em fila de espera do que AGVs disponíveis para as executarem, excepto na fase do arranque do sistema de fabrico;
- por outro lado, a regra de despacho TMCV está presente nas três combinações; como a grande vantagem desta regra é permitir a optimização do tempo gasto pelos AGVs nas execução do passo 1 das tarefas (deslocamento desde o ponto onde se encontrem até ao ponto de carga), conclui-se que o desempenho do sistema é muito influenciado pelo tempo médio de execução do passo 1 de uma tarefa.

Tabela 10.10 – Contagens de tarefas.

Simulação N.º	1	2	3	4	5	6	7	8	9
RAKDV	TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADDCT	VMP					VMU			
REAAFD	Sim	Não							
Tarefas 'NORMAL' lançadas	3352	3408	2964	3138	3093	3350	3097	3072	2908
Tarefas 'NORMAL' executadas	3344	3398	2951	3125	3079	3341	3082	3054	2895
Tarefas 'POSIÇ_ESPEC' lançadas	13	18	10	12	13	15	10	10	56
Tarefas 'POSIÇ_ESPEC' executadas	13	18	10	12	13	15	10	10	56
Total de tarefas s lançadas	3365	3426	2974	3150	3106	3365	3107	3082	2964
Total de tarefas s executadas	3357	3412	2961	3137	3092	3356	3092	3064	2951

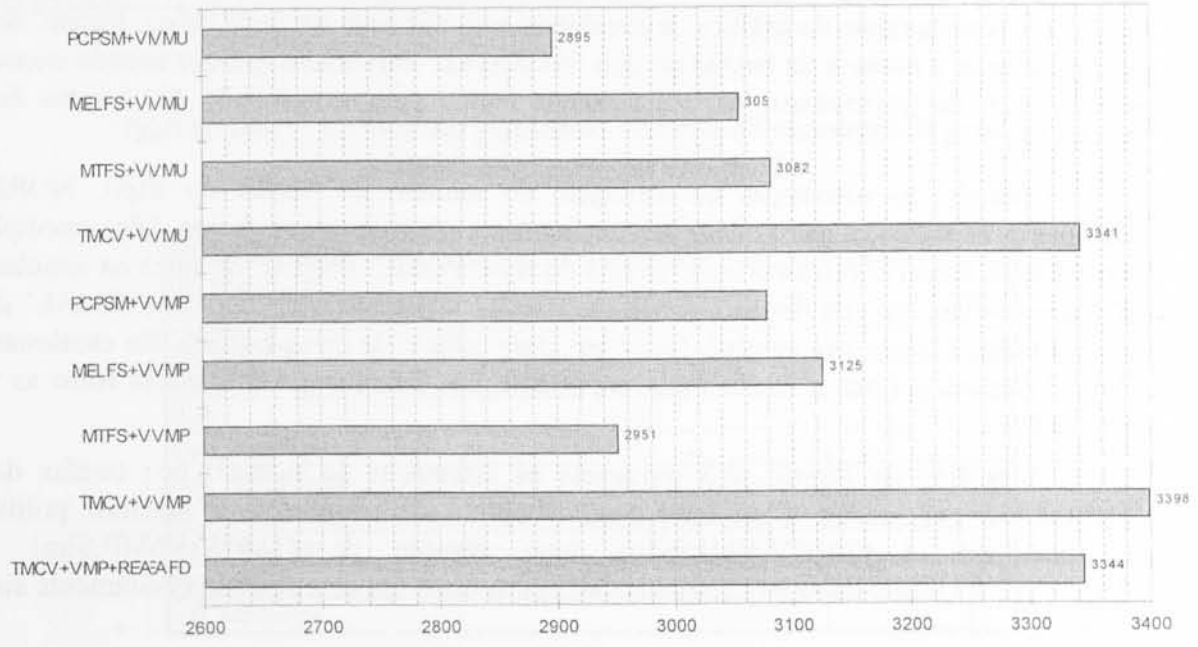


Figura 10.3 – Gráfico do número de Tarefas do tipo 'NORMAL' executadas em cada simulação.

A Tabela 10.11 mostra o número de vezes que o algoritmo de escalonamento foi executado nas várias simulações. O número de tarefas cuja execução foi adiada devido a erros ou reafectações de AGVs não é significativo. Este número é fortemente influenciado pelo número de erros nos AGVs, que implicaram a sua passagem ao modo manual (ver Tabela 10.13). As diferenças entre os dois números correspondem a tarefas que foram canceladas, porque já tinha sido executado o passo 1.

Tabela 10.11 – Estatísticas relacionadas com acções de controlo realizadas.

Simulação N.º	1	2	3	4	5	6	7	8	9
RADV	TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT	VMP					VMU			
REAFD	Sim	Não							
Número de chamadas do escalonador	3352	3404	3968	3129	3086	3351	3092	3065	2909
Número de tarefas cuja execução foi adiada, devido a reafectação ou erro no AGV	62	2	4	3	1	2	2	3	4

É de salientar que a utilização do mecanismo de reafectação dinâmica não permitiu obter qualquer melhoria de desempenho, apesar do maior peso computacional dessa estratégia de escalonamento e apesar de terem sido efectuadas 405 reafectações de tarefas (ver Tabela 10.12). Isto vem contrariar a expectativa teórica da utilidade da reafectação dinâmica, que em princípio devia permitir uma melhoria de desempenho (ver 3.5.5.2).

Não é fácil prever o desempenho dos sistemas dinâmicos de eventos discretos sem a realização de simulações, e normalmente cada caso é um caso. Este facto não permite extrapolar a outros sistemas de fabrico o resultado obtido com o mecanismo de reafectação dinâmica de tarefas. É possível que noutro sistema de fabrico essa estratégia de escalonamento permita obter melhorias de desempenho acentuadas.

Tabela 10.12 – Estatísticas relacionadas com reafectações dinâmicas de tarefas.

Simulação N.º	1	2	3	4	5	6	7	8	9
RADV	TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT	VMP					VMU			
REAFD	Sim	Não							
Número de reafectações de tarefas que implicaram o adiamento da sua execução	60								
Número de tarefas reafectadas que se mantiveram em execução	345								
Número total de reafectações efectuadas	405								

A Tabela 10.13 apresenta o número de avarias simuladas nos AGVs. Apesar da ocorrência dessas avarias não ter qualquer dependência com a estratégia de escalonamento utilizada, afecta necessariamente o desempenho obtido. Por isso, na comparação da contagem do número de tarefas executadas nas várias simulações, não se pode desprezar o efeito deste factor.

Por exemplo, o facto de terem ocorrido mais nove avarias na simulação 1 do que na simulação 2 pode ajudar a explicar o facto da reafectação dinâmica de tarefas não ter permitido obter o melhor resultado, em termos do número de tarefas executadas do tipo 'NORMAL'.

O número elevado de avarias nas simulações 3 e 9 terá contribuído fortemente para o facto de terem sido essas as simulações em que foram executadas menos tarefas, e para cavar um fosso entre o desempenho dessas duas combinações e aquelas que ficaram imediatamente melhor colocadas. Por isso, não se pode ter a certeza que as combinações de regras de despacho (MTFS;VMP;Não) e (PCPSM;VMU;Não) são as piores para o sistema de fabrico em causa. Por outro lado, apesar de terem ocorrido 41 avarias na simulação 3, a combinação de regras de despacho utilizada acabou por ser a quarta melhor, o que leva a pensar que esta combinação poderá ser colocada ao nível das três primeiras (simulações 1, 2 e 6).

Tabela 10.13 – Estatísticas relacionadas com erros ocorridos nos AGVs.

Simulação N.º	1	2	3	4	5	6	7	8	9
RADV	TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT	VMP					VMU			
REAFD	Sim	Não							
Número de ocorrências de erros nos AGVs	37	28	55	41	32	32	33	29	40
Número de erros que não implicaram a passagem ao modo manual	34	23	51	37	30	27	28	24	31
Número de erros que implicaram a passagem ao modo manual	3	5	4	4	2	5	5	5	9

A Tabela 10.14 mostra o número de conflitos ocorridos na gestão de tráfego durante as várias simulações. Em princípio, este factor não tem como causa directa a estratégia de escalonamento utilizada. Contudo, não deve ser desprezado na comparação de estratégias de escalonamento, cujo desempenho, em termos do número de tarefas executadas, tenha sido da mesma ordem de grandeza. Tal como no aspecto do número de avarias, observa-se que também este factor terá penalizado fortemente a combinação de regras de despacho da simulação 9,, devido ao elevado número de ocorrências de conflitos provocados por AGVs no estado 'PARADO'.

Tabela 10.14 – Estatísticas relacionadas com a resolução de conflitos ocorridos na gestão de tráfego.

Simulação N.º	1	2	3	4	5	6	7	8	9
RADV	TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT	VMP					VMU			
REAFD	Sim	Não							
Movimentações de AGVs no estado 'PARADO'	13	18	20	12	13	15	10	10	56
Redireccionamento de AGVs bloqueados por AGVs no estado 'ERRO'	24	9	49	33	41	18	40	40	28
Deadlocks circulares resolvidos	18	19	32	27	42	18	28	34	38

A Tabela 10.15 apresenta os resultados relativos à contagem do número de movimentos elementares (movimentos entre dois nós do *layout*) efectuados pelos AGVs.

A classificação desses movimentos em movimentos que implicaram ou não o arranque do AGV, permite obter um indicador do desempenho da gestão de tráfego. Com base nessa informação, é possível obter o gráfico da Figura 10.4. Na observação deste gráfico notam-se variações significativas no número de trajectos iniciados e no número de arranques devidos a bloqueios, ao longo das várias simulações. Estas diferenças permitem concluir que as melhores estratégias de escalonamento (em número de tarefas executadas) conduzem necessariamente a um melhor desempenho na gestão de tráfego, porque optimizam o fluxo de AGVs e evitam a ocorrência de conflitos ou de bloqueios. O desempenho na simulação 3 (MTFS;VMP;Não) é acentuadamente mais fraco, devido ao elevado número de redireccionamentos de AGVs bloqueados por AGVs no estado 'ERRO' (ver Tabela 10.14).

Outro aspecto importante, que ressalta da análise da Tabela 10.15, é o facto das três simulações em que foram executadas mais tarefas, coincidirem com as simulações em que menos movimentos elementares foram executados. Este facto é uma prova clara da optimização que a regra de despacho TMCV permite obter, ao escolher sempre o AGV que está mais próximo do ponto de carga da tarefa, minimizando dessa forma o tempo consumido na execução do passo 1 das tarefas.

Na simulação 1, o elevado número de movimentos elementares enviados e cancelados resulta da utilização do mecanismo de reafectação dinâmica de tarefas.

Tabela 10.15 – Estatísticas relacionadas com o controlo dos movimentos dos AGVs e com o desempenho da gestão de tráfego.

Simulação N.º	1	2	3	4	5	6	7	8	9
RADV	TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT	VMP					VMU			
READFD	Sim	Não							
Número de movimentos elementares que implicaram o arranque de um AGV	8444	8420	11285	9041	8893	8342	8934	8934	8488
Número de movimentos elementares correspondentes ao início da execução de um trajecto	6706	6820	5920	6268	6175	6705	6182	6124	5849
Número de movimentos elementares cuja execução foi adiada, porque o AGV ficou bloqueado	1738	1600	5365	2773	2718	1637	2752	2810	2639
Número de movimentos elementares que não implicaram o arranque de um AGV	33327	33818	51829	40176	40331	32973	40858	40060	38393
Número de movimentos elementares enviados em avanço	33635	33828	51857	40203	40291	32987	40879	40026	38421
Número de movimentos elementares enviados em avanço cancelados	306	6	29	20	29	11	22	28	17
Total de movimentos elementares iniciados	41771	42238	63114	49217	49224	41315	49792	48994	46881
Total de movimentos elementares executados	41767	42233	63106	49210	49220	41311	49786	48988	46869

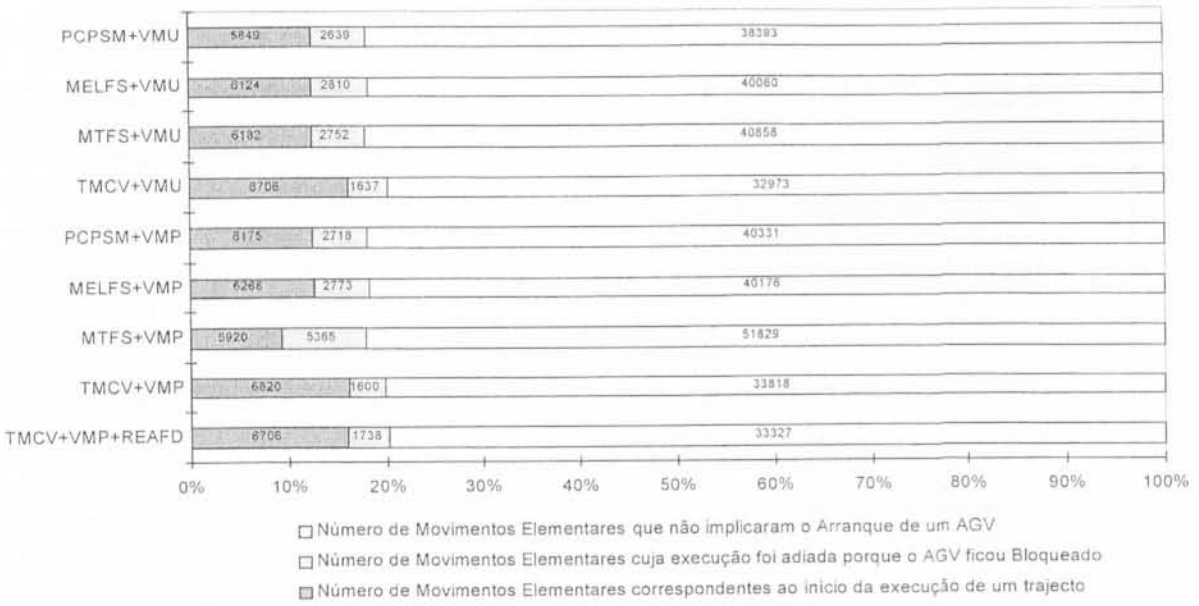


Figura 10.4 – Gráfico representativo do desempenho da gestão de tráfego.

10.3.2 Estatísticas referentes aos tempos de execução de uma tarefa

A Tabela 10.16 mostra tempos característicos de permanência de uma tarefa nos vários estados possíveis (em segundos), a saber:

- tempo mínimo,
- tempo máximo,
- tempo médio e
- desvio padrão.

Também é indicada a percentagem do tempo médio de permanência em cada estado sobre o tempo médio de execução de uma tarefa.

A distribuição média do tempo de execução total de uma tarefa pelos vários estados é ilustrada pelo gráfico da Figura 10.5. Em cada sector do gráfico é indicado o tempo médio de permanência de uma tarefa no estado respectivo (em segundos), para a simulação em causa.

Tabela 10.16 – Tempos característicos da permanência de uma tarefa nos vários estados possíveis.

Simulação N.º		1	2	3	4	5	6	7	8	9
RADV		TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT		VMP					VMU			
REAFD		Sim	Não							
FILA	Mínimo	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	Máximo	1012.040	2339.190	2620.470	955.464	1052.360	1382.560	1175.070	2137.860	1132.810
	Valor Médio	113.729	90.426	236.824	209.263	202.107	101.027	216.610	194.198	99.384
	Desvio Padrão	113.898	103.880	217.922	128.865	99.784	116.996	123.601	142.152	100.921
	% Som. V. Méd.	47.217	41.985	62.195	60.630	59.391	44.331	61.076	58.248	41.024
PASSO 1	Mínimo	0.000	0.000	2.000	2.000	2.000	2.000	0.000	0.000	0.000
	Máximo	1082.400	464.883	2351.190	694.197	805.120	673.768	1017.370	721.188	660.671
	Valor Médio	30.276	27.872	56.516	50.854	53.607	29.516	53.810	52.459	52.502
	Desvio Padrão	38.776	22.820	86.392	32.950	44.546	31.458	44.061	39.483	40.046
	% Som. V. Méd.	12.570	12.941	14.842	14.734	15.753	12.952	15.172	15.735	21.672
PASSO 2	Mínimo	18.267	18.290	18.257	18.290	18.352	18.122	18.356	18.100	18.198
	Máximo	397.903	939.407	405.281	1055.280	422.986	668.402	463.201	718.009	517.338
	Valor Médio	32.705	35.673	22.165	22.600	21.778	34.585	21.650	21.948	27.398
	Desvio Padrão	37.503	45.989	17.370	30.907	15.732	42.029	14.587	18.961	29.562
	% Som. V. Méd.	13.578	16.563	5.821	6.548	6.400	15.176	6.104	6.583	11.310
PASSO 3	Mínimo	2.400	2.400	2.400	2.400	2.400	2.400	2.400	2.400	2.400
	Máximo	1164.860	463.637	2206.680	624.381	1049.030	939.901	681.663	1096.360	639.289
	Valor Médio	43.388	41.411	44.960	41.321	42.429	42.630	42.102	43.923	42.317
	Desvio Padrão	47.985	31.101	74.921	41.883	41.818	44.583	38.270	49.029	38.643
	% Som. V. Méd.	18.014	19.227	11.807	11.972	12.468	18.706	11.871	13.174	17.468
PASSO 4	Mínimo	18.278	18.168	18.239	18.305	18.478	18.131	18.267	18.131	18.491
	Máximo	1400.550	21.949	462.957	974.089	486.026	491.317	513.937	1101.710	822.643
	Valor Médio	20.765	19.994	20.310	21.109	20.376	20.134	20.485	20.871	20.655
	Desvio Padrão	26.189	0.504	10.202	26.321	11.564	8.180	12.422	26.656	18.589
	% Som. V. Méd.	8.621	9.283	5.334	6.116	5.988	8.835	5.776	6.260	8.526

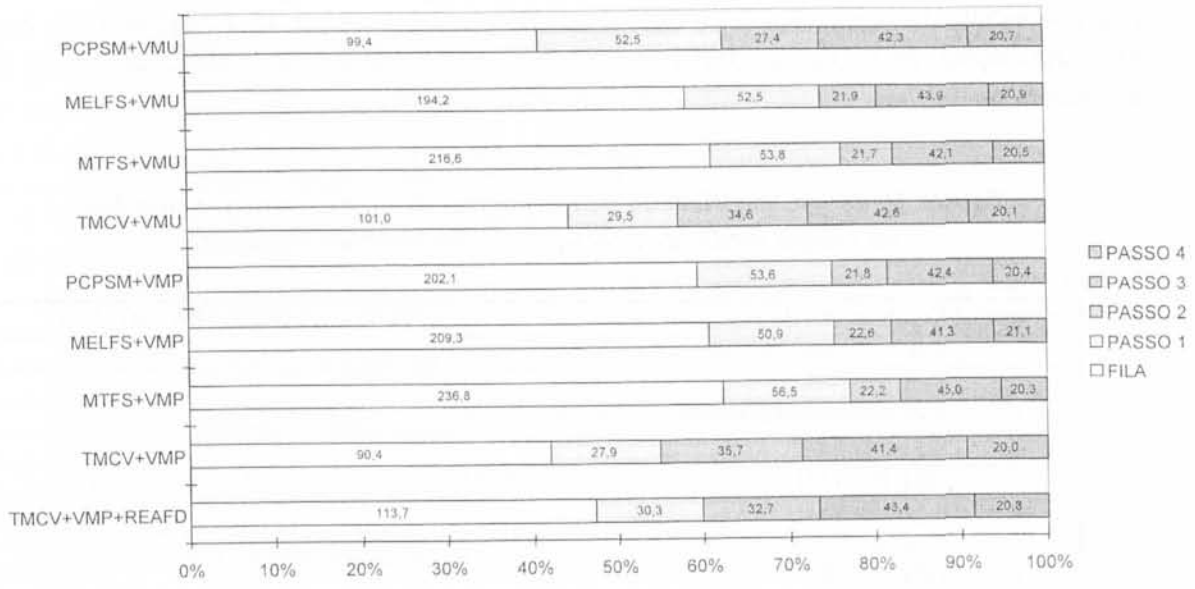


Figura 10.5 – Distribuição média dos tempos de permanência de uma tarefa em cada estado.

Da análise da Tabela 10.16 e do gráfico da Figura 10.5, conclui-se que as variações mais significativas dizem respeito ao tempo de permanência nos estados 'FILA' e 'PASSO_1'. As diferenças verificadas nestes dois indicadores permitem inferir sobre a qualidade do desempenho das várias estratégias de escalonamento, sem se estar condicionado por fenómenos probabilísticos, como acontece em 10.3.1 com o número de avarias dos AGVs. Por isso, esta comparação será mais fidedigna do que a que foi aí efectuada.

A Tabela 10.17 classifica as combinações de regras de despacho, tomando como base o tempo médio de atendimento de uma tarefa (menor tempo de permanência no estado 'FILA').

Tal como em 10.3.1, conclui-se que a estratégia de escalonamento melhor colocada é (TMCV;VMP;Não), seguida de perto por (PCPSM;VMU;Não) e (TMCV;VMU;Não). O facto da regra PCPSM ter permitido a obtenção de um bom resultado, justifica-se pelo facto do seu princípio se basear em atender as tarefas de movimentação pela sua ordem de chegada, contribuindo por isso para a diminuição do tempo de atendimento de uma tarefa.

Tabela 10.17 – Classificação de estratégias de escalonamento com base no tempo médio de atendimento de uma tarefa.

Pos.	Combinação de regras de despacho	Tempo no estado 'FILA' (s)
1	TMCV+VMP	90.4
2	PCPSM+VMU	99.4
3	TMCV+VMU	101.0
4	TMCV+VMP+REAFD	113.7
5	MELFS+VMU	194.2
6	PCPSM+VMP	202.1
7	MELFS+VMP	209.3
8	MTFS+VMU	216.6
9	MTFS+VMP	236.8

A Tabela 10.18 classifica as combinações de regras de despacho, tomando como base o tempo médio de permanência de uma tarefa no estado 'PASSO_1'. Como seria de esperar, as combinações de regras de despacho que incluem a regra TMCV destacam-se de forma acentuada de todas as outras.

Tabela 10.18 – Classificação de estratégias de escalonamento com base no tempo médio de execução do passo 1 de uma tarefa.

Pos.	Combinação de regras de despacho	Tempo no estado 'PASSO_1' (s)
1	TMCV+VMP	27.9
2	TMCV+VMU	29.5
3	TMCV+VMP+REAFD	30.3
4	MELFS+VMP	50.9
5	MELFS+VMU	52.5
6	PCPSM+VMU	52.5
7	PCPSM+VMP	53.6
8	MTFS+VMU	53.8
9	MTFS+VMP	56.5

A Tabela 10.19 apresenta os valores característicos do tempo total de operação e do tempo total de execução de uma tarefa. O primeiro não inclui o tempo de permanência da tarefa no estado 'PASSO_1'. O valor médio desses tempos está representado no gráfico da Figura 10.6.

Tabela 10.19 – Tempos totais de execução de uma tarefa.

Simulação N.º		1	2	3	4	5	6	7	8	9
RADV		TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT		VMP					VMU			
REAFD		Sim	Não							
Tempo de Operação Total (s)										
PASSO1 até PASSO 4	Mínimo	44.521	43.403	45.480	44.799	46.347	44.632	43.870	45.138	0.000
	Máximo	1529.600	1047.920	2474.330	1184.720	1139.430	1020.180	1076.600	1206.290	907.888
	Valor Médio	127.135	124.951	143.951	135.884	138.190	126.866	138.047	139.202	142.871
	Desvio Padrão	78.800	63.848	118.218	67.558	65.175	72.690	62.240	74.073	69.684
Tempo de Execução Total (s)										
FILA até PASSO 4	Mínimo	51.186	47.801	51.799	52.096	51.799	49.012	45.138	45.138	45.138
	Máximo	1633.220	2435.090	2808.460	1369.780	1371.090	1478.490	1406.260	2400.030	1481.500
	Valor Médio	240.863	215.376	380.775	345.147	340.298	227.893	354.657	333.400	242.255
	Desvio Padrão	133.658	113.899	262.619	151.232	121.269	130.099	147.346	170.097	125.704

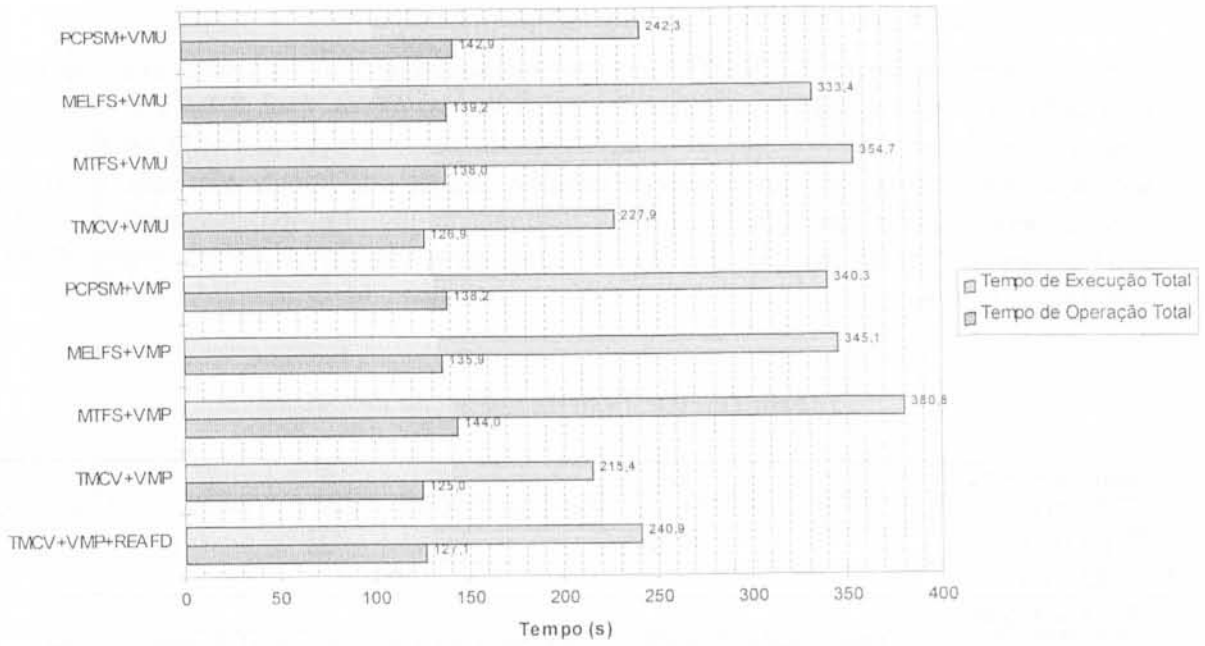


Figura 10.6 – Valor médio do tempo de execução total e do tempo de operação total de uma tarefa.

Da análise da tabela e do gráfico anterior, observa-se que não há uma variação significativa no tempo médio de operação total de uma tarefa, embora haja grandes diferenças no tempo médio total de execução. Este resultado leva a concluir que as diferenças de desempenho das várias estratégias de escalonamento são mais influenciadas pelo tempo de atendimento da tarefa (tempo de permanência no estado 'FILA') do que pelo tempo e permanência no estado 'PASSO_1'.

10.3.3 Estatísticas relacionadas com os AGVs

A Tabela 10.20 mostra os valores característicos (valor final, valor médio e desvio padrão) da taxa de utilização dos AGVs para as simulações efectuadas. A taxa de utilização (TU) de um AGV, num dado instante, é dada pela Equação 10.1. Os tempos intervenientes são os tempos acumulados nos respectivos estados.

$$TU = \frac{t_{EX_MOVIM} + t_{CARGA} + t_{DESCARGA}}{t_{PARADO} + t_{BLOQUEADO} + t_{ERRO} + t_{EX_MOVIM} + t_{CARGA} + t_{DESCARGA}} \qquad (10.1)$$

Observando os valores da Tabela 10.20, observa-se que não há variações significativas nos valores da TU entre as várias simulações, exceptuando o valor médio da TU na simulação 3.

A TU depende sobretudo do número de erros ocorridos nos AGVs e do bom desempenho da gestão de tráfego ('$t_{BLOQUEADO}$'), sendo pouco influenciada pela estratégia de escalonamento utilizada. Como não há grandes variações no desempenho da gestão de tráfego entre as várias simulações, e como 't_{ERRO}' não é significativo quando comparado com os

outros tempos envolvidos, apesar de haver variações no número de erros ocorridos nos AGVs, a TU não apresenta grandes variações entre as várias simulações.

A regra de despacho VMU favorece intrinsecamente o balanceamento da taxa de utilização dos vários AGVs constituintes da frota de AGVs (ver 3.5.4.1.3). Contudo, o desempenho do escalonador é muito pouco afectado pela regra de despacho escolhida para a afectação desencadeada por um centro de trabalho, porque este tipo de afectação ocorre muito pontualmente. Assim, não é possível notar qualquer melhoria no balanceamento da TU dos AGVs (desvio padrão da TU dos AGVs), resultante da utilização da regra VMU nas simulações 6,, 7, 8 e 9.

Tabela 10.20 – Taxa de utilização dos AGVs.

Simulação N.º	1	2	3	4	5	6	7	8	9
RADV	TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT	VMP					VMU			
REAFD	Sim	Não							
Taxa de Utilização (%)									
Valor Final	90.860	93.861	87.936	90.074	90.076	91.958	91.340	90.964	90.374
Valor Médio	87.324	90.103	83.225	88.239	88.888	87.075	87.859	87.533	88.365
Desvio Padrão	7.099	6.197	5.332	5.075	4.956	6.393	5.672	5.252	5.480

O gráfico da Figura 10.7 mostra a evolução da taxa de utilização do AGV 1 da simulação 1, ao longo de um dia de trabalho. Essa é a evolução típica da TU de um AGV.

Como no início da simulação o número de AGVs e o número de tarefas pendentes é da mesma ordem de grandeza, o tempo 't_{ESPERA}' assume particular relevância, degradando o valor da TU. À medida que o tempo avança, começa a haver mais tarefas pendentes do que AGVs, e esse tempo começa a ter pouca relevância face aos outros tempos "úteis" (tempos relativos à execução de movimentos). Assim, a taxa de utilização sobe até um valor de aproximadamente 90 %, estabilizando em torno desse valor, com pequenas oscilações introduzidas principalmemte pelos tempos '$t_{BLOQUEADO}$' e 't_{ERRO}'.

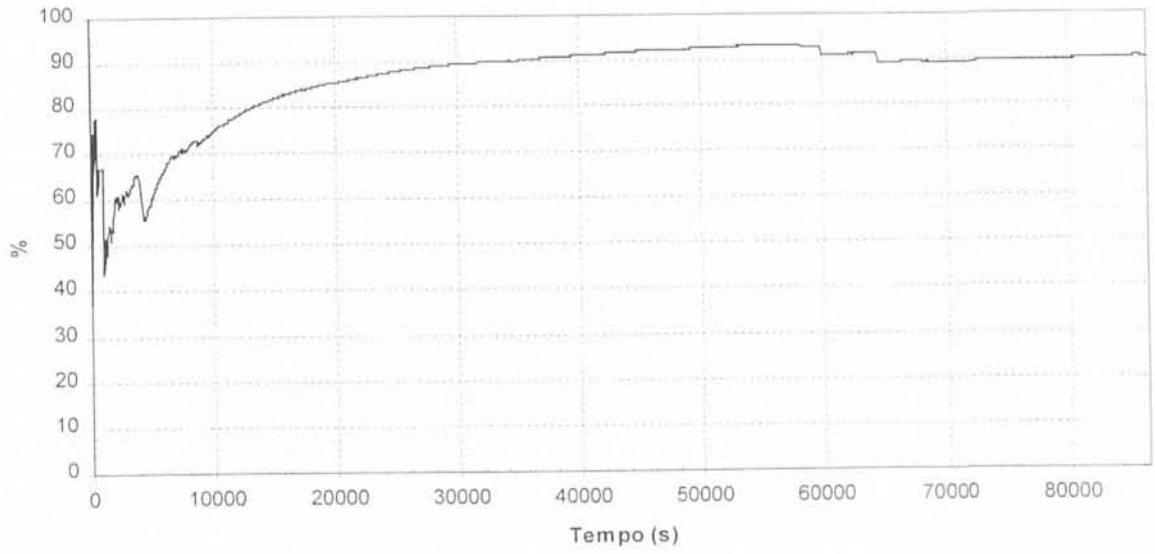


Figura 10.7 – Gráfico da evolução da taxa de utilização de um AGV ao longo de um dia de trabalho.

Tal como para as tarefas, a Tabela 10.21 mostra tempos característicos de permanência de uma tarefa nos vários estados possíveis (em segundos), bem como a percentagem do tempo gasto em cada estado sobre o tempo médio de execução de uma tarefa.

A distribuição média do tempo de simulação pelos tempos de permanência de um AGV em cada estado é ilustrada pelo gráfico da Figura 10.8. Em cada sector do gráfico é indicado o tempo médio de permanência de um AGV no estado respectivo (em segundos), para a simulação em causa.

Tabela 10.21 – Tempos característicos da permanência dos AGVs em cada estado.

Simulação N.º		1	2	3	4	5	6	7	8	9
RADV		TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT		VMP					VMU			
REAFD		Sim	Não							
Disabled	Somatório	5.850	5.850	5.850	5.850	5.850	5.850	5.850	5.850	5.850
	Valor Médio	1.170	1.170	1.170	1.170	1.170	1.170	1.170	1.170	1.170
	Desvio Padrão	1.235	1.235	1.235	1.235	1.235	1.235	1.235	1.235	1.235
	% Somn. V. Méd.	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Enabled	Somatório	647.107	643.569	904.899	962.067	793.582	799.453	868.553	893.507	1115.059
	Valor Médio	129.421	128.714	180.980	192.413	158.716	159.891	173.711	178.701	223.012
	Desvio Padrão	136.010	144.396	132.342	213.396	159.452	146.283	124.158	157.676	195.405
	% Somn. V. Méd.	0.150	0.149	0.209	0.223	0.184	0.185	0.201	0.207	0.258
Espera	Somatório	5920.475	6024.006	5510.068	5420.227	4898.781	6361.310	4868.228	4809.134	15707.790
	Valor Médio	1184.095	1204.801	1102.014	1084.045	979.756	1272.262	973.646	961.827	3141.558
	Desvio Padrão	254.305	194.079	257.007	301.566	200.370	78.002	47.305	46.384	321.797
	% Somn. V. Méd.	1.370	1.394	1.275	1.255	1.134	1.473	1.127	1.113	3.636
Bloqueado	Somatório	24607.680	15099.050	35580.510	26411.840	28345.660	19585.070	24700.960	25874.380	18852.380
	Valor Médio	4921.536	3019.810	7116.102	5282.368	5669.132	3917.014	4940.192	5174.876	3770.476
	Desvio Padrão	649.826	123.501	568.213	653.287	369.464	441.092	582.223	1087.384	457.403
	% Somn. V. Méd.	5.696	3.495	8.236	6.114	6.561	4.534	5.718	5.989	4.364
Movimento Descarr.	Somatório	88600.401	88011.398	142362.69	144868.60	146104.30	89242.901	148864.79	146083.19	140517.35
	Valor Médio	17720.080	17602.280	28472.540	28973.720	29220.860	17848.580	29772.960	29216.640	28103.470
	Desvio Padrão	450.063	297.416	285.827	626.880	795.719	224.727	599.684	444.658	841.660
	% Somn. V. Méd.	20.509	20.373	32.954	33.534	33.820	20.658	34.459	33.816	32.527
Carga	Somatório	107960.20	118342.80	64541.400	67471.800	66551.700	112898.27	66028.100	66073.400	78555.500
	Valor Médio	21592.041	23668.560	12908.280	13494.360	13310.340	22579.655	13205.620	13214.680	15711.100
	Desvio Padrão	1115.131	973.628	151.152	208.547	191.404	761.186	416.833	279.335	572.613
	% Somn. V. Méd.	24.991	27.394	14.940	15.618	15.405	26.134	15.284	15.295	18.184
Movimento Carregado	Somatório	128566.29	130635.30	113167.50	113414.00	116842.80	127566.80	117275.69	118897.60	112434.10
	Valor Médio	25713.260	26127.060	22633.500	22682.801	23368.560	25513.360	23455.140	23779.520	22486.820
	Desvio Padrão	660.827	240.628	661.250	679.341	616.581	379.239	788.506	595.296	372.275
	% Somn. V. Méd.	29.761	30.240	26.196	26.253	27.047	29.529	27.147	27.523	26.026
Descarga	Somatório	66799.200	67878.300	59006.900	62492.300	61624.200	66808.200	61625.500	61095.300	57894.800
	Valor Médio	13359.840	13575.660	11801.380	12498.460	12324.840	13361.640	12325.100	12219.060	11578.960
	Desvio Padrão	254.901	302.949	213.607	147.359	240.181	173.154	167.207	82.855	213.609
	% Somn. V. Méd.	15.463	15.713	13.659	14.466	14.265	15.465	14.265	14.142	13.402
Erro	Somatório	8892.743	5359.653	10920.180	10953.120	6833.063	8731.910	7762.420	8267.702	6917.120
	Valor Médio	1778.549	1071.931	2184.036	2190.624	1366.613	1746.382	1552.484	1653.540	1383.424
	Desvio Padrão	725.522	744.265	770.341	609.877	403.438	434.815	874.535	1357.417	665.263
	% Somn. V. Méd.	2.059	1.241	2.528	2.535	1.582	2.021	1.797	1.914	1.601

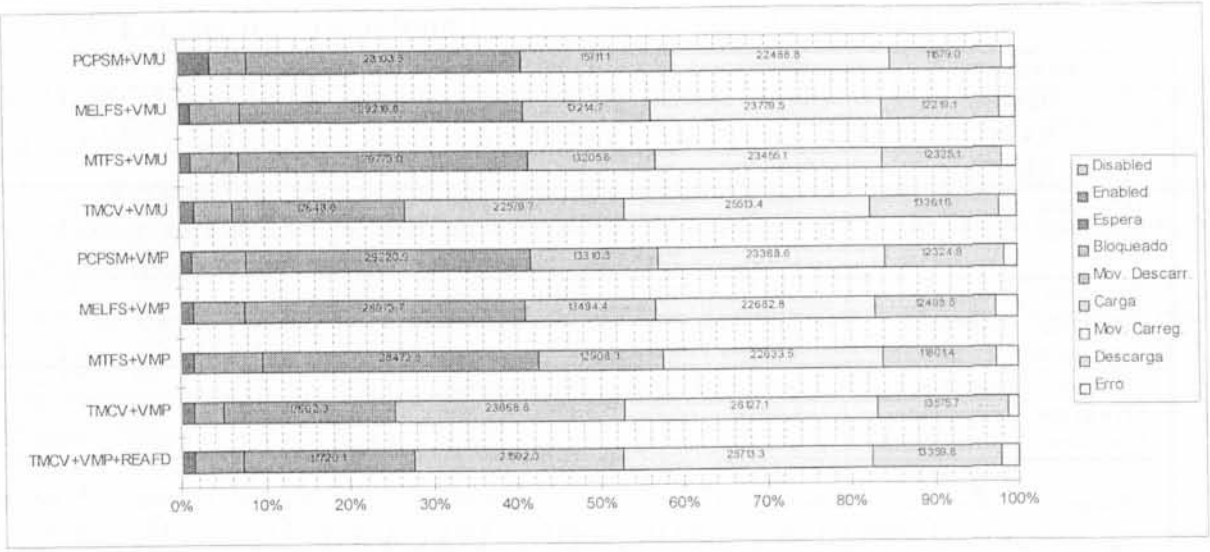


Figura 10.8 – Distribuição média do tempo de permanência de um AGV em cada estado.

Da análise da Tabela 10.21 e do gráfico da Figura 10.8, conclui-se que as maiores variações na distribuição do tempo de simulação pelo tempo de permanência dos AGVs devem-se ao tempo gasto pelos AGVs na execução de movimentos sem carga, que corresponde à execução do passo 1 das tarefas. Este aspecto é uma consequência dos resultados ilustrados pelo gráfico da Figura 10.5.

O tempo gasto nos estados 'DISABLED' e 'ENABLED' são muito pequenos, quando comparados com os restantes tempos. Por isso, não é possível visualiza-los nas barras do gráfico da Figura 10.8.

A Tabela 10.22 mostra estatísticas sobre o número de movimentos de cada tipo efectuados pelos AGVs (somatório, valor médio por AGV e desvio padrão).

Um aspecto importante que ressalta da sua análise é a razão entre o número de trajectos efectuados e o número de arranques respectivos. Este coeficiente é um indicador importante do desempenho da gestão de tráfego. O gráfico da Figura 10.9 ilustra as variações desse coeficiente ao longo das várias simulações. Comparando este gráfico com o da Figura 10.3, conclui-se que as melhores estratégias de escalonamento, em termos do número de tarefas executadas, são as que têm os mais baixos coeficientes do número de arranques por trajecto.

Tabela 10.22 – Estatísticas dos movimentos efectuados pelos AGVs.

Simulação N.º		1	2	3	4	5	6	7	8	9
RADV		TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT		VMP					VMU			
REAFD		Sim	Não							
Arranques Descarreg.	Somatório	4117	4096	4312	4696	4740	4074	4696	4675	4387
	Valor Médio	823.400	819.200	862.400	939.200	948.000	814.800	939.200	935.000	877.400
	Desvio Padrão	11.104	24.098	24.846	22.819	21.691	5.891	20.130	17.635	17.827
Trajectos Descarreg.	Somatório	3362	3424	2967	3142	3095	3362	3098	3069	2953
	Valor Médio	672.400	684.800	593.400	628.400	619.000	672.400	619.600	613.800	590.600
	Desvio Padrão	13.686	15.991	9.813	8.414	11.597	8.820	10.574	5.975	6.504
Média Arranques Traj. Desc.	Somatório	6.124	5.981	7.267	7.472	7.660	6.060	7.579	7.616	7.430
	Valor Médio	1.225	1.196	1.453	1.494	1.532	1.212	1.516	1.523	1.486
	Desvio Padrão	0.023	0.014	0.034	0.020	0.049	0.017	0.022	0.022	0.045
Cargas	Somatório	3346	3399	2957	3129	3081	3345	3088	3060	2901
	Valor Médio	669.200	679.800	591.400	625.800	616.200	669.000	617.600	612.000	580.200
	Desvio Padrão	13.217	14.822	10.065	7.563	11.167	8.631	9.182	4.243	10.257
Arranques Carregado	Somatório	4221	4246	3888	4091	4032	4172	4030	4128	3897
	Valor Médio	844.200	849.200	777.600	818.200	806.400	834.400	806.000	825.600	779.400
	Desvio Padrão	30.768	27.133	14.876	18.966	20.900	16.334	22.858	20.342	19.243
Trajectos Carregado	Somatório	3344	3396	2953	3126	3080	3343	3084	3055	2896
	Valor Médio	668.800	679.200	590.600	625.200	616.000	668.600	616.800	611.000	579.200
	Desvio Padrão	13.103	14.990	10.262	7.596	11.068	8.933	8.701	4.062	10.616
Média Arranques Traj. Carreg.	Somatório	6.310	6.251	6.584	6.544	6.546	6.240	6.533	6.756	6.728
	Valor Médio	1.262	1.250	1.317	1.309	1.309	1.248	1.307	1.351	1.346
	Desvio Padrão	0.022	0.018	0.020	0.030	0.030	0.009	0.022	0.026	0.010
Descargas	Somatório	3344	3398	2951	3125	3079	3341	3082	3054	2895
	Valor Médio	668.800	679.600	590.200	625.000	615.800	668.200	616.400	610.800	579.000
	Desvio Padrão	13.103	15.176	10.474	7.649	11.411	9.257	8.562	4.025	10.770

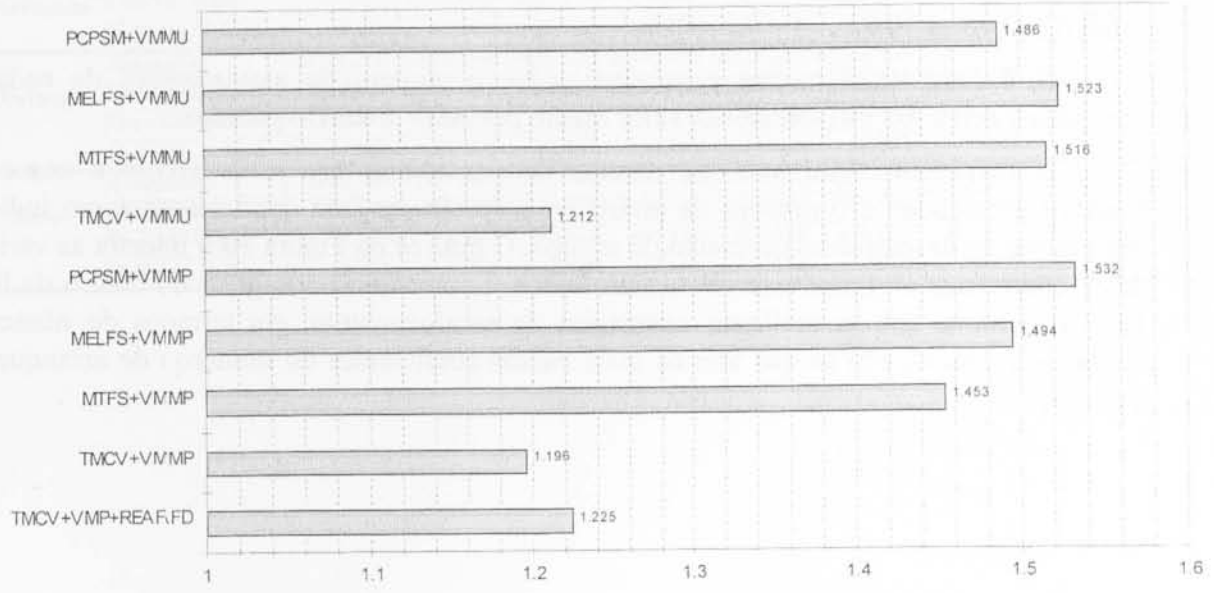


Figura 10.9 – Gráfico com a média de arranques por trajecto para um AGV descarregado em cada uma das simulações efectuadas.

10.3.4 Estatísticas relacionadas com as células da planta fabril

A Tabela 10.23 mostra o número de paletes recebidas e o número de paletes distribuídas à fábrica pelo cais de recepção de matéria-prima (CRMP).

O número de paletes recebidas não pode ser utilizado como um índice de desempenho do sistema, porque depende apenas do tempo médio entre a recepção de lotes consecutivos no cais (parâmetro 'tm_entre_lotes').

O número de paletes distribuídas depende do desempenho do SAGVs. Como se pode observar na tabela, as variações verificadas ao longo das várias simulações estão correlacionadas com o número de tarefas do tipo 'NORMAL' executadas (ver gráfico da Figura 10.3).

Tabela 10.23 – Número de paletes do CRMP.

Simulação N.º		1	2	3	4	5	6	7	8	9
RADV		TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT		VMP					VMU			
REAFD		Sim	Não							
Recebidas	Total	900	600	1200	2100	900	900	600	900	900
Distribuídas	Total	493	499	431	452	446	474	445	437	492

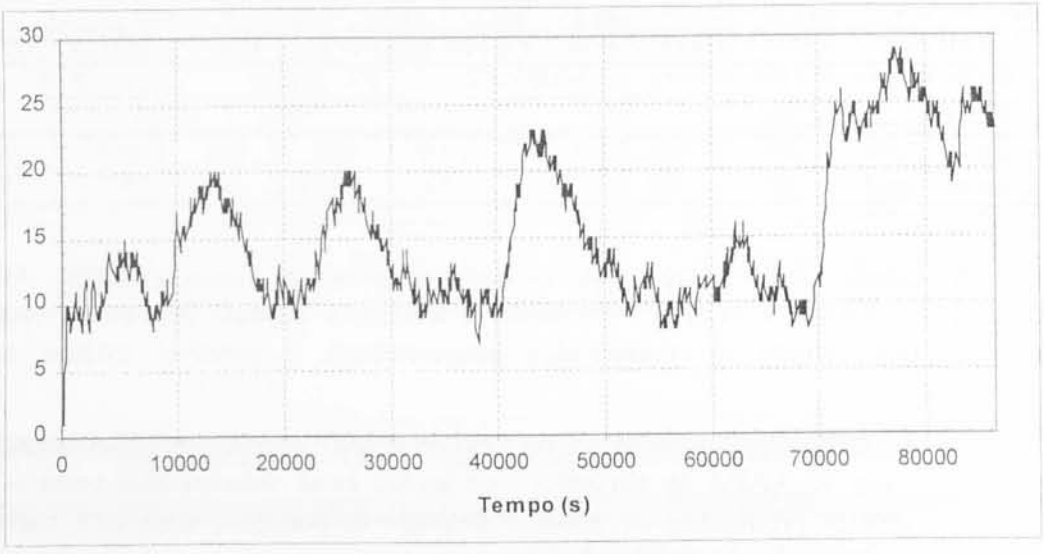
A Tabela 10.24 mostra valores característicos do funcionamento dos armazéns automáticos (AAMP e AAPA). Cruzando a tendência seguida por esses valores com a tendência dos valores já apresentados anteriormente, é possível extrair as seguintes conclusões:

- total de paletes recebidas – no caso do AAMP não há variações significativas; no caso do AAPA as variações verificadas estão relacionadas com o número de paletes produzidas de produto acabado e, portanto, com um melhor ou pior desempenho do sistema de fabrico;
- média do número de paletes armazenadas – quanto melhor for o desempenho do sistema, menor é o nível de ocupação média no armazém de matéria-prima (maior utilização de matéria-prima) e maior é o nível de ocupação média no armazém (maior produção de produto acabado);
- ocupação máxima dos armazéns (máximo de paletes existentes) – um menor pico de ocupação máxima no AAMP e um maior pico de ocupação máxima no AAPA são indicadores de um bom desempenho do sistema de fabrico;
- total de paletes distribuídas.– esta grandeza possui a mesma tendência em ambos os armazéns; quanto mais elevado for o número de paletes distribuídas, melhor é o desempenho do sistema.

O gráfico da Figura 10.10 mostra a evolução do número de paletes existentes no AAMP (número de paletes armazenadas adicionado do número de paletes nos *buffers* de entrada e de saída) ao longo da simulação 1.

Tabela 10.24 – Número de paletes dos armazéns.

Simulação N.º		1	2	3	4	5	6	7	8	9
RADV		TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCIT		VMP					VMU			
REAFDD		Sim	Não							
Armazém Automático de Matéria-Prima (AAMP)										
Recebidas	Total	416	425	417	431	420	422	421	416	420
Armazenadas	Média	11.585	8.718	21.424	14.517	10.248	15.881	14.359	13.048	9.619
Existentes	Máximo	29	25	35	30	20	34	29	28	23
Distribuídas	Total	400	403	391	403	408	410	401	405	399
Armazém Automático de Produto Acabado (AAPA)										
Recebidas	Total	385	403	348	361	345	391	369	370	366
Armazenadas	Média	39.134	38.875	32.413	33.142	37.373	39.369	36.630	39.488	31.433
Existentes	Máximo	82	82	67	69	76	81	74	80	84
Distribuídas	Total	335	354	316	350	301	345	330	324	315



Figgura 10.10 – Número de paletes existentes no AAMP ao longo de uma simulação.

A Tabbela 10.25 mostra os valores do WIP (em-curso-de-fabrico) médio e do total de paletes proceessadas em cada célula de produção manual ou automatizada, na célula de montagem e ı no centro de controle de qualidade.

Um nñível mais baixo do WIP médio e um número de paletes processadas mais elevado são indicadoores de um melhor desempenho do sistema de fabrico. O centro de controle de qualidade é ı uma excepção a esta afirmação, uma vez que um maior valor do WIP médio é acompanhaddo de um maior número de paletes de produto acabado processadas, ou seja, de um melhor ddesempenho do sistema.

Tabela 10.25 – Número de paletes dos centros de maquinação, postos de trabalho manuais, célula de montagem e centro de controle de qualidade.

Simulação N.º		1	2	3	4	5	6	7	8	9
RADV		TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT		VMP					VMU			
REAFD		Sim	Não							
Centro Maq. 1	WIP Médio	1.683	1.681	1.593	1.282	1.560	1.630	1.621	1.297	1.589
(CM1)	Processadas	228	234	171	182	193	229	183	189	235
Centro Maq. 2	WIP Médio	2.577	2.523	2.641	2.624	2.555	1.955	2.703	2.540	2.011
(CM2)	Processadas	238	240	230	239	238	230	239	234	232
Centro Maq. 3	WIP Médio	1.277	1.082	1.210	1.021	1.064	1.062	1.154	1.115	1.021
(CM3)	Processadas	125	118	87	79	88	108	88	93	112
Centro Maq. 4	WIP Médio	1.908	1.952	1.871	2.064	1.737	2.133	1.814	2.309	1.771
(CM4)	Processadas	119	120	108	128	107	123	109	86	119
Posto Man. 1	WIP Médio	0.948	1.016	1.137	1.049	1.079	0.968	0.959	1.194	0.966
(PM1)	Processadas	107	145	82	88	101	103	82	92	98
Posto Man. 2	WIP Médio	1.022	0.744	1.118	1.094	1.066	1.046	1.130	1.206	1.096
(PM2)	Processadas	119	87	87	92	90	124	99	95	135
Posto Man. 3	WIP Médio	1.407	1.691	1.533	1.739	1.577	1.291	1.609	1.596	1.453
(PM3)	Processadas	161	215	153	171	169	177	167	161	179
Posto Man. 4	WIP Médio	2.843	3.010	1.666	1.725	1.640	1.603	1.642	1.661	2.998
(PM4)	Processadas	176	131	157	166	169	171	160	163	166
Posto Man. 5	WIP Médio	3.852	3.433	2.494	2.505	2.613	3.267	2.621	2.755	3.554
(PM5)	Processadas	226	228	198	209	214	223	208	207	225
Posto Man. 6	WIP Médio	3.395	4.163	2.449	2.512	2.518	3.461	2.738	2.670	3.567
(PM6)	Processadas	226	223	193	203	208	225	202	204	224
C. Montagem	WIP Médio	2.471	2.461	2.470	2.461	2.317	1.427	2.385	2.550	2.313
(CM)	Processadas	109	108	93	92	103	106	98	105	108
Contr. Qualid.	WIP Médio	3.343	3.432	2.843	2.159	3.241	3.433	2.858	2.390	2.535
(CQ)	Processadas	290	305	267	284	253	299	283	275	267

A Tabela 10.26 mostra o número de paletes recebidas e o número de paletes expedidas pelo cais de expedição de produto acabado. Ambas as grandezas são essencialmente influenciadas pelo tempo médio entre a expedição de dois lotes consecutivos (parâmetro 'tm_entre_carreg'), uma vez que o número de paletes expedidas depende directamente desse factor, e porque o transporte de paletes, desde os *buffers* de entrada até ao cais de expedição, só é permitido aquando da expedição de um novo lote (ver pressupostos de implementação do CEPA em 9.2.4). Por isso, essas duas grandezas não constituem indicadores do desempenho do sistema de fabrico.

Tabela 10.26 – Número de paletes do CEPA.

Simulação N.º		1	2	3	4	5	6	7	8	9
RADV		TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT		VMP					VMU			
REAFD		Sim	Não							
Recebidas	Total	55	55	55	74	55	55	55	55	55
Expedidas	Total	50	50	50	50	50	50	50	50	50

A Tabela 10.27 apresenta os valores do comprimento médio das filas nos *buffers* de entrada das várias células do sistema de fabrico. O valor médio do comprimento das filas é tanto maior quanto maior for a actividade do sistema de fabrico, devida a uma estratégia de escalonamento mais eficiente no SAGVs.

Tabela 10.27 – Comprimento médio das filas dos pontos de descarga.

Simulação N.º		1	2	3	4	5	6	7	8	9
RADV		TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT		VMP					VMU			
REAFID		Sim	Não							
Armaz. Aut. Matéria-Prima		3.021	3.206	2.924	3.250	2.650	3.062	3.135	3.182	3.209
Armaz. Aut. Prodt. Acabado		0.942	1.011	0.794	0.813	0.624	1.033	0.806	0.858	0.546
Centro Maquinação 1		0.499	0.500	0.499	0.499	0.499	0.499	0.499	0.499	0.499
Centro Maquinação 2		1.239	1.240	0.728	0.710	0.896	0.728	0.766	0.771	0.761
Centro Maquinação 3		0.498	0.506	0.497	0.497	0.497	0.507	0.497	0.497	0.498
Centro Maquinação 4		0.582	0.574	0.549	0.533	0.554	0.587	0.534	0.783	0.567
Posto Trabalho Manual 1		0.498	0.498	0.497	0.497	0.498	0.500	0.497	0.497	0.497
Posto Trabalho Manual 2		0.498	0.497	0.497	0.497	0.500	0.498	0.498	0.497	0.498
Posto Trabalho Manual 3		0.542	0.600	0.498	0.569	0.504	0.521	0.522	0.517	0.510
Posto Trabalho Manual 4		0.612	0.736	0.524	0.522	0.534	0.545	0.505	0.511	0.708
Posto Trabalho Manual 5		2.399	2.052	0.840	0.855	1.053	1.889	0.957	1.112	2.140
Posto Trabalho Manual 6		1.758	2.645	0.797	0.797	0.950	1.873	1.123	1.024	2.119
Célula de Montagem		0.635	0.635	0.605	0.664	0.617	0.622	0.600	0.678	0.606
Controle de Qualidade		1.758	0.865	0.562	0.538	0.554	0.865	0.567	0.564	0.496
Cais Exped. Prodt. Acabado		0.309	0.242	0.265	0.236	0.226	0.286	0.318	0.254	0.338
Paletes numa fila de entrada	Média	1.053	1.054	0.738	0.765	0.744	0.934	0.788	0.816	0.933
	Desv. Padrão	0.818	0.880	0.623	0.705	0.567	0.760	0.682	0.693	0.845
Média do número total de paletes nas filas entrada		21.913	22.108	16.814	17.635	16.442	20.227	17.817	18.674	19.863

A Tabela 10.28 apresenta os valores do comprimento médio das filas nos *buffers* de saída das várias células do sistema de fabrico. O valor médio do comprimento das filas é tanto maior quanto maior for a actividade do sistema de fabrico, devida a uma estratégia de escalonamento mais eficiente no SAGVs.

Comparando o desvio padrão do número de paletes médio de uma fila de saída ao longo das várias simulações, é possível concluir que as estratégias de escalonamento definidas à custa das regras de despacho MTFS, MELFS e PCPSM permitem obter um maior nivelamento do nível de ocupação dos *buffers* de saída das células (menor valor do desvio padrão do número de paletes médio num *buffer* de saída), embora o desempenho do sistema seja pior, do ponto de vista do número de tarefas executadas pelo SAGVs, e da quantidade de produto acabado produzida. A regra de despacho MELFS é a que permite o maior nivelamento.

O maior nivelamento permitido pelas regras MTFS e MELFS resulta do facto das decisões tomadas, com base nessas regras, se basearem no estado da ocupação das filas de saída (ver 3.5.4.2.2 e 3.5.4.2.3). A política definida pela regra PCPSM acaba também por favorecer indirectamente o nivelamento da ocupação das filas de saída (embora não seja neste aspecto tão eficiente como as anteriores), porque o SAGVs procura atender as tarefas de movimentação pela ordem de chegada (ver 3.5.4.2.4), evitando que as paletes respectivas permaneçam durante muito tempo nas filas de saída e, consequentemente, contribuindo para a diminuição do comprimento das filas.

Tabela 10.28 – Comprimento médio das filas dos pontos de carga.

Simulação N.º		1	2	3	4	5	6	7	8	9
RADV		TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT		VMP					VMU			
REAFD		Sim	Não							
Cais Recep. Matéria-Prima		3.887	3.895	3.882	3.888	3.886	3.904	3.903	3.901	3.912
Armaz. Aut. Matéria-Prima		0.509	0.514	0.545	0.537	0.560	0.524	0.535	0.551	0.499
Armaz. Aut. Prod. Acabado		0.516	0.502	0.521	0.568	0.645	0.511	0.605	0.505	0.599
Centro Maquinação 1		0.705	0.705	0.563	0.516	0.535	0.638	0.637	0.511	0.526
Centro Maquinação 2		0.579	0.705	1.102	1.107	0.733	0.555	1.103	0.910	0.559
Centro Maquinação 3		0.594	0.523	0.669	0.572	0.497	0.507	0.565	0.575	0.498
Centro Maquinação 4		0.626	0.589	0.571	0.609	0.507	0.660	0.589	0.717	0.508
Posto Trabalho Manual 1		0.507	0.500	0.533	0.568	0.557	0.498	0.524	0.652	0.520
Posto Trabalho Manual 2		0.507	0.497	0.554	0.670	0.519	0.506	0.538	0.602	0.520
Posto Trabalho Manual 3		0.622	0.570	0.876	0.921	0.764	0.544	0.851	0.879	0.612
Posto Trabalho Manual 4		1.792	1.788	0.946	0.949	0.875	0.749	0.903	0.884	1.810
Posto Trabalho Manual 5		1.792	0.504	0.854	0.924	0.701	0.526	0.923	0.857	0.605
Posto Trabalho Manual 6		0.628	0.575	0.917	0.953	0.678	0.676	0.800	0.873	0.603
Célula de Montagem		0.516	0.525	0.565	0.574	0.519	0.516	0.579	0.519	0.498
Controle de Qualidade		1.387	1.415	1.460	1.138	1.457	1.427	1.451	1.195	0.704
Paletes numa fila de saída	Média	1.011	0.920	0.971	0.966	0.896	0.849	0.967	0.942	0.865
	Desv. Padrão	0.917	0.905	0.849	0.837	0.862	0.877	0.853	0.842	0.904
Média do número total de paletes nas filas saída		20.705	19.306	20.076	20.093	19.030	18.338	20.135	19.804	18.491

A Tabela 10.29 apresenta os valores do WIP médio total da fábrica (somatório do WIP médio de todas as células de fabrico) e o número de paletes produzidas de produto acabado, para todas as simulações efectuadas. Num dado momento, o número de paletes produzidas de produto acabado é igual à soma das paletes na fila de saída da CM, mais o número de paletes existentes no CQ, no AAPA e no CEPA, mais o número de paletes expedidas pelo CEPA.

O valor do WIP médio total da fábrica não apresenta variações significativas, exceptuando as duas primeiras simulações, que são as piores deste ponto de vista.

Tabela 10.29 – WIP médio e total de paletes de produto acabado produzidas.

Simulação N.º	1	2	3	4	5	6	7	8	9
RADV	TMCV	TMCV	MTFS	MELFS	PCPSM	TMCV	MTFS	MELFS	PCPSM
RADCT	VMP					VMU			
REAFD	Sim	Não							
WIP médio total de todas as células de fabrico	26.726	27.188	23.025	22.235	22.967	23.276	23.234	23.283	23.874
Número de paletes de prod. acabado produzidas	107	106	91	87	101	103	96	103	106

O número de paletes produzidas de produto acabado em cada simulação está representado no gráfico da Figura 10.11. Atendendo aos indicadores de desempenho apresentados anteriormente, nomeadamente o número de tarefas de movimentação e os tempos de execução de uma tarefa, a tendência verificada na quantidade produzida de produto

acabado não é surpreendente, à excepção da simulação 9, que acaba por permitir a obtenção de um resultado ao nível das estratégias de escalonamento baseadas na regra TMCV, que obtiveram indicadores de desempenho acentuadamente melhores no número de tarefas de movimentação efectuadas e no tempo médio de execução do passo 1 das tarefas.

Este resultado é aparentemente estranho, uma vez que a regra de despacho TMCV permite intrinsecamente obter um desempenho do SAGVs mais optimizado. A razão que pode explicar o bom resultado da estratégia (PCPSM;VMU;Não) é a escolha de caminhos tecnológicos mais favoráveis nessa simulação do que nas outras simulações (por razões de ordem probabilística), tornando possível a obtenção da mesma quantidade de produto acabado com menos tarefas de movimentação executadas pelo SAGVs (ver Tabela 10.10).

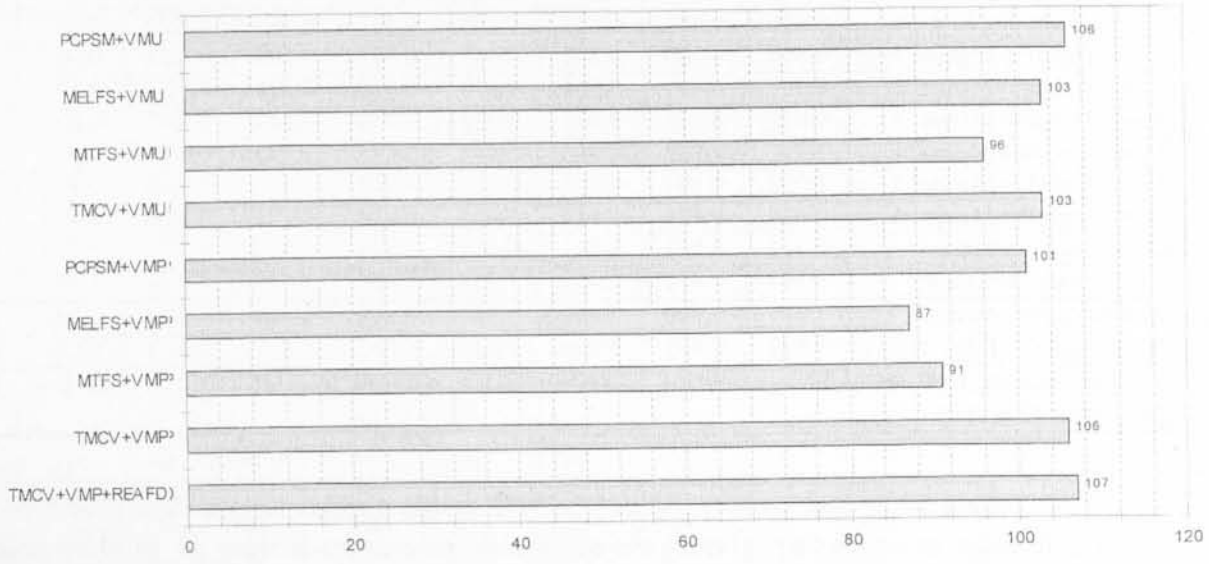


Figura 10.11 – Gráfico do número de paletes produzidas de produto acabado.

11. Conclusões

Esta dissertação procurou analisar os problemas fundamentais que se levantam na concepção de SAGVs.

O desempenho dos SAGVs terá de ser suficientemente bom para dar resposta aos requisitos colocados pelos actuais sistemas de fabrico flexíveis, onde é cada vez mais gritante a necessidade de garantir flexibilidade e capacidade de resposta a um mercado cada vez mais competitivo e em constante mutação. Neste contexto, o primeiro capítulo fez o enquadramento da aplicação dos sistemas de transporte e de manuseamento automático nos actuais sistemas de fabrico, e identificou os requisitos de desempenho que os SAGVs deverão cumprir, para se conceberem sistemas de fabrico com um bom desempenho global.

No segundo capítulo, apresentou-se a tecnologia envolvida na construção de um AGV e algumas técnicas actualmente existentes para a implementação do seu sistema de navegação. Os AGVs actualmente utilizados continuam a ser na sua maioria do tipo filoguiado, apesar de já serem conhecidas outras técnicas de navegação mais sofisticadas, que permitem obter sistemas de transporte automáticos mais flexíveis e de mais fácil reconfiguração. Os AGVs filoguiados continuam a liderar o mercado de fabrico de AGVs, devido à sua maior robustez, maior fiabilidade e melhor domínio do seu princípio de funcionamento. No entanto, a tendência actual permite antever a progressiva substituição, ao longo dos próximos anos, dos AGVs filoguiados por AGVs baseados em sistemas *laser*, fruto da experiência que se tem vindo a adquirir na implementação de sistemas baseados nessa tecnologia inovadora.

Também no segundo capítulo, definiu-se uma arquitectura de controlo para um SAGVs, constituída por três camadas hierárquicas. O capítulo terminou com a apresentação das razões que tornam inviável a avaliação do desempenho de um SAGVs de dimensão real pela via analítica, e com a justificação da necessidade de se recorrer a técnicas de análise baseadas na simulação de modelos dos sistemas. Foram ainda apresentados alguns índices de desempenho de um SAGVs, e foi discutido o problema da não existência de uma métrica universalmente aceite, para a avaliação e comparação do desempenho de sistemas de fabrico e, em particular, de SAGVs.

No terceiro capítulo fez-se uma análise sistemática dos problemas que se levantam na concepção de um SAGVs: concepção do *layout*, determinação do número de AGVs, gestão de tráfego e *deadlocks*, encaminhamento e escalonamento.

Apresentaram-se aspectos a ter em conta na concepção de um *layout* eficiente e tolerante a falhas de AGVs, nomeadamente o planeamento racional da localização das trajectórias e a criação de caminhos alternativos.

Compararam-se as vantagens e desvantagens de *layouts* unidireccionais e bidireccionais: o *layout* unidireccional facilita o controlo do SAGVs, mas implica a utilização de mais veículos do que um *layout* bidireccional, para servir os requisitos de um mesmo sistema de fabrico; o *layout* bidireccional permite melhorar a flexibilidade e a eficiência do SAGVs, desde que suportado em técnicas de controlo eficazes e mais complexas do que as

técnicas utilizadas no *layout* unidireccional, e permite obter uma configuração mais compacta dos sistemas flexíveis de fabrico, conseguindo servir os pontos do *layout* com menos pistas.

Apresentou-se um modelo de representação de um *layout* de AGVs, baseado num grafo constituído por nós e ramos (arcos). Comparou-se a utilização de matrizes e de listas como possíveis representações computacionais de um *layout*, tendo-se elegido as listas como o melhor processo, devido aos requisitos de armazenamento de informação imprevisíveis e à necessidade de manipulação intensiva da informação associada ao *layout*.

Para a determinação do número mínimo de AGVs necessários, apresentou-se um modelo matemático de uma fábrica servida por um SAGVs, bem como um conjunto de expressões que permitem chegar a uma estimativa do número mínimo de veículos.

No que diz respeito à gestão de tráfego:

- apresentou-se um modelo do movimento de um AGV, as regras de controlo associadas (política de reserva e de libertação de nós) e uma representação do *layout*, que incluísse informação necessária à realização da gestão de tráfego; referiu-se a utilidade da utilização de nós fictícios na gestão de cruzamentos e de áreas partilhadas por mais do que uma trajectória;
- abordou-se o problema dos *deadlocks*, definindo o conceito, ilustrando a sua ocorrência a diferentes níveis do controlo de um sistema de fabrico, referindo condições necessárias para a sua ocorrência, e comparando a fazibilidade e as vantagens e desvantagens de várias estratégias para o tratamento do problema (prevenção, evitação e detecção e resolução).

Formulou-se o problema do encaminhamento de AGVs, apresentando-se os requisitos a satisfazer por um bom algoritmo de encaminhamento. Classificaram-se as abordagens possíveis ao problema em planeamento completo do caminho e planeamento incremental do caminho. Referiu-se a existência de uma estreita relação entre algoritmos de encaminhamento e de escalonamento, classificando esses algoritmos em não adaptativos (estáticos) e em adaptativos. Apresentaram-se vantagens e desvantagens dos dois tipos de abordagem, tendo-se concluído que a escolha do algoritmo de encaminhamento tem de resultar de uma solução de compromisso entre complexidade do algoritmo e a garantia dos requisitos de tempo real do controlo do SAGVs. Classificaram-se os algoritmos adaptativos em centralizados, locais e distribuídos. Referiram-se técnicas inovadoras de encaminhamento baseadas em janelas temporais, e que evitam a ocorrência de conflitos entre AGVs. Apresentou-se o algoritmo de Dijkstra para a determinação do caminho mais curto.

Definiu-se o problema de escalonamento de AGVs, dividindo o problema em afectações desencadeadas por um veículo e afectações desencadeadas por um centro de trabalho. Descreveram-se regras de despacho para ambas os tipos de afectações. Efectuou-se uma formulação matemática do problema de afectação de AGVs a tarefas, baseada em programação linear, e descreveu-se o algoritmo húngaro, que permite a sua resolução eficiente. Fez-se ainda referência ao mecanismo de reafectação dinâmica, como sendo uma optimização adicional ao escalonamento baseado na determinação da solução óptima do problema de afectação.

O capítulo quatro apresentou os conceitos básicos das Redes de Petri (RP) autónomas, como sendo uma ferramenta poderosa e rigorosa para a modelização de sistemas dinâmicos de eventos discretos. Os assuntos tratados foram: elementos constituintes, marcação, mecanismo de disparo de uma transição, definição de RP autónoma e não autónoma, estruturas particulares de RP, propriedades de uma RP, e introdução a técnicas de análise

formais, baseadas em grafos de marcações, álgebra linear e métodos de redução. Definiram-se ainda comportamentos dos sistemas de eventos discretos facilmente identificáveis através de uma RP.

O capítulo cinco estendeu a aplicabilidade das RP, através da sincronização do disparo de transições com a ocorrência de eventos externos, da introdução do factor tempo de forma fixa ou estocástica, e da definição de modelos de alto nível. As RP de alto nível possibilitam que as marcas (*tokens*) possam ter informação associada, bem como a definição de tipos de dados abstractos para os valores guardados nas marcas. As extensões não autónomas apresentadas foram as RP: sincronizadas, temporizadas, estocásticas e coloridas. Foi feita ainda uma breve introdução às RP orientadas a objectos, que são actualmente objecto de muitos trabalhos de investigação na área das ciências da computação.

O capítulo seis descreveu ferramentas de *software* actualmente disponíveis no mercado para suportar a modelização baseada em RP, terminando com a apresentação do *software* Artifex, baseado em RP estocásticas e coloridas, que foi utilizado na parte experimental do trabalho descrito nesta dissertação.

O capítulo sete especificou um controlador de um SAGVs, baseado na arquitectura de três camadas definida no capítulo dois.

Os capítulos oito e nove relataram a implementação, no ambiente de modelização Artifex, de um modelo do controlador do SAGVs especificado no capítulo sete, bem como da sua integração num sistema de fabrico idealizado para simular o controlador do SAGVs modelizado.

O capítulo dez descreveu a realização de simulações do modelo construído no ambiente de simulação Artifex. As simulações efectuadas consistiram na simulação da mesma configuração do sistema de fabrico, utilizando nove estratégias de escalonamento diferentes: duas baseadas no algoritmo húngaro e as outras sete baseadas na combinação de regras de despacho definidas no capítulo três. Foram apresentados e discutidos os principais resultados obtidos nas simulações, com o objectivo de comparar o desempenho obtido com cada estratégia de escalonamento utilizada.

As principais conclusões obtidas foram:

- as estratégias de escalonamento baseadas na utilização da regra de despacho 'Tempo Mais Curto de Viagem' (TMCV), incluindo o algoritmo húngaro, apresentam um desempenho claramente superior a todas as outras, no que diz respeito ao número de tarefas de movimentação executadas no mesmo intervalo de tempo e ao tempo de execução do passo 1 das tarefas de movimentação (tempo do movimento entre a posição inicial do AGV e o ponto de carga da carga a transportar);
- como na grande maioria das decisões de escalonamento há menos AGVs disponíveis para a realização de tarefas do que tarefas por executar, a regra de despacho para a afectação desencadeada por um veículo é utilizada muitas mais vezes do que a regra de despacho para a afectação desencadeada por um centro de trabalho; por esta razão o desempenho do SAGVs é muito dependente da escolha da primeira regra e pouco dependente da escolha da segunda regra;

- a utilização do algoritmo húngaro, em conjugação com o mecanismo de reafectação dinâmica, pode não conduzir a melhorias significativas do desempenho do SAGVs, porque o tipo de situações que permite optimizar podem ocorrer muito pontualmente, e porque a optimização resultante da reafectação é, em muitos casos, pouco significativa;
- a eficiência da estratégia de escalonamento utilizada reflecte-se numa melhoria do desempenho da gestão de tráfego, porque diminui a propensão do sistema para a ocorrência de conflitos entre AGVs, devido à optimização do fluxo de AGVs;
- a regra de despacho 'Primeiro a Chegar, Primeiro a ser Servir Modificada' permite obter um bom desempenho do ponto de vista do tempo de atendimento das tarefas, ou seja, uma boa minimização do tempo de permanência em fila de espera das tarefas; mesmo assim, é superada pelo desempenho da regra 'Tempo Mais Curto de Viagem';
- as diferenças de desempenho verificadas entre as várias estratégias de escalonamento simuladas são fundamentalmente influenciadas pela optimização do tempo de atendimento das tarefas, ou seja, pela optimização do tempo de permanência das tarefas em fila de espera; a optimização do tempo de viagem desde a posição inicial do AGV até ao ponto de carga é menos relevante na obtenção de um bom desempenho global do SAGVs;
- as melhores estratégias de escalonamento (em número de tarefas de movimentação efectuadas) apresentam um baixo coeficiente de arranques de um AGV por trajecto executado;
- a utilização de regras de despacho baseadas no estado de ocupação dos *buffers* de saída das células de fabrico ('Máximo Tamanho da Fila de Saída' e 'Mínimo Espaço Livre na Fila de Saída') permite obter um maior nivelamento na ocupação desses mesmos *buffers*;
- um bom desempenho do escalonador reflecte-se num maior número de tarefas de movimentação do sistema de transporte por unidade de tempo (optimização do fluxo de materiais no interior da fábrica) e, consequentemente, num aumento da produtividade do sistema de fabrico.

11.1 Desenvolvimentos futuros

O desenvolvimento de um Sistema de Transporte Automático baseado em AGVs é uma tarefa multi-disciplinar, pois envolve várias áreas do conhecimento: Electrónica (tecnologia do AGV), Automação Industrial (automatização, informática industrial e integração de dispositivos de automação), Sistemas de Manufactura (planeamento, gestão, análise de desempenho, integração, CAD/CAM para a concepção de *layouts*, etc), Ciências da Computação (modelização com Redes de Petri, escalonamento, *deadlocks*, encaminhamento, etc) e Investigação Operacional (optimização, escalonamento e encaminhamento).

Este facto confere ao problema um âmbito muito alargado, que se reflecte necessariamente numa grande dispersão das fontes de informação existentes sobre assuntos relevantes na análise e concepção de SAGVs. Por esta razão, esta dissertação resulta de um compromisso entre conseguir abranger todos os assuntos relevantes, e conseguir tratar cada um deles com um nível de detalhe suficiente à percepção do essencial das questões, que se

levantam no desenvolvimento de SAGVs. Como é óbvio, muito mais haveria para dizer sobre problemas como o escalonamento, o encaminhamento ou os *deadlocks*, mas o âmbito alargado do trabalho torna impossível um maior nível de detalhe.

O trabalho procurou analisar o estado da arte das técnicas disponíveis para o desenvolvimento de AGVs, e identificar questões cuja resolução continua em aberto.

De seguida, são apontados alguns aspectos que constituem linhas de orientação para futuros trabalhos de investigação científica nesta área.

- O desenvolvimento de sistemas de navegação de AGVs mais flexíveis (e.g sistema *laser*) terá de ser acompanhado pelo desenvolvimento de técnicas de encaminhamento e de escalonamento mais complexas e elaboradas, do que a simples aplicação de heurísticas ou de regras de despacho, de forma a permitir tirar partido da flexibilidade oferecida pelas novas soluções tecnológicas.
- A análise e concepção de sistemas de fabrico, e de SAGVs em particular, baseia-se quase sempre na modelização e simulação dos sistemas. Por isso, é importante que se proceda ao desenvolvimento de técnicas de modelização mais poderosas, nomeadamente à custa da evolução das Redes de Petri no sentido da orientação a objectos, e de um maior nível de abstracção e de reutilização dos modelos. A reutilização de modelos e a criação de librarias de modelos básicos são aspectos importantes, para facilitar a rápida modelização e prototipagem de novos sistemas desenvolvidos.
- O desenvolvimento dos paradigmas de modelização de sistemas de eventos discretos (e.g. Redes de Petri) terá de ser acompanhado pelo desenvolvimento de ferramentas de *software* de suporte à actividade de modelização e simulação, de forma a facilitar a aplicabilidade das novas técnicas de análise desenvolvidas à resolução de problemas práticos de dimensão real.
- A ausência de métricas universalmente aceites para a avaliação do desempenho dos sistemas de fabrico impede a sistematização e a formalização matemática do processo de análise e de concepção desses sistemas. Será importante que a investigação científica na área dos sistemas de manufactura dê particular atenção ao desenvolvimento de métricas, que permitam a avaliação e a comparação não ambígua de várias soluções possíveis para um mesmo problema de engenharia de sistemas de fabrico.

Lista de Referências

[1] "Automation, Production Systems, and Computer Integrated Manufacturing", M.P. Groover; Prentice-Hall

[2] "Comparision of Dynamic Routeing Techniques for Automated Guided Vehicle System", F. Taghaboni-Dutta , J.M.A. Tanchoco; International Journal of Production Research, 1995, Vol. 33, N.º 10, pp. 2653-2669

[3] "Computer Integrated Manufacturing and Engineering", U. Reembold, B.O. Nuagi, A. Store; Addison and Wesley Publishing Company, 1995

[4] "Design and Evaluation of Automated Guided Vehicle Systems for Flexible Manufacturing Systems: an extended timed Petri Net-based approach", K.R. Raju, O.V.K. Chetty; International Journal of Production Research, 1993, Vol. 31, N.º 5, pp. 1069-1096

[5] "Computer Control of Flexible Manufacturing Systems – Research and Development", S.B. Joshi, J.S. Smith; Chapman&Hall, 1994

[6] "Developing Conflict-Free Routes for Automated Guided Vehicles", N. Krishnamurthy, R. Batta, M.H. Karwan; Operations Research, Nov.-Dez. 1993, Vol. 41, N.º 6

[7] "Potentials for Bi-Directional Guidepath for Automated Guided Vehicle Based Systems", P.J.Eglebu, J.M.A. Tanchoco; International Journal of Production Research 24, 1986, pp. 1075-1097

[8] "Operations Research – an Introduction – 5th Edition", H.A. Taha; Prentice Hall International Editions, 1992

[9] "An Introduction to Data Structures with Applications – 2nd Edition", J.P.Tremblay, P.G. Sorenson; McGraw-Hill, 1984

[10] "Design of an Automated Guided Vehicle-Based Material Handling System for a Flexible Manufacturing System", B. Mahadevan, T.T. Narendran; International Journal of Production Research, 1990, Vol. 28, N.º 9, pp. 1611-1622

[11] "Traffic Management and Deadlock Detection and Recovery in an Automated Guided Vehicles Management System", F. A. Lobo, J. R. Lopes, J. M. Machado, F. J. Restivo; Centro de CIM do Porto, 1997

[12] "Deadlock Avoidance in Flexible Manufacturing Systems with Concurrently Competing Process Flows", Z.A. Banaszak, B.H. Krogh; IEEE Transactions on Robotics and Automation, Vol. 6, N.º 6, Dez. 1990

[13] "Detection of Deadlocks in Flexible Manufacturing Cells", R.A. Wysk, N.S. Yang, S. Joshi; IEEE Transactions on Automation and Robotics, Vol. 7, N.º 6, Dez. 1991

[14] "Deadlock Prevention in Manufacturing Systems With AGV Systems: Banker's Algorithm Approach", C.W. Kim, J.M.A. Tanchoco, P. Koo; Journal of Manufacturing Science and Engineering, Vol. 119, pp. 849-854, Nov. 1997

[15] "Prevention of System Deadlocks", A.N. Habermann; Communications of the ACM, 1969, Vol. 12, N.º 7, pp. 373-377

[16] "Characterization of Automatic Guided Vehicle Dispatching Rules", P.J. Egbelu, J.M.A. Tanchoco; International Journal of Production Research, 1984, Vol. 22, N.º 3, pp. 359-374

[17] "Complexity of the AGV Shortest Path and Single-Loop Guide Path Layout Problems", M.C. Guzman, N. Prabhu, J.M.A. Tanchoco; School of Industrial Engineering, Universidade de Purdue, EUA, 1994

[18] "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows", M. Desrochers, J. Desrosiers, M. Solomon; Operations Research, Mar.-Abr. 1992, Vol. 40, N.º 2

[19] "Conflict-Free Shortest-Time Bidirectional AGV Routeing", C.W. Kim, J.M.A. Tanchoco; International Journal of Production Research, 1991, Vol. 29, N.º 12, pp. 2377-2391

[20] "A New Shortest Paths Ranking Algorithm", E.Q.V. Martins, J.L.E. Santos; Departamento de Matemática, Universisade de Coimbra, Out. 1996

[21] "Routing and Dispatching in an Automated Guided Vehicles Management System", F. A. Lobo, J. R. Lopes, J. M. Machado, F. J. Restivo; Centro de CIM do Porto, 1997

[22] "Advanced Engineering Mathematics - 6th Edition", E. Kreyszig; John Wiley and Sons, New York, 1988

[23] "Path Planning under Time-Dependent Uncertainty", M.P. Wellman, M. Ford, K. Laarson; Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence (UAI-95), Agosto 1995, pp. 18-20

[24] "A Time Window Approach to Simultaneous Scheduling of Machines and Material Handling System in an FMS", U. Bilge, G. Ulusoy; Operations Research, Dez. 19995, Vol. 43, N.º 6

[25] "A Survey of Scheduling Rules", S.S. Panwalkar, W. Iskander; Operations Research, Jan.-Fev. 1977, Vol. 25, N.º 1

[26] "Petri Nets and Grafcet – Tools for modeling discrete event systems", R. David, H. Alla; Prentice-Hall, 1992

[27] "A Brief Introduction to Coloured Petri Nets", K. Jensen; Computer Science Department, University of Aarhus, Denmark, 1997

[28] "A Methodology for the Construction of Simulation Models Using Object-Oriented Petri Nets", C.D. Keen, C.A. Lakos; Department of Computer Science, University off Tasmania, Australia, 1993

[29] "From Coloured Petri Nets to Object Petri Nets", C. Lakos; Computer Science Department, University of Tasmania, Australia, 1995

[30] "Functional and Object Oriented Structuring of Petri Nets", V. Janousek; Department of Computer Science and Engineering, FEI VUT, Czech Republic, 1997

[31] "PNtalk – a Computerized Tool for Object Oriented Petri Nets Modelling", M. Ceska, V. Janousek, T. Vojnar; Department of Computer Science and Engineering, FEI VUT, Czech Republic, 1997

[32] "A Formal Model for Object Oriented Petri Nets Modeling", M. Ceska, V. Janousek; Department of Computer Science and Engineering, FEI VUT, Czech Republic, Jan. 1997

[33] "SYROCO: a C++ Implementation of Cooperative Objects", C. Sibertin-Blanc; IRIT, Université Toulouse 1, France, Jun. 1995

[34] "Efficient Simulation of THOR Nets", S. Schof, M. Sonnenschein, R. Wieting; Fachbereich Informatik, Universitat Oldenburg, Germany, 1995

[35] "Artifex 4.1 – Tutorial", Artis s.r.l., Jun. 1997

Anexos

Anexo A – Código C incluído nas classes do modelo do SAGVs

A.1 Ficheiros de cabeçalho (header files)

A.1.1 ulint.h

```
/* ulint.h */

#ifndef MAXINT
#define MAXINT     4294967295 /* maior numero inteiro sem sinal */
typedef unsigned long ulint; /* inteiro sem sinal */
#endif
```

A.1.2 lefich.h

```
/* lefich.h */

/* prototipos de funcoes utilizadas na leitura de ficheiros */

void procura_caracter(/* FILE *f, int c, ulint *linha */);
int procura_prox_campo(/* FILE *f, ulint *linha */);
ulint le_ulint(/* FILE *f, ulint i */);
int procura_prox_linha(/* FILE *f, int c, ulint *linha */);
void le_double(/* FILE *f, double *d */);
```

A.1.3 listas.h

```
/* listas.h */

/* tipos e funcoes relacionados com a manipulacao
de listas ligadas */


#include <stdlib.h>        /* malloc() e free() */
#include <malloc.h>        /* idem */
#include "ulint.h"


/* *** listas ligadas de inteiros sem sinal *** */

/* no de uma lista ligada */
typedef struct s_no_l_ulint{
     ulint i;
     struct s_no_l_ulint *anterior, *seguinte;
     } no_l_ulint;

typedef no_l_ulint* ap_no_l_ulint;
```

```
/* prototipos de funcoes */
ap_no_l_ulint get_ultimo_ulint(/* ap_no_l_ulint primeiro */);
ap_no_l_ulint procura_ulint(/* ulint i, ap_no_l_ulint primeiro */);
int insere_ulint(/* ulint i, ap_no_l_ulint *primeiro,
    ap_no_l_ulint *ultimo, char repetidos */);
int remove_ulint(/* ulint i, ap_no_l_ulint *primeiro, ap_no_l_ulint
*ultimo */);
void limpa_l_ulint(/* ap_no_l_ulint primeiro */);
ap_no_l_ulint append_l_ulint(/* ap_no_l_ulint *p_l1,
    ap_no_l_ulint p_l2, char repetidos */);
void prn_l_ulint(/* ap_no_l_ulint primeiro */);
void max_l_ulint(/* ap_no_l_ulint primeiro, ulint *mx, ulint *n_mx */);
void min_l_ulint(/* ap_no_l_ulint p, ulint lim_min, ulint *m, ulint *im
*/);
void atribui_ulint_pos(/* ulint i, ulint pos, ap_no_l_ulint p */);
ulint elem_l_ulint(/* ulint pos, ap_no_l_ulint p */);
int inters_igual_vazio(/* ap_no_l_ulint l1, ap_no_l_ulint l2 */);
int copia_l_ulint(/* ap_no_l_ulint l1, ap_no_l_ulint *p_l2,
ap_no_l_ulint *u_l2 */);


/* *** listas ligadas de estruturas com 2 inteiros sem sinal *** */

/* no de uma lista ligada */
typedef struct s_no_l_2uint{
    ulint i1, i2;
    struct s_no_l_2uint *seguinte;
    } no_l_2uint;

typedef no_l_2uint* ap_no_l_2uint;

/* prototipos de funcoes */
ap_no_l_2uint procura_2uint(/* ulint i1, ulint i2, ap_no_l_2uint
primeiro */);
int insere_2uint(/* ulint i1, ulint i2, ap_no_l_2uint *primeiro,
    ap_no_l_2uint *ultimo */);
ap_no_l_2uint limpa_l_2uint(/* ap_no_l_2uint h */);
void prn_l_2uint(/* ap_no_l_2uint h */);
int copia_l_2uint(/* ap_no_l_2uint l1, ap_no_l_2uint *l2 */);
void remove_cab_l_2uint(/* ap_no_l_2uint *p */);
void remove_2uint(/* ap_no_l_2uint e, ap_no_2uint *p */);


/* *** listas ligadas de estruturas com um inteiro sem sinal e 4 doubles
*** */

/* no de uma lista ligada */
typedef struct s_no_l_ui4d{
    uiint i;
    double d1, d2, d3, d4;
    struct s_no_l_ui4d *seguinte;
    } no_l_ui4d;

typedef no_l_ui4d* ap_no_l_ui4d;

/* prototipos de funcoes */
```

```
ap_no_l_ui4d get_ui4d_ui(/* ap_no_l_ui4d h, ulint i */);
ap_no_l_ui4d insere_ui4d(/* ap_no_l_ui4d h, ap_no_l_ui4d ultimo,
    ulint i, double d1, double d2, double d3, double d4, char repetidos
*/);
ap_no_l_ui4d limpa_l_ui4d(/* ap_no_l_ui4d h */);
void prn_l_ui4d(/* ap_no_l_ui4d h */);


/* *** listas ligadas de estruturas com um inteiro sem sinal e um double
*** */

/* no de uma lista ligada */
typedef struct s_no_l_uid{
    ulint i;
    double d;
    struct s_no_l_uid *seguinte;
    } no_l_uid;

typedef no_l_uid* ap_no_l_uid;

/* prototipos de funcoes */
ap_no_l_uid get_uid_ui(/* ap_no_l_uid h, ulint i */);
ap_no_l_uid insere_uid(/* ap_no_l_uid h, ap_no_l_uid ultimo, ulint i,
    double d, char repetidos */);
ap_no_l_uid limpa_l_uid(/* ap_no_l_uid h */);
void prn_l_uid(/* ap_no_l_uid h */);
ap_no_l_uid obtem_uid_min_d(/* ap_no_l_uid h */);
int remove_uid(/* ulint i, ap_no_l_uid *h, ap_no_l_uid *u */);


/* *** listas ligadas de estruturas com um inteiro sem sinal e 3 doubles
*** */

/* no de uma lista ligada */
typedef struct s_no_l_ui3d{
    ulint i;
    double d1, d2, d3;
    struct s_no_l_ui3d *seguinte;
    } no_l_ui3d;

typedef no_l_ui3d* ap_no_l_ui3d;

/* prototipos de funcoes */
ap_no_l_ui3d get_ui3d_ui(/* ap_no_l_ui3d h, ulint i */);
ap_no_l_ui3d insere_ui3d(/* ap_no_l_ui3d h, ap_no_l_ui3d ultimo,
    ulint i, double d1, double d2, double d3, char repetidos */);
ap_no_l_ui3d limpa_l_ui3d(/* ap_no_l_ui3d h */);
void prn_l_ui3d(/* ap_no_l_ui3d h */);


/* *** listas ligadas de estruturas com 1 inteiro com sinal *** */

/* no de uma lista ligada */
typedef struct s_no_l_int{
    int i;
    struct s_no_l_int *seguinte;
```

```
    } no_l_int;

typedef no_l_int* ap_no_l_int;

/* prototipos de funcoes */
int insere_int(/* int i, ap_no_l_int *primeiro, ap_no_l_int *ultimo */);
int obtem_int(/* int indice, ap_no_l_int primeiro */);
void limpa_l_int(/* ap_no_l_int *h */);
```

A.1.4 rand_fun.h

```
/* rand_fun.h */

/* prototipos de funcoes para geracao de numeros aleatorios */

double random(void);
double rnd_exp_neg(/* double vm */);
double rnd_normal(/* double vm, double dp */);
int rnd_binomial(/* double p */);
```

A.1.5 layout.h

```
/* layout.h */

/* definicao de constantes e de tipos relacionados com a estrutura
utilizada para
representar um layout de AGV's */



/* no de uma lista ligada de ramos de um no de um layout de AGV's */
typedef struct s_no_l_ramos{
 ulint destino;            /* no destino do ramo */
 int orient_final; /* orientacao final do AGV em relacao ao sent. + */
 ulint custo;              /* custo associado ao movimento atraves do
ramo (10^^-1s)*/
 ulint pen_aux;            /* penalizacao auxiliar (s) */
 ulint tipo_ramo;  /* tipo do ramo */
 ap_no_l_ulint bloqueios; /* lista de bloqueios */
 struct s_no_l_ramos *seguinte;
} no_l_ramos;

typedef no_l_ramos* ap_no_l_ramos;


/* no de uma lista ligada de nos de um layout de AGV's */
typedef struct s_no_l_nos{
 ulint id_no, tipo_no;
 int pos_rel_mesa; /* posicao relativa da mesa em relacao ao sent. + */
                   /* quando nao e relevante vale 0 */
 ap_no_l_ulint ocupacao_nos;
 ap_no_l_ramos ramos;
 struct s_no_l_nos *seguinte;
} no_l_nos;

typedef no_l_nos* ap_no_l_nos;
```

```
/* tipos de ramos */
#define TERMINAL   0
#define PASSAGEM   1
#define TERM_PASSAG        2


/* tipos de nos */
#define TRANSITO   0
#define CARGA_DESCARGA     1
#define CARGA              2
#define DESCARGA   3


/* prototipos de funcoes para leitura e validacao do layout */
ap_no_l_ramos limpa_l_ramos(/* ap_no_l_ramos h*/);
ap_no_l_nos limpa_layout(/* ap_no_l_nos h */);
ap_no_l_ulint le_ocupacao_nos(/* FILE *f, ulint *linha, ulint id_no */);
ap_no_l_ulint le_bloqueios(/* FILE *f, ulint *linha,
    ulint origem, ulint destino, int *or_final */);
ulint arredonda(/* float f */);
ulint fcusto(/* ulint dre, ulint dcr, ulint dma, ulint p_aux,
    ulint vre, ulint vcr, ulint vma */);
ap_no_l_ramos le_ramo(/* FILE *f, ulint *linha, ulint id_no,
    ulint vre, ulint vcr, ulint vma */);
int le_ramos(/* FILE *f, ulint *linha, ulint id_no, ap_no_l_ramos
*ramos,
    ulint vre, ulint vcr, ulint vma */);
int le_no(/* FILE *f, ulint *linha, ap_no_l_nos *novo,
    ulint vre, ulint vcr, ulint vma */);
ap_no_l_nos le_layout(/* char *nome_ficheiro,
    ulint vre, ulint vcr, ulint vma */);
ap_no_l_nos get_no(/* ap_no_l_nos layout, ulint id */);
ap_no_l_ramos get_ramo(/* ap_no_l_nos no_origem, ulint id_no_destino
*/);
int unicidade_id_nos(/* ap_no_l_nos layout */);
int ocup_nos_e_ramos_validos(/* ap_no_l_nos layout */);
void completa_bloqueios(/* ap_no_l_nos layout */);
void prn_ramo(/* ap_no_l_ramos r */);
void prn_no(/* ap_no_l_nos no */);
void prn_layout(/* ap_no_l_nos layout */);
void le_f_pos_rel_pcd(/* char *nome_ficheiro, ap_no_l_nos layout */);
int pos_mesas_conhecidas(/* ap_no_l_nos layout */);
```

A.1.6 mens_log.h

```
/* mens_log.h */

/* prototipos de funcoes que permitem fazer o display de mensagens no
stdout
e, opcionalmente, num ficheiro de log
*/

void preenche_str_data_hora(/* char *s */);
void abre_fich_log(/* char *nome, char *modo */);
void prn_string(/* char *s */);
void fecha_fich_log(/* void */);
```

A.1.7 sgm_sga.h

```
/* sgm_sgga.h */

/* Tipo utilizado para passar MENSAGENS SGM <-> SGAGV */

typedef struct s_m_SGM_SGAGV{
    ulinit id_mens;
    ulinit tipo_mens;
    ulinit p1, p2, p3, p4;
    } mensag_SGM_SGAGV;


/* funcaio utilizada para imprimir no stdout o conteudo de uma mensagem
SGM<->SGAGV */
void prni_m_SGM_SGAGV(/* double t, mensag_SGM_SGAGV mens */);
```

A.1.8 sga_spa.h

```
/* sga_spa.h */

/* Tipo utilizado para passar MENSAGENS SGAGV <-> SPAGV */

typedef struct s_m_SGAGV_SPAGV{
    ulimt id_agv;
    ulimt tipo_mens;
    ulimt p1, p2;
    } mensag_SGAGV_SPAGV;


/* funcaao utilizada para imprimir no stdout o conteudo de uma mensagem
SGAGV<->SPAGV */
void prrn_m_SGAGV_SPAGV(/* double t, mensag_SGAGV_SPAGV mens */);
```

A.1.9 sistaagvs.h

```
/* sistaagvs.h */


/* CONSSTANTES do SISTEMA */


/* Estaados de Movimentacao de um AGV */
#definee DISABLED   0
#definee ENABLED          1
#definee PARADO           2
#definee EX_MOVIM   3
#definee CARGA_AGV  4
#definee DESCARGA_AGV     5
#definee BLOQUEADO  6
#definee ERRO             7

/* Estaados de Alocacao de um AGV */
#definee NAO_ALOCADO      0
#definee ALOCADO          1
```

```
/* Estados de Carga de um AGV */
#define DESCARREGADO      0
#define CARREGADO  1

/* Tipos de Tarefas */
#define NORMAL            0
#define NORMAL_AGV 1
#define POSICION   2
#define T_CARGA           3
#define CARGA_AGV  4
#define T_DESCARGA 5
#define POSIC_ESPEC       6

/* Estados de uma Tarefa */
#define FILA              0
#define PASSO1            1
#define PASSO2            2
#define PASSO3            3
#define PASSO4            4
#define CONCLUIDA  5
#define REJEITADA  6



/*
MENSAGENS SGM -> SGAGV

tipo_mens          p1            p2            p3            p4
ENABLE_AGV <id_agv>       ALOCA_AGV      -              -
ENABLE_AGV <id_agv>       DESALOCA_AGV   -              -
DISABLE_AGV<id_agv>       -              -              -
TAREFA_NORMAL      QUALQUER       <origem>       <destino>      <prior>
TAREFA_NORMAL      <id_agv>       <origem>       <destino>      -
TAREFA_POSICION <id_agv>  <no>           -              -
TAREFA_CARGA       QUALQUER       <no>           -              <prior>
TAREFA_CARGA       <id_agv>       <no>           -              -
TAREFA_DESCARGA    <id_agv>       <no>           -              -
CANCELA_TAREFA     UMA_TAREFA     <id_tarefa>    -              -
CANCELA_TAREFA     TODAS_TAR_EXEC -              -              -
CANCELA_TAREFA     TODAS_TAR      -              -              -


MENSAGENS SGAGV -> SGM

tipo_mens   p1             p2             p3             p4
CONFIRMACAO SUCESSO        -              -              -
CONFIRMACAO INSUCESSO      <erro>         -              -
TAREFA_CANCEL      <estado_tar>   <id_agv>       <razao>        -
ESTADO_AGV <id_agv>       <estado_movim><estado_aloc> <estado_carga>
ESTADO_TAREFA      <id_tarefa>    <estado_tar>   <id_agv>       -
T_POSIC_ESPEC      <id_tarefa>    <id_agv>       <no>           -
*/


/* CONSTANTES ASSOCIADAS A MENSAGENS ENTRE O SGM E O SGAGV */
```

```
/* tipos de mensagens SGM -> SGAGV */

#define ENABLE_AGV          0
#define DISABLE_AGV         1
#define TAREFA_NORMAL       2
#define TAREFA_POSICION     3
#define TAREFA_CARGA        4
#define TAREFA_DESCARGA     5
#define CANCELA_TAREFA      6


/* outras constantes associadas a mensagens SGM -> SGAGV */

#define ALOCA_AGV 1
#define NAO_ALOCA_AGV    2
#define QUALQUER  0
#define UMA_TAREFA 3
#define TODAS_TAR_EXEC   4
#define TODAS_TAR 5


/* CONSTANTES ASSOCIADAS A MENSAGENS ENTRE O SGAGV E O SGM */


/* tipos de mensagens SGAGV -> SGM */

#define CONFIRMACAO      7
#define TAREFA_CANCEL    8
#define ESTADO_AGV 9
#define ESTADO_TAREFA    10
#define T_POSIC_ESPEC    11


/* outrais constantes associadas a mensagens SGAGV -> SGM */

#define SUCESSO          6
#define INSUCESSO 7


/* erross mensagem do tipo CONFIRMACAO, INSUCESSO (SGAGV -> SGM) */

#define AGV_INEX  1
#define AGV_INDISPON     2
#define AGV_CARR  3
#define AGV_DESCARR      4
#define TAREFA_INEX      5
#define ORIGEM_INVAL     6
#define DESTINO_INVAL    7
#define NO_INVALIDO      8
#define CAMINHO1_INEX    9
#define CAMINHO2_INEX    10
#define OP_CAR_IMP 11
#define OP_DES_IMP 12


/* consttantes do campo <razao> da mensagem do tipo TAREFA_CANCEL */
```

```
#define ORDEM_SGM          0
#define AGV_DESACTIVADO    1
#define DEADLOCK           2



/*
MENSAGENS SGAGV -> SPAGV

tipo_mens   p1             p2
MOVIMENTO   <tipo_mov>     <param>
INF_EST_AGV <estado_movim> -
ESTADO_INTR <result_intr>  -
CANCEL_MOV_AVAN    -              -


MENSAGENS SPAGV -> SGAGV

tipo_mens   p1      p2

FIM_MOVIMENTO       -       -
INTRODUCAO  <no>    <orient>
ERRO_AGV    -       -
FIM_ERRO_AGV        -       -
AGV_MANUAL  -       -
MOV_AVAN_CANCEL     -       -
*/


/* CONSTANTES ASSOCIADAS A MENSAGENS ENTRE O SGAGV E O SPAGV */


/* tipos de mensagens SGAGV -> SPAGV */

#define MOVIMENTO          0
#define INF_EST_AGV                1
#define ESTADO_INTR                2
#define CANCEL_MOV_AVAN    3


/* constantes do campo <tipo_mov> da mensagem do tipo MOVIMENTO */

#define MOV_TRANSL 1       /* <param> e o no */
#define MOV_CARGA  2       /* <param> e o lado */
#define MOV_DESCARGA       3       /* idem */


/* constantes que definem o lado nos movimentos de carga/descarga */

#define ESQUERDA   2
#define DIREITA            0


/* constantes do campo <result_intr> da mensagem do tipo ESTADO_INTR */

#define AGV_INTROD         0
```

```
#define AGV_NOT_ENABLED   1
#define NO_INEX                    2
#define NO_OCUPADO        3


/* CONSTANTES ASSOCIADAS A MENSAGENS ENTRE O SPAGV E O SGAGV */


/* tipos de mensagens SPAGV -> SGAGV */

#define FIM_MOVIMENTO             4
#define INTRODUCAO        5
#define ERRO_AGV          6
#define FIM_ERRO_AGV              7
#define AGV_MANUAL        8
#define MOV_AVAN_CANCEL   9



/* prototipos de funcoes utilizaveis em todo o sistema */

void escr_str_est_movim_agv(/* char *s, ulint est_movim */);
void escr_str_est_aloc_agv(/* char *s, ulint est_movim, ulint est_aloc
*/);
void escr_str_est_carga_agv(/* char *s, ulint est_movim, ulint est_carga
*/);
void escr_str_tipo_tarefa(/* char *s, ulint tipo_tarefa */);
void escr_str_est_tarefa(/* char *s, ulint est_tarefa */);


/* Constantes booleanas */

#define FALSO             0
#define VERDADEIRO 1
```

A.1.10 sgm.h

```
/* sgm..h */


/* Constantes */

#define TOLER_IGUAL_DOUBLES 1E-10 /* tolerancia de igualdade de doubles
*/
#define MAX_DOUBLE 1.7E308        /* valor maximo do tipo double */


/* estrutura utilizada para representar um AGV no SGM */
/* (no  de uma lista ligada de AGV's) */

typedef struct s_AGV{
    uint id_agv;
    uint estado_movim;
    uint estado_alocacao;
    uint estado_carga;
    uint ult_destino;
    struct s_Tarefa *tarefa_agv;
```

```
    double t_enable;      /* instante em que ocorrera o ENABLE do AGV
                          (AGV DISABLED) */
    double t_disable;     /* instante em que ocorrera o DISABLE do AGV
                          (AGV nao DISABLED) */
    double t_mud_est_aloc;      /* instante em que ocorrera a mudanca
do estado
                          de alocacao (AGV nao DISABLED) */
    struct s_AGV *seguinte, *anterior;
    } AGV;

typedef AGV* ap_AGV;


/* estrutura utilizada para representar uma Tarefa no SGM */
/* (no de uma lista ligada de Tarefas) */

typedef struct s_Tarefa{
    ulint id_tar;
    ulint tipo_tar;
    ulint prior;
    ulint no1; /* origem tar. normal ou ponto de carga ou posicion. */
    ulint no2; /* destino tar. normal ou ponto de uma descarga */
    ulint estado_tar;
    ap_AGV agv_tarefa;
    struct s_Tarefa *seguinte, *anterior;
    } Tarefa;

typedef Tarefa* ap_Tarefa;


/* prototipos de funcoes para leitura e validacao
do ficheiro de configuracao do SGM */

ap_no_l_uid le_f_nos_tarefas(/* char *nome_ficheiro */);
int som_prob_nos_unit(/* ap_no_l_uid nos_tar */);


/* prototipos de funcoes para manipulacao das listas de AGV's */

int insere_AGV(/* ap_AGV agv, ap_AGV *p, ap_AGV *u */);
int remove_AGV(/* ap_AGV agv, ap_AGV *p, ap_AGV *u */);
ap_AGV obtem_AGV(/* ulint id, ap_AGV p */);
void limpa_l_AGVs(/* ap_AGV p */);
void prn_agv(/* ap_AGV agv */);
void prn_l_AGVs(/* ap_AGV p */);


/* prototipos de funcoes para manipulacao das listas de Tarefas */

int insere_Tarefa(/* ap_Tarefa tarefa, ap_Tarefa *p, ap_Tarefa *u */);
int remove_Tarefa(/* ap_Tarefa tarefa, ap_Tarefa *p, ap_Tarefa *u */);
ap_Tarefa obtem_Tarefa(/* ulint id, ap_Tarefa p */);
void limpa_l_Tarefas(/* ap_Tarefa p */);
void prn_Tarefa(/* ap_Tarefa tar */);
void prn_l_Tarefas(/* ap_Tarefa p */);


/* prototipos de outras funcoes utilizadas na RP do SGM */
```

```
/*void proc_estado_agv(double t, mensag_SGM_SGAGV *m, ap_AGV p_agvs);*/
void proc_estado_tarefa(/* double t, mensag_SGM_SGAGV *m,
    ap_Tarefa tar, ap_AGV p_agvs */);
ap_Tarefa proc_t_posic_espec(/* double t, mensag_SGM_SGAGV *m,
    ap_Tarefa *p_tars, ap_Tarefa *u_tars, ap_AGV p_agvs */);
double calc_carga(/* ulint n_tars, ulint n_agvs_activos */);
int existe_AGV_nova_tarefa(/* ap_AGV p_agvs, double p_esc_sgm */);
ap_AGV deter_AGV_nova_tarefa(/* ap_AGV p_agvs */);
ulint deter_tipo_tarefa(/* ap_AGV agv_tarefa, double p_tar_carga, double
p_posicion,
    double p_tar_carga_agv */);
ulint deter_prior_tar(/* double p_tar_prior, ulint prior_tar_max */);
ulint decide_no_tarefa(/* ap_no_l_uid nos_tarefas */);
ulint decide_tipo_cancel_tar(/* double p_l_tar, double p_todas_exec */);
ulint decide_tar_a_cancelar(/* ap_Tarefa p_tars, ulint n_tars */);
```

A.1.11 sgagv.h

```
/* sgagv.h */


/* Regras de Despacho disponiveis no SGAGV */

 /* do tipo DCT */
#define VMP 0
#define VMU 1

 /* do tipo DV */
#define TMCV       0
#define MTFS       1
#define MELFS      2
#define PCPSM      3



/* TIPOS DE DADOS */


/* tipos de dados para determinacao do trajecto optimo entre 2 pontos */

typedef  struct s_Ramo{
    struct s_No *cabeca;
    ulint custo;
    struct s_Ramo *seguinte;
    } Ramo;

typedef  Ramo* ap_Ramo;

typedef  struct s_No{
    ulint id_no;
    ulint w;
    struct s_No *pn;
    ap_Ramo ramos;
    struct s_No *seguinte;
    } No;

typedef  No* ap_No;
```

```
typedef struct s_no_l_apnodtraj{
    ap_No no;
    struct s_no_l_apnodtraj *anterior, *seguinte;
    } no_l_apnodtraj;

typedef no_l_apnodtraj* ap_no_l_apnodtraj;

typedef struct s_no_l_apno{
    ap_no_l_nos no;
    ulint custo;  /* custo associado ao movimento entre
          o no anterior do trajecto e o proprio no */
    struct s_no_l_apno *anterior, *seguinte;
    } no_l_apno;

typedef no_l_apno* ap_no_l_apno;



/* AGV's */

typedef struct s_AGV{
    ulint id_agv;
    ap_no_l_nos posicao;
    ap_no_l_nos ult_posicao;
    int orient;          /* orient. do AGV em relacao ao sent. + */
    ulint estado_movim;
    ulint estado_alocacao;
    ulint estado_carga;
    ulint est_mov_ant;
    struct s_Tarefa *tarefa_agv;
    ulint num_mov_avan;
    double t_mud_estado; /* instante em que passou ao estado actual */

    /* estatisticas */
    double dt_espera;
    double dt_movim_descarr;
    double dt_movim_carr;
    double dt_bloqueado;
    double dt_carga;
    double dt_descarga;
    double dt_erro;
    double dt_enabled;
    double dt_disabled;
    double taxa_util;
    ulint n_arr_carr;
    ulint n_arr_descarr;
    ulint n_mov_carr;    /* trajectos completos */
    ulint n_mov_descarr; /* idem */
    ulint n_cargas;
    ulint n_descargas;

    struct s_AGV *seguinte;
    } AGV;

typedef AGV* ap_AGV;

typedef struct s_no_l_ap_AGVs{
    ap_AGV agv;
```

```
    struct s_no_l_ap_AGVs *seguinte;
    } no_l_ap_AGVs;

typedef no_l_ap_AGVs* ap_no_l_ap_AGVs;


/* Tarefas */

typedef struct s_Tarefa{
    ulint id_tar;
    ulint tipo_tar;
    ulint prior;
    ap_no_l_nos no1; /* origem tar. normal ou ponto de carga ou
posicion. */
    ap_no_l_nos no2; /* destino tar. normal ou ponto de uma descarga */
    ulint estado_tar;
    ulint est_antes_desp;
    ap_AGV agv_tarefa;
    ulint ctraj1, ctraj2; /* custo associado aos trajectos calculados */
    ap_no_l_apno traj1, traj2;
    double t_mud_estado; /* instante em que passou ao estado actual */
    double t_lancam;     /* instante em que a tarefa e lancada */

    /* estatisticas */
    double dt_fila;
    double dt_p1;
    double dt_p2;
    double dt_p3;
    double dt_p4;
    ulint num_af;

    struct s_Tarefa *seguinte;
    } Tarefa;

typedef Tarefa* ap_Tarefa;

typedef struct s_no_l_ap_Tarefas{
    ap_Tarefa tarefa;
    struct s_no_l_ap_Tarefas *seguinte;
    } no_l_ap_Tarefas;

typedef no_l_ap_Tarefas* ap_no_l_ap_Tarefas;


/* Bloqueios de No´s do Layout */

typedef struct s_bloqueio{
    ap_no_l_nos no;
    ap_AGV agv;
    struct s_bloqueio *seguinte;
    } bloqueio;

typedef bloqueio* ap_bloqueio;


/* AGV Bloqueado */
```

```
typedef struct s_AGV_bloqueado{
    ap_AGV agv;                    /* AGV bloqueado */
    ap_AGV agv_bloq;               /* AGV bloqueante*/
    ap_no_l_ramos ramo;   /* ramo associado ao bloqueio */
    ulint custo_adicional;         /* custo adicional de desbloqueamento
*/
    struct s_AGV_bloqueado *seguinte;
    } AGV_bloqueado;

typedef AGV_bloqueado* ap_AGV_bloqueado;



/* Estado das Filas */

typedef struct s_EstadoFila{
    ulint id, Cf, Lf, Rf;
    ap;Tarefa tar_mais_ant;
    double t_MFCFS;
    struct s_EstadoFila *seguinte;
    } EstadoFila;

typedef EstadoFila* ap_EstadoFila;



/* PROTOTIPOS DE FUNCOES */


/* funcoes para manipulacao de listas de apontadores para nos do layout
*/

ap_no_l_apnodtraj procura_apnodtraj(/* ap_No no, ap_no_l_apnodtraj h);
int insere_apnodtraj(/* ap_No no, ap_no_l_apnodtraj *h,
ap_no_l_apnodtraj *u,
    char repetidos */);
int remove_apnodtraj(/* ap_No no, ap_no_l_apnodtraj *h,
ap_no_l_apnodtraj *u */);
void limpa_l_apnodtraj(/* ap_no_l_apnodtraj h */);


/* funcoes para manilupacao de listas representando trajectos */

int insere_a_cabeca_l_apno(/* ap_no_l_nos no, ulint custo, ap_no_l_apno
*h */);
void remove_cabeca_l_apno(/* ap_no_l_apno *h */);
void limpa_l_apno(/* ap_no_l_apno *h */);
void prn_l_apno(/* ap_no_l_apno h */);


/* funcoes para determinacao do trajecto optimo entre 2 pontos */

ap_No obtem_no_lay_dtraj(/* ulint id, ap_No h */);
void limpa_ramos_no_dtraj(/* ap_Ramo h */);
void limpa_lay_dtraj(/* ap_No h */);
void elimina_nos_isolados(/* ap_No *h, ap_no_l_nos origem */);
ap_No cria_lay_dtraj(/* ap_no_l_nos layout, ap_no_l_nos origem,
ap_no_l_nos destino,
```

```
 ap_no_l_ulint nos_a_ignorar, ap_no_l_ramos ramo_a_ignorar, int
poss_acesso */);
void general_labeling_algorithm(/* ap_No nos, ap_No origem */);
ap_no_l_apno obtem_trajecto(/* ap_No destino, ap_no_l_nos layout */);
ulint traj_optimo(/* ap_no_l_nos layout, ap_no_l_nos origem,
 ap_no_l_nos destino, ap_no_l_ulint nos_a_ignorar, ap_no_l_ramos
ramo_a_ignorar,
 int poss_acesso, ap_no_l_apno *traj_opt */);
ulint custo_traj_optimo(/* ap_no_l_nos layout, ap_no_l_nos origem,
 ap_no_l_nos destino, ap_no_l_ulint nos_a_ignorar, ap_no_l_ramos
ramo_a_ignorar,
 int poss_acesso */);


/* funcoes para manipulacao de listas de apontadores para AGV's */

int insere_no_l_ap_AGVs(/* ap_AGV agv, ap_no_l_ap_AGVs *h,
ap_no_l_ap_AGVs *t */);
void remove_cab_l_ap_AGVs(/* ap_no_l_ap_AGVs *h */);
void limpa_l_ap_AGVs(/* ap_no_l_ap_AGVs *h */);
ap_AGV obtem_ap_agv_pos(/* ap_no_l_ap_AGVs h, ulint pos */);
void prn_l_ap_AGVs(/* ap_no_l_ap_AGVs h */);



/* funcoes para manipulacao de listas de apontadores para Tarefas */

int existe_no_l_ap_Tarefas(/* ap_Tarefa tarefa, ap_no_l_ap_Tarefas h
*/);
int insere_no_l_ap_Tarefas(/* ap_Tarefa tarefa, ap_no_l_ap_Tarefas *h,
                              ap_no_l_ap_Tarefas *t */);
void remove_cab_l_ap_Tarefas(/* ap_no_l_ap_Tarefas *h */);
void limpa_l_ap_Tarefas(/* ap_no_l_ap_Tarefas *h */);
ap_Tarefa obtem_ap_tarefa_pos(/* ap_no_l_ap_Tarefas h, ulint pos */);
void prn_l_ap_Tarefas(/* ap_no_l_ap_Tarefas h */);


/* funcoes para manipular a matriz de eficiencia */

ulint* aloca_matriz_quad(/* ulint dim */);
void escr_elem(/* ulint *m, ulint dim, ulint l, ulint c, ulint i */);
ulint lee_elem(/* ulint *m, ulint dim, ulint l, ulint c */);
void prn_matriz(/* ulint *m, ulint dim */);
void destroi_matriz(/* ulint *m */);


/* funcoes para escalonamento e despacho */

ulint deter_num_af(/* ap_Tarefa tarefas, ap_AGV agvs, ulint num_max_af,
    int reaf_dinam */);
int forma_l_tar_e_agvs_af(/* ulint num_af, ap_Tarefa tarefas, ap_AGV
agvs,
    int reaf_dinam, ap_no_l_ap_Tarefas *tar_af, ulint *num_tar_af,
    ap_no_l_ap_AGVs *agvs_af, ulint *num_agvs_af */);
ulint* forma_matr_efic(/* ulint num_af, ap_no_l_ap_Tarefas tarefas,
ulint num_tar_af,
  ap_no_l_ap_AGVs agvs, ulint num_agvs_af, ap_no_l_nos layout, int
poss_acesso */);
void alg_hung_p1(/* ulint *m, ulint dim */);
```

```
int forma_zl_zc_lzeros(/* ulint *m, ulint dim, ap_no_l_ulint *zl,
    ap_no_l_ulint *zc, ap_no_l_2uint *lzeros */);
void act_zl(/* ulint *m, ulint dim, ulint c, ap_no_l_ulint zl */);
void act_zc(/* ulint *m, ulint dim, ulint l, ap_no_l_ulint zc */);
int forma_l_zeros(/* ulint *m, ulint dim, ap_no_l_2uint *lzeros */);
void limpa_zeros_linha(/* ulint linha, ap_no_l_2uint *lzeros */);
void limpa_zeros_coluna(/* ulint coluna, ap_no_l_2uint *lzeros */);
int cobre_zeros(/* ap_no_l_2uint lzeros, ap_no_l_ulint *ll,
ap_no_l_ulint u_ll,
  ulint *tam_ll, ap_no_l_ulint *lc, ap_no_l_ulint u_lc, ulint *tam_lc
*/);
int alg_hung_p2(/* ulint *m, ulint dim, ap_no_l_ulint *ll,
    ulint *tam_ll, ap_no_l_ulint *lc, ulint *tam_lc */);
ulint alg_hung_p4(/* ulint *m, ulint dim, ap_no_l_ulint ll,
ap_no_l_ulint lc */);
void alg_hung_p5(/* ulint *m, ulint dim, ap_no_l_ulint ll, ap_no_l_ulint
lc, ulint emin */);
ap_no_l_2uint alg_hung_p6(/* ulint *m, ulint dim */);
ap_no_l_2uint algoritmo_hungaro(/* ulint *m, ulint dim */);
void act_lagvs_e_ftar(/* ap_no_l_2uint l_af, ap_no_l_ap_Tarefas tar_af,
  ulint num_tar_af, ap_no_l_ap_AGVs agvs_af, ulint num_agvs_af,
  ap_no_l_nos layout, int poss_acesso, ap_no_l_ap_Tarefas *tar_mud_est,
  ap_no_l_ap_AGVs *agvs_reaf, double t */);
void escal_desp_VMP_TMCV(/* ap_Tarefa tarefas, ap_AGV agvs,
  ap_no_l_nos layout, int poss_acesso, ulint num_max_af,
  int reaf_dinam, ap_no_l_ap_Tarefas *tar_mud_est,
  ap_no_l_ap_AGVs *agvs_reaf, double t */);

    /* outras regras de despacho... */
void determ_primeiro_AGV_dispon_e_tarefa_pendente(/* ap_Tarefa tarefas,
    ap_AGV agvs, ap_Tarefa *tarefa, ap_AGV *agv */);
int cria_lista_tar}fas_escalonaveis(/* ap_Tarefa tarefas, ulint
num_max_af,
    ap_no_l_ap_Tarefas *tar_escalonaveis */);
int cria_lista_agvs_escalonaveis(/* ap_AGV agvs, ulint num_max_af,
    ap_no_l_ap_AGVs *agvs_escalonaveis */);
ap_Tarefa aplica_regra_TMCV(/* ap_no_l_ap_Tarefas tarefas, ap_AGV agv,
    ap_no_l_nos layout, int poss_acesso */);
ap_Tarefa aplica_regra_MTFS(/* ap_no_l_ap_Tarefas tarefas,
    ap_EstadoFila est_filas */);
ap_Tarefa aplica_regra_MELFS(/* ap_no_l_ap_Tarefas tarefas,
    ap_EstadoFila est_filas */);
ap_Tarefa aplica_regra_PCPSM(/* ap_no_l_ap_Tarefas tarefas,
    ap_EstadoFila filas */);
ap_AGV aplica_regra_VMP(/* ap_Tarefa tarefa, ap_no_l_ap_AGVs agvs,
    ap_no_l_nos layout, int poss_acesso */);
ap_AGV aplica_regra_VMU(/* ap_no_l_ap_AGVs agvs, double t */);
void afecta_agv_a_tarefa(/* ap_AGV agv, ap_Tarefa tarefa, ap_no_l_nos
layout,
    int poss_acesso, double t, ap_no_l_ap_Tarefas *tar_mud_est */);

    /* funcao de escalonamento e de despacho principal */
void escalon_despacho(/* int regra_ADCT, int regra_ADV,
  ap_Tarefa tarefas, ap_AGV agvs,
  ap_no_l_nos layout, int poss_acesso, ulint num_max_af,
  int reaf_dinam, ap_EstadoFila est_filas, double t,
  ap_no_l_ap_Tarefas *tar_mud_est, ap_no_l_ap_AGVs *agvs_reaf */);
```

```
/* funacoes para manipulacao de listas de AGV's */

int innsere_AGV_DISABLED(/* ulint id_agv, ap_AGV *h, ap_AGV *t */);
void pprn_el_l_AGVs(/* ap_AGV agv */);
void pprn_AGVs(/* ap_AGV h */);
void llimpa_lista_AGVs(/* ap_AGV *h */);
ap_AGV/ obtem_ap_AGV(/* ulint id_agv, ap_AGV h */);


/* funacoes para manipulacao de listas de Bloqueios de No´s do Layout */

int innsere_bloqueio(/* ap_no_l_nos no, ap_AGV agv, ap_bloqueio *h,
ap_blooqueio *t */);
void iinsere_bloqueios(/* ap_no_l_nos no, ap_AGV agv, ap_bloqueio *h,
ap_blooqueio *t,
    app_no_l_nos layout */);
void iinsere_bloqs_movim(/* ap_AGV agv, ap_no_l_nos no_origem,
    ullint id_no_destino, ap_no_l_nos layout, ap_bloqueio *h,
    app_bloqueio *t */);
ap_blooqueio obtem_bloqueio(/* ulint id_no, ap_bloqueio h */);
ap_blooqueio bloq_no_indisponivel(/* ap_no_l_nos no, ap_AGV agv,
ap_blooqueio h */);
ap_blooqueio bloq_movim_inviavel(/* ap_AGV agv, ap_no_l_nos no_origem,
    ullint id_no_destino, ap_bloqueio est_lay */);
void rremove_bloqueios_agv(/* ap_AGV agv, ap_bloqueio *h, ap_bloqueio *t
*/);
void llimpa_l_bloqueios(/* ap_bloqueio *h */);
void pprn_l_bloqueios(/* ap_bloqueio h */);


/* funncoes para manipulacao de listas de Tarefas */

void iinicial_tarefa(/* ap_Tarefa tar, ulint id_tar, ulint tipo_tar,
ulint : prior,
    aap_no_l_nos no1, ap_no_l_nos no2, double t */);
int innsere_tarefa(/* ap_Tarefa tar, ap_Tarefa *h, ap_Tarefa *t */);
void pprn_el_l_tarefas(/* ap_Tarefa tar */);
void pprn_tarefas(/* ap_Tarefa h */);
void llimpa_lista_tarefas(/* ap_Tarefa *h */);
ap_Tarefa obtem_ap_tarefa(/* ulint id_tar, ap_Tarefa h */);
void rremove_tarefa(/* ulint id_tar, ap_Tarefa *h, ap_Tarefa *t */);


/* fuuncoes para manipulacao de listas representando AGV's bloqueados */

int iinsere_AGV_bloq(/* ap_AGV agv, ap_AGV agv_bloq, ap_no_l_ramos ramo,
            ap_AGV_bloqueado *p, ap_AGV_bloqueado *u */);
ap_AGV_bloqueado obtem_AGV_bloq(/* ap_AGV agv, ap_AGV_bloqueado p */);
void  limpa_l_AGVs_bloq(/* ap_AGV_bloqueado *p */);
void  prn_l_AGVs_bloq(/* ap_AGV_bloqueado p */);


/* ouutras funcoes utilizadas na RP do SGAGV */

void  act_estat_temporais_agv(/* ap_AGV agv, double t */);
```

```
void act_estat_temporais_tarefa(/* ap_Tarefa tar, double t */);
ulint deter_num_tar_cancel(/* ap_Tarefa h1, ulint tipo_cancel, ulint
id_tar,
    ap_no_l_ap_Tarefas *h2 */);
int ha_l_AGV_disponivel(/* ap_AGV h */);
int ha_l_Tar_n_escalon(/* ap_Tarefa h */);
ap_AGV agv_PARADO_BLOQUEADO(/* ap_AGV h */);
void preenche_mens_ESTADO_AGV(/* mensag_SGM_SGAGV *m, ap_AGV agv */);
void preenche_mens_ESTADO_TAREFA(/* mensag_SGM_SGAGV *m, ap_Tarefa tar
*/);
ap_AGV agv_BLOQUEADO_MOVIMENTO(/* ap_AGV h, ap_bloqueio est_lay */);
ap_AGV agv_BLOQUEADO_CD(/* ap_AGV h */);
ap_no_l_apno obtem_no_traj_mov_avan(/* ap_AGV agv */);
ap_AGV agv_enviar_mov_avan(/* ap_AGV h, ap_bloqueio est_lay,
    ulint num_max_mov_avan */);
void proc_tarefa_concluida(/* ap_Tarefa tar, double t */);
ap_Tarefa tarefa_traj_nulo(/* ap_AGV h */);
ap_AGV agv_BLOQUEADO_PARADO(/* ap_AGV h */);
int ha_agv_bloq_por_agv_PARADO(/* ap_AGV h, ap_no_l_nos layout,
    ap_bloqueio est_lay, ulint num_max_mov_avan */);
ap_AGV agv_bloq_por_agv_PARADO(/* ap_AGV h, ap_no_l_nos layout,
    ap_bloqueio est_lay, ulint num_max_mov_avan,
    ap_AGV *agv_bloqueante, ap_no_l_ramos *r_no_escape */);
ap_no_l_ulint forma_l_nos_oc_AGVs_ERRO(/* ap_bloqueio est_lay */);
ap_AGV_bloqueado obtem_deadlock_circular(/* ap_AGV h, ap_no_l_nos
layout,
  ap_bloqueio est_lay, int poss_acesso, ap_no_l_ulint
*p_nos_oc_agvs_erro */);
int existe_deadlock_circular(/* ap_AGV h, ap_no_l_nos layout,
    ap_bloqueio est_lay, int poss_acesso */);
void resolve_deadlock_circular(/* ap_AGV_bloqueado agvs_dead,
  ap_no_l_nos layout, int poss_acesso, ap_no_l_ulint p_nos_oc_agvs_erro
*/);
int cam_bloq_nos_ocup(/* ap_no_l_nos pos,
    ap_no_l_apno traj, ap_no_l_ulint nos_oc */);
int ha_agvs_bloq_por_agvs_ERRO(/* ap_AGV h, ap_bloqueio est_lay,
    ap_no_l_nos layout, int poss_acesso */);
void resolve_bloq_por_agvs_ERRO(/* ap_AGV h, ap_bloqueio est_lay,
 ap_no_l_nos layout, int poss_acesso, ap_no_l_ap_AGVs *agvs_redir */);
void act_dados_MFCFS_fila(/* ap_EstadoFila fila, ap_Tarefa h */);


/* funcoes para manipulacao de listas de elementos do tipo EstadoFila */

ap_EstadoFila procura_EstadoFila(/* ulint id, ap_EstadoFila primeiro
*/);
int insere_EstadoFila(/* ulint id, ulint Cf, ap_EstadoFila *primeiro,
    ap_EstadoFila *ultimo */);
void limpa_l_EstadoFila(/* ap_EstadoFila *h */);
void prn_EstadoFila(/* ap_EstadoFila p */);
void prn_l_EstadoFila(/* ap_EstadoFila h */);
```

A.1.12 spagv.h

```
/* spagv.h */
```

```
#define TOLER_IGUAL_DOUBLES 1E-10 /* tolerancia de igualdade de doubles
*/


/* estrutura de dados que guarda a informacao sobre o layout, necessaria
para o SPAGV estimar os tempos associados aos movimentos atraves dos
ramos */

struct s_No_SP;

typedef struct s_Ramo_SP{
    struct s_No_SP *cabeca;
    double tempo; /* em segundos (com precisao ate `as decimas de seg */
    struct s_Ramo_SP *seguinte;
    } Ramo_SP;

typedef Ramo_SP* ap_Ramo_SP;

typedef struct s_No_SP{
    ulint id_no;
    ulint tipo_no;
    int pos_rel_mesa;
    ap_Ramo_SP ramos;
    struct s_No_SP *seguinte;
    } No_SP;

typedef No_SP* ap_No_SP;


/* prototipos de funcoes para leitura e validacao
dos ficheiros de configuracao do SPAGV */

void limpa_l_ramos_SP(/* ap_Ramo_SP h */);
void limpa_l_nos_SP(/* ap_No_SP */);
ap_No_SP get_no_lay_SP(/* ulint id, ap_No_SP h */);
ap_No_SP le_layout_SP(/* char *n_fich_lay, char *n_fich_mesas,
    ulint vre, ulint vcr, ulint vma */);
void prn_ap_No_SP(/* ap_No_SP ap_no */);
void prn_lay_SP(/* ap_No_SP h */);
ap_no_l_ui4d le_f_tempos_op_cd(/* char *nome_ficheiro */);
int tt_op_mesas_conhec(/* ap_No_SP nos, ap_no_l_ui4d t_op_cd */);
ap_no_l_uid le_f_nos_introd(/* char *nome_ficheiro */);
int ssom_prob_unit(/* ap_no_l_uid nos_intr */);


/* prototipos de outras funcoes utilizadas na RP do SPAGV */

ulintt decide_no_intr(/* ap_no_l_uid nos_intr */);
double processa_mov_transl(/* mensag_SGAGV_SPAGV mens, ap_No_SP *posicao
*/);
void processa_mov_car_des(/* mensag_SGAGV_SPAGV mens, ap_No_SP posicao
*/);
```

A.2 Ficheiros fonte (source files)

A.2.1 fun_aux.c

```
/* fun_aux.c */

#include <math.h> /* fabs() */
#include <stdio.h>
#include "listas.h"
#include "fun_aux.h"



/* prototipos de funcoes auxiliares */


int dligual2(d1, d2, tol)
 double d1, d2, tol;
{
 if (fabs(d1-d2)<=tol) return 1; else return 0;
}


ulint max_ulint(i1, i2)
/*
- retorna o maximo de 2 inteiros sem sinal
*/
ulint i1, i2;
{ if (i1>i2) return i1; else return i2; }


ulint min_ulint(i1, i2)
/*
- retorna o minimo de 2 inteiros sem sinal
*/
ulint i1, i2;
{ if (i1<i2) return i1; else return i2; }

void prn_debug(s)
/*
- funcao utilizada para fazer debug
*/
  char *s;
{
  printf("\n%s", s);
  getc(stdin);
}
```

A.2.2 listas.c

```
/* listas.c */

/* definicao das funcoes para manipulacao de listas ligadas */


#include <stdio.h>
#include <string.h>
```

```
#include "listas.h"

char s[255]; /* buffer utilizado para chamar 'prn_string()' */

/* **** listas ligadas de inteiros sem sinal *** */

ap_no_l_ulint get_ultimo_ulint(primeiro)
/*
- retorna um apontador para o ultimo elemento da lista
*/
 ap_no_l_ulint primeiro;
{
 while (primeiro!= NULL && primeiro->seguinte!=NULL)
     primeiro=primeiro->seguinte;
 return primeiro;
}


ap_no_l_ulint procura_ulint(i, primeiro)
/*
- retorna o 1. elemento encontrado ou NULL se nao for encontrado nenhum
- primeiro e cabeca da lista ligada
*/
 ulint i;
 ap_no_l_ulint primeiro;
{
 while (primeiro!=NULL && primeiro->i!=i) primeiro=primeiro->seguinte;
 return primeiro;
}


int insere_ulint(i, primeiro, ultimo, repetidos)
/*
- insere um novo elemento no fim da lista ligada
- permite ou nao repetidos consoante o valor passado em 'repetidos'
- actualiza os apontadores *primeiro e *ultimo
- retorna 1 quando bem sucedida ou 0 se ocorrer algum erro
*/
  ulint i;
  ap_no_l_ulint *primeiro, *ultimo;
  char repetidos;
{
  ap_no_l_ulint novo;

  if (repetidos || procura_ulint(i, *primeiro)==NULL)
  {
    novo= (ap_no_l_ulint) malloc(sizeof(no_l_ulint));
    if (novo!=NULL)
    {
      novo->i=i;
      novo->seguinte=NULL;
      novo->anterior=*ultimo;
      if (*ultimo!=NULL) (*ultimo)->seguinte=novo;
      *ultimo=novo;
      if (*primeiro==NULL) *primeiro=novo;
      return 1;
```

```
    }
    else prn_string("\"insere_ulint()\": erro de alocacao de
memoria\n");
  }
  /*else printf("\"insere_ulint()\": elemento repetido nao
inserido\n");*/
  return 0;
}


int remove_ulint(i, primeiro, ultimo)
/*
- remove da lista o elemento i
- actualiza *primeiro e *ultimo quando isso for necessario
- retorna 1 quando e bem sucedida ou 0 senao
*/
  ulint i;
  ap_no_l_ulint *primeiro, *ultimo;
{
  ap_no_l_ulint e;

  if ((e=procura_ulint(i, *primeiro))!=NULL)
  {
    if (e!=*primeiro) e->anterior->seguinte=e->seguinte;
    else *primeiro=e->seguinte;
    if (e!=*ultimo) e->seguinte->anterior=e->anterior;
    else *ultimo=e->anterior;
    free((void *) e);
    return 1;
  }
  else return 0;
}


void limpa_l_ulint(primeiro)
/*
- destroi uma lista, comecando pela cabeca
*/
  ap_no_l_ulint primeiro;
{
  ap_no_l_ulint prox;

  while (primeiro!=NULL)
  {
    prox=primeiro->seguinte;
    free((void *) primeiro);
    primeiro=prox;
  }
}


ap_no_l_ulint append_l_ulint(p_l1, p_l2, repetidos)
/*
- adiciona `a lista cujo ponto de entrada e apontado por p_l1 os
elementos da lista p_l2
- sao adicionados elementos repetidos consoante repetidos valha 1 ou 0,
respectivamente
- *p_l1 e actualizado quando a 1. lista e vazia
- retorna um apontador para o ultimo elemento da lista
```

```
*/
 ap_no_l_ulint *p_l1;
 ap_no_l_ulint p_l2;
 chair repetidos;
{
 ap_no_l_ulint u;

 /* procura o ultimo elemento da 1. lista */
 u=get_ultimo_ulint(*p_l1);

 while (p_l2!=NULL) /* para todos os elementos da 2. lista ... */
 {
  insere_ulint(p_l2->i, p_l1, &u, repetidos);
  p_l2=p_l2->seguinte;
 }
 return u;
}


voidi prn_l_ulint(primeiro)
/*
- mostra no stdout o conteudo da lista ligada cuja cabeca e primeiro
*/
  app_no_l_ulint primeiro;
{
  app_no_l_ulint p=primeiro;

  whhile (p!=NULL)
  {
     if (p!=primeiro) sprintf(s, ",%u", p->i);
     else sprintf(s, "%u", p->i);
     prn_string(s);
     p=p->seguinte;
  }
}


voidd max_l_ulint(primeiro, mx, n_mx)
/*
- deetermina o maximo dos elementos da lista de inteiros sem sinal (*mx)
e
o nuumero de vezes que ocorre na lista (*n_mx)
- see a lista for vazia retorna *mx=0 e *n_mx=0
*/
  app_no_l_ulint primeiro;
  ullint *mx;
  ullint *n_mx;
{
  *mmx=0;
  *rn_mx=0;

  while (primeiro!=NULL)
  {
     if (primeiro->i > *mx) { *mx=primeiro->i; *n_mx=1; }
       else if (primeiro->i == *mx) (*n_mx)++;
     primeiro=primeiro->seguinte;
  }
}
```

```
void min_l_ulint(p, lim_min, m, im)
/*
- determina o minimo dos elementos da lista (*m), retornando o indice do
primeiro elemento igual a *m (*im)
- o minimo e limitado por lim_min
- se a lista for vazia ou se nao for encontrado qq elemento >= lim_min-
1,
retorna *m=MAXINT e *im=0
*/
  ap_no_l_ulmnt p;
  ulint lim_min;
  ulint *m;
  ulint *im;
{
  ulint i=1;

  *m=MAXINT;
  *im=0;

  while (p!=NULL)
  {
    if (p->i < *m && p->i >= lim_min)
    {
      *m=p->i;
      *im=i;
    }
    i++;
    p=p->seguinte;
  }
}


void atribui_ulint_pos(i, pos, p)
/*
- modifica o campo i do elemento que ocupe a posicao pos na lista,
pressupondo que a lista e percorrida no sentido cabeca->cauda a
partir de p
- quando nao existir tal elemento, a lista nao e modificada
*/
  ulint i;
  ulint pos;
  ap_no_l_ulint p;
{
  ulint c=1;

  while (p!=NULL && c++<pos) p=p->seguinte;
  if (p!=NULL) p->i=i;
}

ulint elem_l_ulint(pos, p)
/*
- retorna o campo i do elemento que ocupa a posicao pos na lista,
pressupondo que a lista e percorrida no sentido cabeca->cauda a
partir de p
- quando nao existir tal elemento, retorna por defeito 0
*/
  ulint pos;
  ap_no_l_ulint p;
{
```

```
  uulint c=1;

  wwhile (p!=NULL && c++<pos) p=p->seguinte;
  iif (p!=NULL) return(p->i); else return 0;
}

intt inters_igual_vazio(l1, l2)
/*
- vverifica se a interseccao de 2 listas e ou nao igual a lista vazia
- sse for retorna 1, senao retorna 0
*/
  aap_no_l_ulint l1;
  aap_no_l_ulint l2;
{
  wwhile (l1!=NULL)
  {{
     if (procura_ulint(l1->i, l2)!=NULL) return 0;
     l1=l1->seguinte;
  }}
  rreturn 1;
}

intt copia_l_ulint(l1, p_l2, u_l2)
/*
- ccria uma copia de uma lista
*/
  aap_no_l_ulint l1;
  aap_no_l_ulint *p_l2;
  aap_no_l_ulint *u_l2;
{
  uulint noerrors=1;

  *p*p_l2=NULL;
  **u_l2=NULL;

  wwhile (l1!=NULL && noerrors)
  {{
     noerrors=insere_ulint(l1->i, p_l2, u_l2, 1);
     l1=l1->seguinte;
  }}

  iif (!noerrors)
  {{
     limpa_l_ulint(*p_l2);
     *p_l2=NULL;
     *u_l2=NULL;
  }}

  rreturn noerrors;
}


/* *** listas ligadas de estruturas com 2 inteiros sem sinal *** */


ap_no_l_2uint procura_2uint(i1, i2, primeiro)
/*
- rretorna o 1. elemento encontrado que satisfaca i1 ou i2, ou NULL
```

```
se nao for encontrado nenhum
- primeiro e cabeca da lista ligada
*/
 ulint i1, i2;
 ap_no_l_2uint primeiro;
{
 while (primeiro!=NULL && primeiro->i1!=i1 && primeiro->i2!=i2)
    primeiro=primeiro->seguinte;
 return primeiro;
}


int insere_2uint(i1, i2, primeiro, ultimo)
/*
- insere o elemento (i1,i2) no fim da lista
- primeiro e a cabeca da lista
- ultimo e o ultimo elemento inserido na lista (ignorado se valer NULL)
- retorna 1 em caso de sucesso ou 0 em caso de insucesso
*/
  ulint i1;
  ulint i2;
  ap_no_l_2uint *primeiro;
  ap_no_l_2uint *ultimo;
{
  ap_no_l_2uint novo;

  novo= (ap_no_l_2uint) malloc(sizeof(no_l_2uint));
  if (novo!=NULL)
  {
    novo->i1=i1;
    novo->i2=i2;
    novo->seguinte=NULL;
    if (*primeiro==NULL) *primeiro=novo;
    if (*ultimo!=NULL) (*ultimo)->seguinte=novo;
    *ultimo=novo;
    return 1;
  }
  else
  {
   sprintf(s, "\"insere_2uint()\": erro de alocacao de memoria; elemento
(%u,%u) nao inserido\n",i1,i2);
   prn_string(s);
   return 0;
  }
}


ap_no_l_2uint limpa_l_2uint(h)
/*
- elimina os elementos de uma lista
- h e a cabeca da lista
- retorna NULL
*/
 ap_no_l_2uint h;
{
 ap_no_l_2uint proximo;

 while (h!=NULL)
 {
```

```
    proximo=h->seguinte;
    free((void *)h);
    h=proximo;
   }
   return h;
}


void prn_l_2uint(h)
/*
- mostra no stdout o conteudo da lista ligada cuja cabeca e h
*/
   ap_no_l_2uint h;
{
   ap_no_l_2uint p=h;

   while (p!=NULL)
   {
      if (p!=h) prn_string(", ");
      sprintf(s, "(%u, %u)", p->i1, p->i2);
      prn_string(s);
      p=p->seguinte;
   }
}

int copia_l_2uint(l1, l2)
/*
- cria uma copia de uma lista
*/
   ap_no_l_2uint l1;
   ap_no_l_2uint *l2;
{
   ulint noerrors=1;
   ap_no_l_2uint u=NULL;

   *l2=NULL;

   while (l1!=NULL && noerrors)
   {
      noerrors=insere_2uint(l1->i1, l1->i2, l2, &u);
      l1=l1->seguinte;
   }

   if (!noerrors) *l2=limpa_l_2uint(*l2);

   return noerrors;
}

void remove_cab_l_2uint(p)
/*
- remove o 1. elemento da lista, actualizando p
*/
   ap_no_l_2uint *p;
{
   ap_no_l_2uint aux;

   if (*p!=NULL)
   {
      aux=(*p)->seguinte;
```

```
      free(*p);
      *p=aux;
    }
  }

  void remove_2uint(e, p)
  /*
  - remove um elemento da lista, actualizando o apontador para a cabeca da
  lista
  */
    ap_no_l_2uint e;
    ap_no_l_2uint *p;
  {
    ap_no_l_2uint ap=*p, u=NULL;

    while (ap!=NULL && ap!=e)
    {
      u=ap;
      ap=ap->seguinte;
    }

    if (ap!=NULL)
    {
      if (ap==*p) *p=ap->seguinte;
      else u->seguinte=ap->seguinte;
      free(ap);
    }
  }



  /* *** listas ligadas de estruturas com um inteiro sem sinal e 4 doubles
  *** */


  ap_no_l_ui4d get_ui4d_ui(h, i)
  /*
  - retorna o 1. elemento da lista com o 1. campo = i ou NULL senao
  encontrar nenhum
  - h e cabeca da lista ligada
  */
   ap_no_l_ui4d h;
   ulint i;
  {
   while (h!=NULL)
   {
    if (h->i==i) return h;
    h=h->seguinte;
   }
   return NULL;
  }


  ap_no_l_ui4d insere_ui4d(h, ultimo, i, d1, d2, d3, d4, repetidos)
  /*
  - insere o elemento (i,d1,d2,d3,d4) na lista ligada
  - permite ou nao elem. repetidos consoante repetidos valha 1 ou 0,
  respectivamente
```

```
- Hh e a cabeca da lista
- uultimo e o ultimo elemento inserido na lista (ignorado se valer NULL)
- rretorna um apontador para o ultimo elemento da lista
*/
 app_no_l_ui4d h;
 app_no_l_ui4d ultimo;
 ullint i;
 dcouble d1;
 dcouble d2;
 dcouble d3;
 dcouble d4;
 chhar repetidos;
{
 app_no_l_ui4d novo;

 iff (!repetidos && get_ui4d_ui(h,i)!=NULL)
 {
  ssprintf(s, "\"insere_ui4d()\": elemento (%u,%.3f,%.3f,%.3f,%.3f) nao
inserido para evitar elementos repetidos\n",
     i,d1,d2,d3,d4);
  pprn_string(s);
  rreturn ultimo;
 }
 ellse
 {
  rnovo= (ap_no_l_ui4d) malloc(sizeof(no_l_ui4d));
  iif (novo!=NULL)
  {{
    novo->i=i;
    novo->d1=d1;
    novo->d2=d2;
    novo->d3=d3;
    novo->d4=d4;
    novo->seguinte=NULL;
    if (ultimo!=NULL) ultimo->seguinte=novo;
    return novo;
  }}
  eelse
  {{
    sprintf(s, "\"insere_ui4d()\": erro de alocacao de memoria; elemento
(%uu,%.3f,%.3f,%.3f,%.3f) nao inserido\n",
     i,d1,d2,d3,d4);
    prn_string(s);
    return ultimo;
  }}
 }
}


ap__no_l_ui4d limpa_l_ui4d(h)
/*
- eelimina os elementos de uma lista ligada
- Hh e a cabeca da lista
- rretorna NULL
*/
 ap_no_l_ui4d h;
{
 ap_no_l_ui4d proximo;
```

```
 while (h!=NULL)
 {
  proximo=h->seguinte;
  free((void *)h);
  h=proximo;
 }
 return h;
}


void prn_l_ui4d(h)
/*
- mostra no stdout o conteudo da lista ligada
*/
  ap_no_l_ui4d h;
{
  ap_no_l_ui4d p=h;

  while (p!=NULL)
  {
    if (p!=h) prn_string(", ");
    sprintf(s, "(%u,%.4f,%.4f,%.4f,%.4f)", p->i, p->d1, p->d2, p->d3, p-
>d4);
    prn_string(s);
    p=p->seguinte;
  }
}



/* *** listas ligadas de estruturas com um inteiro sem sinal e um
double*** */


ap_no_l_uid get_uid_ui(h, i)
/*
- retorna o 1. elemento da lista com o 1. campo = i ou NULL senao
encontrar nenhum
- h e cabeca da lista ligada
*/
 ap_no_l_uid h;
 ulint i;
{
 while (h!=NULL)
 {
  if (h->i==i) return h;
  h=h->seguinte;
 }
 return NULL;
}


ap_no_l_uid insere_uid(h, ultimo, i, d, repetidos)
/*
- insere o elemento (i,d) na lista ligada
- permite ou nao elem. repetidos consoante repetidos valha 1 ou 0,
respectivamente
- h e a cabeca da lista
```

```
- ultimo e o ultimo elemento inserido na lista (ignorado se valer NULL)
- retorna um apontador para o ultimo elemento da lista
*/
 ap_no_l_uid h;
 ap_no_l_uid ultimo;
 ullint i;
 double d;
 char repetidos;
{
 ap_no_l_uid novo;

 iff (!repetidos && get_uid_ui(h,i)!=NULL)
 {
  ssprintf(s, "\"insere_uid()\": elemento (%u,%.3f) nao inserido para
eviitar elementos repetidos\n",i, d);
  prn_string(s);
  rreturn ultimo;
 }
 ellse
 {
  rnovo= (ap_no_l_uid) malloc(sizeof(no_l_uid));
  iif (novo!=NULL)
  {{
    novo->i=i;
    novo->d=d;
    novo->seguinte=NULL;
    if (ultimo!=NULL) ultimo->seguinte=novo;
    return novo;
  }
  else
   {
    sprintf(s, "\"insere_uid()\": erro de alocacao de memoria; elemento
(%u,%.3f) nao inserido\n",i,d);
    prn_string(s);
    return ultimo;
   }
 }}
}


ap_no_l_uid limpa_l_uid(h)
/**
-  elimina os elementos de uma lista ligada
-  h e a cabeca da lista
-  retorna NULL
*//
 aap_no_l_uid h;
{
 aap_no_l_uid proximo;

 wwhile (h!=NULL)
 {{
   proximo=h->seguinte;
   free((void *)h);
   h=proximo;
 }}
 rreturn h;
}
```

```
void prn_l_uid(h)
/*
- mostra no stdout o conteudo da lista ligada
*/
  ap_no_l_uid h;
{
  ap_no_l_uid p=h;

  while (p!=NULL)
  {
    if (p!=h) prn_string(", ");
    sprintf(s, "(%u,%.4f)", p->i, p->d);
    prn_string(s);
    p=p->seguinte;
  }
}


ap_no_l_uid obtem_uid_min_d(h)
/*
- retorna um apontador para o elemento cujo campo d e minimo
- retorna o primeiro elem. cujo d e igual ao minimo se houver varios
- retorna NULL se a lista for vazia
*/
  ap_no_l_uid h;
{
  double d=1e300;  /* double de valor muito elevado */
  ap_no_l_uid m=NULL;

  while (h!=NULL)
  {
    if (h->d<d) { d=h->d; m=h; }
    h=h->seguinte;
  }
  return m;
}


int remove_uid(i, h, u)
/*
- remove um elemento da lista pesquisando pelo campo i
- actualiza os apontadores para os primeiro e ultimo elementos
- retorna 0 se nao for removido nenhum elemento senao retorna 1
*/
  ulint i;
  ap_no_l_uid *h;
  ap_no_l_uid *u;
{
  ap_no_l_uid p=*h, a=NULL, aux;
  int r=0;

  while (p!=NULL)
  {
    if (p->i==i)
    {
      if (a!=NULL) a->seguinte=p->seguinte;
    else *h=p->seguinte;
      if (p==*u) *u=a;
```

```
      aux=p->seguinte;
      free(p);
      p=aux;
      r=1;
    }
    else { a=p; p=p->seguinte; }
  }
  return r;
}


/* *** listas ligadas de estruturas com um inteiro sem sinal e 3 doubles
*** */


ap_no_l_ui3d get_ui3d_ui(h, i)
/*
- retorna o 1. elemento da lista com o 1. campo = i ou NULL senao
encontrar nenhum
- h e cabeca da lista ligada
*/
 ap_no_l_ui3d h;
 uint i;
{
 while (h!=NULL)
 {
  if (h->i==i) return h;
  h=h->seguinte;
 }
 return NULL;
}


ap_no_l_ui3d insere_ui3d(h, ultimo, i, d1, d2, d3, repetidos)
/*
- insere o elemento (i,d1,d2,d3) na lista ligada
- permite ou nao elem. repetidos consoante repetidos valha 1 ou 0,
respectivamente
- h e a cabeca da lista
- ultimo e o ultimo elemento inserido na lista (ignorado se valer NULL)
- retorna um apontador para o ultimo elemento da lista
*/
 ap_no_l_ui3d h;
 ap_no_l_ui3d ultimo;
 uint i;
 double d1;
 double d2;
 double d3;
 char repetidos;
{
 ap_no_l_ui3d novo;

 if (!repetidos && get_ui3d_ui(h,i)!=NULL)
 {
  sprintf(s, "\"insere_ui3d()\": elemento (%u,%.3f,%.3f,%.3f) nao
imserido para evitar elementos repetidos\n",
     i,d1,d2,d3);
  prn_string(s);
```

```
    return ultimo;
   }
   else
   {
    novo= (ap_no_l_ui3d) malloc(sizeof(no_l_ui3d));
    if (novo!=NULL)
    {
     novo->i=i;
     novo->d1=d1;
     novo->d2=d2;
     novo->d3=d3;
     novo->seguinte=NULL;
     if (ultimo!=NULL) ultimo->seguinte=novo;
     return novo;
    }
    else
    {
     sprintf(s, "\"insere_ui3d()\": erro de alocacao de memoria; elemento
(%u,%.3f,%.3f,%.3f) nao inserido\n",
       i,d1,d2,d3);
     prn_string(s);
     return ultimo;
    }
   }
}


ap_no_l_ui3d limpa_l_ui3d(h)
/*
- elimina os elementos de uma lista ligada
- h e a cabeca da lista
- retorna NULL
*/
 ap_no_l_ui3d h;
{
 ap_no_l_ui3d proximo;

 while (h!=NULL)
 {
  proximo=h->seguinte;
  free((void *)h);
  h=proximo;
 }
 return h;
}


void prn_l_ui3d(h)
/*
- mostra no stdout o conteudo da lista ligada
*/
  ap_no_l_ui3d h;
{
  ap_no_l_ui3d p=h;

  while (p!=NULL)
  {
    if (p!=h) prn_string(", ");
    sprintf(s, "(%u,%.4f,%.4f,%.4f)", p->i, p->d1, p->d2, p->d3);
```

BIBLIOTECA F.E. PORTO

```
      prn_string(s);
      p=p->seguinte;
    }
}



/*  *** listas ligadas de estruturas com um inteiro sem sinal e 3 doubles
**** */


int: insere_int(i, primeiro, ultimo)
/*
- iinsere o elemento i no fim da lista
- pprimeiro e a cabeca da lista
- uultimo e o ultimo elemento inserido na lista (ignorado se valer NULL)
- rretorna 1 em caso de sucesso ou 0 em caso de insucesso
*/
  iint i;
  aap_no_l_int *primeiro;
  aap_no_l_int *ultimo;
{
  aap_no_l_int novo;

  nnovo= (ap_no_l_int) malloc(sizeof(no_l_int));
  iif (novo!=NULL)
  {{
     novo->i=i;
     novo->seguinte=NULL;
     if (*primeiro==NULL) *primeiro=novo;
     if (*ultimo!=NULL) (*ultimo)->seguinte=novo;
     *ultimo=novo;
     return 1;
  }}
  eelse
  {{
    sprintf(s, "\"insere_int()\": erro de alocacao de memoria; elemento
%u  nao inserido\n",i);
    prn_string(s);
    return 0;
  }}
}

intt obtem_int(indice, primeiro)
/*
- rretorna o inteiro armazenado na posicao correspondente ao indice
(1,,..,n)
- rretorna -1 se a posicao nao existir
*/
  iint indice;
  aap_no_l_int primeiro;
{
  iint i=1;

  wwhile (primeiro!=NULL && i++<indice) primeiro=primeiro->seguinte;
  iif (primeiro!=NULL) return primeiro->i;
     else return -1;
}
```

```
void limpa_l_int(h)
/*
- elimina os elementos da lista ligada
- h e a cabeca da lista, que fica a apontar para NULL no fim
*/
 ap_no_l_int *h;
{
 ap_no_l_int proximo;

 while (*h!=NULL)
 {
  proximo=(*h)->seguinte;
  free((void *) *h);
  *h=proximo;
 }
}
```

A.2.3 rand_fun.c

```
/* rand_fun.c */

#include <math.h>
#include "rand_fun.h"
#include "xu_rnd.h"


#define MAX_DOUBLE 1.7E+308


/* funcoes para geracao de numeros aleatorios */


/* variaveis globais da funcao random() */

/*double RandSeed1=123.0;
double RandSeed2=341.0;
double a1=40014.0;
double a2=40692.0;
double m1=2147483563.0;
double m2=2147483399.0;*/

double random(void)
/*
- gera um numero aleatorio entre 0 e 1.0
- os valores resultantes de sucessivas chamadas a funcao aproximam uma
fdp uniforme entre 0 e 1.0
*/
{
  /*double temp1, temp2, delta;

  temp1=a1*RandSeed1;
  temp2=a2*RandSeed2;

  RandSeed1=temp1- ( (double) ( (int) (temp1/m1) ) ) * m1;
  RandSeed2=temp2- ( (double) ( (int) (temp2/m2) ) ) * m2;
  delta=RandSeed1-RandSeed2;
  if (delta<1.0) return ((delta+m1-1.0)/m1);
```

```
    else return (delta/m1);*/

    return(xu_RndDouble());
}

double rnd_exp_neg(vm)
/*
- gera um numero aleatorio a partir de uma fdp exponencial negativa
com parametro lambeda=1/vm (inverso do valor esperado)
- se vm=0.0 retorna 0.0 e se vm<0.0 retorna MAX_DOUBLE
*/
  double vm;
{
  if (vm==0.0) return 0.0;
  else if (vm<0.0) return MAX_DOUBLE;
      else return(-vm*log(1.0-random())); /* nao consegui utilizar
xu_RndNExp() */
}


/* variaveis globais da funcao random() */

int b=0;
double y2=0.0;
double pi=3.141592654;

double rnd_normal(vm, dp)
/*
- gera um numero aleatorio a partir de uma fdp normal com valor medio vm
e desvio padrao dp
com parametro lambeda=1/vm (inverso do valor esperado)
- se dp<=0.0 retorna vm
*/
  double vm;
  double dp;
{
  double sub1, sub2;

  if (dp<=0.0) return vm;
  else if (b) { b=0; return (vm+dp*y2); }
     else /* nao consegui utilizar xu_RndGauss() */
     {
       sub1=sqrt(-2.0*log(random()));
       sub2=2.0*pi*random();
       b=1;
       y2=sub1*sin(sub2);
       return(vm+dp*sub1*cos(sub2));
     }
}

int rnd_binomial(p)
/*
- retorna 1 com probabilidade p e 0 com probabilidade 1-p
*/
  double p;
{ if (random()<p) return 1; else return 0; }
```

A.2.4 layout.c

```
/* layout.c */

/* definicao de funcoes para leitura e validacao do layout */


#include <stdio.h>
#include <ctype.h> /* isdigit() */
#include "mens_log.h"
#include "listas.h"
#include "layout.h"
#include "lefich.h"


char s[255]; /* buffer utilizado para chamar 'prn_string()' */


ap_no_l_ramos limpa_l_ramos(h)
/*
- elimina os elementos de uma lista ligada de ramos
- h e a cabeca da lista
- retorna NULL
*/
 ap_no_l_ramos h;
{
 ap_no_l_ramos proximo;

 while (h!=NULL)
 {
  limpa_l_ulint(h->bloqueios); /* limpa a lista de bloqueios do ramo */
  proximo=h->seguinte;
  free((void *)h);
  h=proximo;
 }
 return h;
}


ap_no_l_nos limpa_layout(h)
/*
- elimina os elementos de uma lista ligada de nos (layout)
- h e a cabeca da lista
- retorna NULL
*/
 ap_no_l_nos h;
{
 ap_no_l_nos proximo;

 while (h!=NULL)
 {
  limpa_l_ulint(h->ocupacao_nos); /* limpa a lista de ocupacao de nos do
no */
  limpa_l_ramos(h->ramos); /* limpa a lista de ramos do no */
  proximo=h->seguinte;
  free((void *)h);
  h=proximo;
 }
 return h;
```

```
}

ap_no_l_ulint le_ocupacao_nos(f, linha, id_no)
/*
- le a lista de ocupacao de nos a partir de um ficheiro de layout (apos
os caracteres "O:")
- actualiza a variavel linha quando le o caracter de fim de linha
- id_no e o identif. do no em questao
- retorna a cabeca da lista lida (NULL se houver algum erro)
*/
 FILE *f;
 ulint *linha;
 ulint id_no;
{
 ap_no_l_ulint h=NULL, u=NULL;
 ulint i;
 int c;

 sprintf(s, "  leitura da ocupacao de nos do no %u (linha %u)...",
id_no, *linha);
 prn_string(s);

 c=fgetc(f);
 while (isdigit(c))
 {
   i=le_ulint(f,c-'0');
   c=fgetc(f);
   if (c==',' || c=='\n')
   {
     if (h==NULL) insere_ulint(id_no, &h, &u, 0); /* insere o proprio no
na lista */
     insere_ulint(i, &h, &u, 0);
     if (c=='\n')
     {
       (*linha)++;
       prn_string("  concluida.\n");
       return h;
     }
     c=fgetc(f);
   }
 }

 sprintf(s, " caracter invalido: '%c'\n",c);
 prn_string(s);
 limpa_l_ulint(h);
 return NULL;
}


ap_no_l_ulint le_bloqueios(f, linha, origem, destino, or_final)
/*
- le a lista de bloqueios de um ramo origem->destino (apos os caracteres
"B:")
- actualiza a variavel linha quando le o caracter de fim de linha
- retorna a cabeca da lista lida (NULL se houver algum erro)
*/
 FILE *f;
 ulint *linha;
```

```
 ulint origem;
 ulint destino;
 int *or_final;
{
  ap_no_l_ulint h=NULL, u=NULL;
  char s1;
  int c;
  ulint n;

  sprintf(s, "    leitura dos bloqueios do ramo %u->%u (linha %u)...\n",
          origem, destino, *linha);
  prn_string(s);
  *or_final=0;
  do{
    if (fgetc(f)=='(')
      if (isdigit((c=fgetc(f))))
      {
    n=le_ulint(f,c-'0');
    if (fgetc(f)==',')
    {
    /* le sentido do bloqueio */
      if ((c=fgetc(f))=='-') { c=fgetc(f); s1=-1; } else s1=1;
      if (isdigit(c))
      {
        le_ulint(f,c-'0'); /* parte do principio que s1=1 ou s1=-1 */
        if (fgetc(f)==')')
        {
          c=fgetc(f);
          if (c==',' || c=='\n')
          {
           insere_ulint(n, &h, &u, 0);
           if (n==destino) *or_final=s1;       /* orientacao final do
AGV */
           if (c=='\n')
           {
             (*linha)++;
             prn_string("    concluida.\n");
             return h;
           }
          }
          else { sprintf(s, "    erro: caracter '%c' invalido\n",c);
                 prn_string(s); break; }
        }
        else { prn_string("    erro: caracter ')' nao encontrado\n");
break; }
      }
      else { prn_string("    erro: parametro sentido do bloqueio nao
encontrado\n");
           break; }
    }
    else { prn_string("    erro: caracter ',' nao encontrado apos o
identif. de no\n");
           break; }
      }
      else { prn_string("    erro: parametro identificador de no nao
encontrado\n");
           break; }
    else { prn_string("    erro: caracter '(' nao encontrado\n"); break;
}
```

```
  } while (1);
  limpa_l_ulint(h);
  return NULL;
}


ulint arredonda(f)
/*
- arredonda um float positivo ou igual a zero para o inteiro mais
proximo
*/
 float f;
{
 ulint pi_ulint;
 float pi_float;

 pi_ulint = (ulint) f;
 pi_float = (float) pi_ulint;
 if (f-pi_float<0.5) return pi_ulint; else return (pi_ulint+1);
}


ulint fcusto(dre, dcr, dma, p_aux, vre, vcr, vma)
/*
- calcula a funcao custo de um ramo do layout;
- assume-se as seguintes unidades
    distancias            : mm
    penalizacao auxiliar  : s
    velocidades           : cm/s
    custo                 : 10^-1 s
*/
 ulint dre, dcr, dma, p_aux, vre, vcr, vma;
{
 float tre, tcr, tma;

 tre= (float) dre/(float) vre;
 tcr= (float) dcr/(float) vcr;
 tma= (float) dma/(float) vma;
 return (arredonda(tre+tcr+tma)+10*p_aux);
}


ap_no_l_ramos le_ramo(f, linha, id_no, vre, vcr, vma)
/*
- le um ramo com origem em id_no (apos os caracteres "R:")
- actualiza a variavel linha quando le um caracter de fim de linha
- retorna um apontador para o novo ramo criado (NULL se houver algum
erro)
*/
  FILE *f;
  ulint *linha;
  ulint id_no;
  ulint vre, vcr, vma;
{
  no_l_ramos ramo;
  int c;
  ap_no_l_ramos novo=NULL;
  ulint recta, curva, matras, pen_auxiliar;
```

```
   sprintf(s, "   leitura na linha %u do ramo %u->", *linha, id_no);
   prn_string(s);

   if (isdigit((c=fgetc(f))))
   {
     ramo.destino=le_ulint(f,c-'0');
     sprintf(s, "%u...\n",ramo.destino);
     prn_string(s);
     if (fgetc(f)==',')
       if (isdigit((c=fgetc(f))))
       {
     recta=le_ulint(f,c-'0');
     if (fgetc(f)==',')
       if (isdigit((c=fgetc(f))))
       {
         curva=le_ulint(f,c-'0');
         if (fgetc(f)==',')
           if (isdigit((c=fgetc(f))))
           {
            matras=le_ulint(f,c-'0');
            if (fgetc(f)==',')
              if (isdigit((c=fgetc(f))))
              {
                pen_auxiliar=le_ulint(f,c-'0');
                if (fgetc(f)==',')
                {
                  if ((c=fgetc(f))=='-') c=fgetc(f);
                  if (isdigit(c))
                  {
                   le_ulint(f,c-'0'); /* le parametro pivotacao */
                   if (fgetc(f)==',')
                     if (isdigit((c=fgetc(f))))
                     {
                       ramo.tipo_ramo=le_ulint(f,c-'0');
                       if (ramo.tipo_ramo==TERMINAL ||
ramo.tipo_ramo==PASSAGEM ||
                           ramo.tipo_ramo==TERM_PASSAG)
                       {
                         procura_caracter(f,'\n',linha);
                         c=procura_prox_campo(f,linha);
                         if (c=='B')
                         {
                          if ((ramo.bloqueios=le_bloqueios(f, linha,
                               id_no, ramo.destino,
&ramo.orient_final))!=NULL)
                           {
                              novo= (ap_no_l_ramos)
malloc(sizeof(no_l_ramos));
                              if (novo!=NULL)
                              {
                                novo->destino=ramo.destino;
                                novo->orient_final=ramo.orient_final;
                                novo-
>custo=fcusto(recta,curva,matras,pen_auxiliar,
                                 vre,vcr,vma);
                                novo->pen_aux=pen_auxiliar;
/*                              novo->recta=ramo.recta;
                                novo->curva=ramo.curva;
                                novo->matras=ramo.matras;
```

```
                                    novo->pen_auxiliar=ramo.pen_auxiliar;*/
                                    novo->tipo_ramo=ramo.tipo_ramo;
                                    novo->bloqueios=ramo.bloqueios;
                                    novo->seguinte=NULL;
                                    sprintf(s, "   ramo %u->%u lido com
sucesso\n",
                                            id_no, ramo.destino);
                                     prn_string(s);
                                   }
                                   else
                                   {
                                    prn_string("   erro de alocacao de
memoria\n");
                                    limpa_l_ulint(ramo.bloqueios);
                                   }
                                 }
                                }
                                else { sprintf(s, "   erro: campo invalido na
linha %u do ficheiro de layout\n", *linha);
                                       prn_string(s);}
                               }
                               else { sprintf(s, "   erro: %u e um tipo de ramo
invalido\n", ramo.tipo_ramo);
                                       prn_string(s); }
                             }
                             else prn_string("   erro: parametro tipo_ramo nao
encontrado\n");
                           else prn_string("   erro: caracter ',' nao encontrado
entre os parametros pivot e tipo_ramo\n");
                          }
                          else prn_string("   erro: parametro pivotacao nao
encontrado\n");
                        }
                        else prn_string("   erro: caracter ',' nao encontrado
entre os parametros pen_auxiliar e pivotacao\n");
                      }
                      else prn_string("   erro: parametro pen_auxiliar nao
encontrado\n");
                   else prn_string("   erro: caracter ',' nao encontrado entre
os parametros matras e pen_auxiliar\n");
                 }
                 else prn_string("   erro: parametro matras nao encontrado\n");
               else prn_string("   erro: caracter ',' nao encontrado entre os
parametros curva e matras\n");
             }
             else prn_string("   erro: parametro curva nao encontrado\n");
           else prn_string("   erro: caracter ',' nao encontrado entre os
parametros recta e curva\n");
             }
             else prn_string("   erro: parametro recta nao encontrado\n");
           else prn_string("   erro: caracter ',' nao encontrado entre os
parametros destino e recta\n");
         }
         else prn_string("?\n   erro: parametro destino nao encontrado\n");

         return novo;
}
```

```
int le_ramos(f, linha, id_no, ramos, vre, vcr, vma)
/*
- le a lista de ramos (e respectivos bloqueios) do no id_no (apos os
caracteres "R:")
- actualiza a variavel linha quando le um caracter de fim de linha
- actualiza a variavel *ramos para ficar a apontar para a lista de ramos
criada
  (NULL se houver algum erro)
- retorna o proximo campo a ser lido no ficheiro de layout
*/
  FILE *f;
  ulint *linha;
  ulint id_no;
  ap_no_l_ramos *ramos;
  ulint vre, vcr, vma;
{
  ap_no_l_ramos ultimo, novo;
  int campo;

  *ramos=NULL;
  sprintf(s, "  leitura dos ramos do no %u\n", id_no);
  prn_string(s);
  do{
    novo=le_ramo(f, linha, id_no, vre, vcr, vma);
    if (novo!=NULL)
    {
      if (*ramos==NULL) *ramos=novo;
      else ultimo->seguinte=novo;
      ultimo=novo;
    }
    else
    {
      *ramos=limpa_l_ramos(*ramos);
      return 'R';
    }
    campo=procura_prox_campo(f,linha);
  } while (campo=='R');

  sprintf(s, "  ramos do no %u lidos com sucesso\n", id_no);
  prn_string(s);
  return campo;
}


int le_no(f, linha, novo, vre, vcr, vma)
/*
- le um no do ficheiro de layout (apos os caracteres "N:")
- actualiza a variavel linha quando le um caracter de fim de linha
- actualiza *novo para ficar a apontar para o no criado (NULL se houver
algum erro)
- retorna o proximo campo a ser lido no ficheiro de layout ('e' se
houver algum erro
  antes da leitura da ocupacao de nos e dos ramos do novo no)
*/
  FILE *f;
  ulint *linha;
  ap_no_l_nos *novo;
  ulint vre, vcr, vma;
{
```

```
no_l_nos no;
int c;

*novo=NULL;
sprintf(s, " leitura na linha %u do no ", *linha);
prn_string(s);

if (isdigit((c=fgetc(f))))
{
  no.id_no=le_ulint(f,c-'0');
  sprintf(s, "%u\n", no.id_no);
  prn_string(s);
  if (fgetc(f)==',')
    if (isdigit((c=fgetc(f))))
    {
   le_ulint(f,c-'0'); /* le parametro yi */
   if (fgetc(f)==',')
     if (isdigit((c=fgetc(f))))
     {
       le_ulint(f,c-'0'); /* le parametro xi */
       if (fgetc(f)==',')
         if (isdigit((c=fgetc(f))))
         {
          le_ulint(f,c-'0'); /* le parametro yf */
          if (fgetc(f)==',')
            if (isdigit((c=fgetc(f))))
            {
              le_ulint(f,c-'0'); /* le parametro xf */
              if (fgetc(f)==',')
                if (isdigit((c=fgetc(f))))
                {
                 no.tipo_no=le_ulint(f,c-'0');
                 if (no.tipo_no==TRANSITO || no.tipo_no==CARGA_DESCARGA
||
                        no.tipo_no==CARGA || no.tipo_no==DESCARGA)
                 {
                   procura_caracter(f, '\n', linha);
                   c=procura_prox_campo(f, linha);
                   switch (c)
                   {
                     case 'O':no.ocupacao_nos=le_ocupacao_nos(f, linha,
no.id_no);
                              c=procura_prox_campo(f, linha);
                              break;
                     case 'R': case 'N': case EOF:
                              no.ocupacao_nos=NULL;
                              insere_ulint(no.id_no,
&no.ocupacao_nos, &no.ocupacao_nos, 0);
                              break;
                     default:sprintf(s, " campo invalido na linha %u do
ficheiro de layout\n", *linha);
                              prn_string(s);
                              no.ocupacao_nos=NULL;
                   }
                   if (no.ocupacao_nos!=NULL)
                   {
                     if (c=='R') c=le_ramos(f, linha, no.id_no,
&no.ramos,
                                            vre, vcr, vma);
```

```
                        else no.ramos=NULL;
                        if (no.ramos!=NULL || c!='R')
                        {
                          *novo= (ap_no_l_nos) malloc(sizeof(no_l_nos));
                          if (*novo!=NULL)
                          {
                           (*novo)->id_no=no.id_no;
                           (*novo)->tipo_no=no.tipo_no;
                           (*novo)->pos_rel_mesa=0;
                           (*novo)->ocupacao_nos=no.ocupacao_nos;
                           (*novo)->ramos=no.ramos;
                           (*novo)->seguinte=NULL;
                           sprintf(s, " no %u lido com sucesso\n",
no.id_no);
                           prn_string(s);
                           return c;
                          }
                          else
                          {
                           sprintf(s, " erro de alocacao de memoria para
o no %u\n",no.id_no);
                           prn_string(s);
                           limpa_l_ulint(no.ocupacao_nos);
                           limpa_l_ramos(no.ramos);
                          }
                        }
                        else limpa_l_ulint(no.ocupacao_nos);
                      }
                    }
                    else {sprintf(s, " erro: %u e um tipo de no
invalido\n", no.tipo_no);
                          prn_string(s); }
                  }
                  else prn_string(" erro: parametro tipo_no nao
encontrado\n");
                else prn_string(" erro: caracter ',' nao encontrado entre
os parametros xf e tipo_no\n");
              }
            else prn_string(" erro: parametro xf nao encontrado\n");
          else prn_string(" erro: caracter ',' nao encontrado entre os
parametros yf e xf\n");
        }
        else prn_string(" erro: parametro yf nao encontrado\n");
      else prn_string(" erro: caracter ',' nao encontrado entre os
parametros xi e yf\n");
    }
    else prn_string(" erro: parametro xi nao encontrado\n");
  else prn_string(" erro: caracter ',' nao encontrado entre os
parametros yi e xi\n");
    }
    else prn_string(" erro: parametro yi nao encontrado\n");
  else prn_string(" erro: caracter ',' nao encontrado entre os
parametros id_no e yi\n");
 }
 else prn_string("?\n erro: parametro id_no nao encontrado\n");
 return 'e';
}
```

```
ap_no_l_nos le_layout(nome_ficheiro, vre, vcr, vma)
/*
- carrega um ficheiro de layout na memoria
- retorna a cabeca da lista ligada dos nos do layout (NULL se houver
algum erro)
*/
  char *nome_ficheiro;
  ulint vre, vcr, vma;
{
  FILE *f;
  ap_no_l_nos h=NULL, u=NULL, novo;
  ulint linha=1, num_nos=0;
  int c;

  sprintf(s, ": Leitura do Ficheiro de Layout '%s'\n", nome_ficheiro);
  prn_string(s);
  f=fopen(nome_ficheiro,"r");
  if (f!=NULL)
  {
    c=procura_prox_campo(f, &linha);
    do{
      switch (c)
      {
    case 'N': c=le_no(f, &linha, &novo, vre, vcr, vma);
              if (novo!=NULL)
              {
                if (h==NULL) h=novo;
                else u->seguinte=novo;
                u=novo;
                num_nos++;
              }
              else h=limpa_layout(h);
              break;
    case EOF: sprintf(s, "foi detectado o fim do ficheiro. Foram lidos
%u nos com sucesso\n", num_nos);
              prn_string(s);
              break;
    default:  sprintf(s, "erro: campo %c invalido na linha %u do
ficcheiro de layout %s\n",
                    c, linha, nome_ficheiro);
              prn_string(s);
      }
    } while (c=='N');
    fclose(f);
  }
  else { sprintf(s, "erro na abertura do ficheiro %s\n", nome_ficheiro);
           prn_string(s); }
  return h;
}


ap_no_l_ramos get_ramo(no_origem, id_no_destino)
/*
- retorna um apontador para o ramo com origem em 'no_origem' e destino
no no
cujo identificador e 'id_no_destino'
*/
  ap_no_l_nos no_origem;
  ulint id_no_destino;
```

```
{
  ap_no_l_ramos r=no_origem->ramos;

  while (r!=NULL)
  {
    if (r->destino==id_no_destino) return r;
    r=r->seguinte;
  }
  return r;
}


ap_no_l_nos get_no(layout, id)
/*
- retorna um apontador para o 1. no com identif=id ou NULL se nao
existir
*/
  ap_no_l_nos layout;
  ulint id;
{
  while (layout!=NULL)
  {
    if (layout->id_no==id) return layout;
    layout=layout->seguinte;
  }
  return NULL;
}


int unicidade_id_nos(layout)
/*
- testa se nao ha identificadores de nos repetidos
- retorna 1 se nao houver ou 0 se houver
*/
  ap_no_l_nos layout;
{
  ap_no_l_ulint h_lsr=NULL, u_lsr=NULL;
  int unicidade=1;

  prn_string("\n\nSGAGV: teste de unicidade dos identificadores de
no...\n");
  while (layout!=NULL && unicidade)
  {
    if (procura_ulint(layout->id_no, h_lsr)!=NULL) unicidade=0;
    if (unicidade) insere_ulint(layout->id_no, &h_lsr, &u_lsr, 1);
    else { sprintf(s, "erro: foram encontrados 2 nos com id=%u\n",
layout->id_no);
           prn_string(s); }
    layout=layout->seguinte;
  }
  if (unicidade) prn_string("Concluido com sucesso\n\n");
  limpa_l_ulint(h_lsr);
  return unicidade;
}


int ocup_nos_e_ramos_validos(layout)
/*
- testa a validade da ocupacao de nos e dos ramos de um layout de AGV's
```

```
- retorna 1 se forem validos ou 0 se não forem
*/
  ap_no_l_nos layout;
{
  ap_no_l_nos no;
  ap_no_l_ramos r;
  ap_no_l_ulint o, b;
  int validos=1;

  prn_string("\n\nSGAGV: teste de validade da ocupacao de nos e dos
ramos do layout...\n");
  no=layout;
  while (no!=NULL && validos)
  {
    o=no->ocupacao_nos;
    while (o!=NULL && validos)
    {
      if (get_no(layout, o->i)==NULL) validos=0;
      if (!validos)
     { sprintf(s, "\nerro: no %u invalido na lista de ocupacao de nos do
no %u\n",
            o->i, no->id_no);
      prn_string(s);
     }
      o=o->seguinte;
    }

    r=no->ramos;
    while (r!=NULL && validos)
    {
      if (r->destino==no->id_no)
      {
        validos=0;
        sprintf(s, "erro: ramo %u->%u invalido porque a origem e o
destino sao coincidentes",
           no->id_no, r->destino);
      prn_string(s);
      }
      else
      {
     if (get_no(layout, r->destino)==NULL) validos=0;
     if (validos)
     {
       if (r->orient_final==0)
       {
         validos=0;
         sprintf(s, "erro: ramo %u->%u nao bloqueia o no destino %u\n",
            no->id_no, r->destino, r->destino);
         prn_string(s);
       }
       else
       {
         b=r->bloqueios;
         while (b!=NULL && validos)
         {
           if (get_no(layout, b->i)==NULL) validos=0;
           if (!validos)
           {
```

```
            sprintf(s, "erro: no %u invalido na lista de bloqueios do
ramo %u->%u\n",
                    b->i, no->id_no, r->destino);
            prn_string(s);
           }
           b=b->seguinte;
         }
       }
     }
     else { sprintf(s, "erro: no destino %u invalido num ramo com origem
no no %u\n",
            r->destino, no->id_no);
            prn_string(s); }
       }
       r=r->seguinte;
     }
     no=no->seguinte;
   }
   if (validos) prn_string("Concluido com sucesso\n");
   prn_string("\n");
   return validos;
}


void completa_bloqueios(layout)
/*
- completa as listas de bloqueios dos ramos do layout, garantindo que
para qq ramo
   a sua lista de bloqueios contem os nos que fazem parte da lista de
ocupacao de nos
   do no destino
*/
   ap_no_l_nos layout;
{
   ap_no_l_nos no1, no2;
   ap_no_l_ramos r;

   prn_string("\n\nSGAGV: completa lista de bloqueios dos ramos do
layout...\n");
   no1=layout;
   while (no1!=NULL)
   {
     no2=layout;
     while (no2!=NULL)
     {
       if (no2!=no1)
       {
     r=no2->ramos;
     while (r!=NULL)
     {
       if (r->destino==no1->id_no)
       {
            sprintf(s, "ramo %u->%u\n", no2->id_no, r->destino);
            prn_string(s);
            append_l_ulint(&r->bloqueios, no1->ocupacao_nos, 0);
       }
       r=r->seguinte;
     }
       }
```

```
        no2=no2->seguinte;
      }
      no1=no1->seguinte;
    }
  prn_string("Concluido.\n\n");
}

void prn_ramo(r)
/*
- mostra no stdout a informacao relativa a um ramo do layout
*/
  ap_no_l_ramos r;
{
  sprintf(s, "  RAMO para o no %u\n", r->destino);
  prn_string(s);
  sprintf(s, "   custo=%u (pen_aux=%u)\n   tipo=", r->custo, r-
>pen_aux);
  prn_string(s);
  switch (r->tipo_ramo)
  {
    case TERMINAL: prn_string("terminal"); break;
    case PASSAGEM: prn_string("passagem"); break;
    default:          prn_string("terminal/passagem");
  }
  sprintf(s, ", orient_final=%d\n   bloqueios={", (int) r-
>orient_final);
  prn_string(s);
  prn_l_ulint(r->bloqueios);
  prn_string("}\n");
}


void prn_no(no)
/**
- mostra no stdout a informacao relativa a um no do layout
*//
  ap_no_l_nos no;
{
  ap_no_l_ramos r;
  sprintf(s, " NO %u\n  tipo=", no->id_no);
  prn_string(s);
  switch (no->tipo_no)
  {
    case TRANSITO: prn_string("transito"); break;
    case CARGA_DESCARGA: prn_string("carga/descarga"); break;
    case CARGA: prn_string("carga"); break;
    case DESCARGA: prn_string("descarga");
  }
  prn_string(", ocupacao de nos={");
  prn_l_ulint(no->ocupacao_nos);
  sprintf(s, "}\n  pos_rel_mesa=%d\n", (int) no->pos_rel_mesa);
  prn_string(s);
  r=no->ramos;
  while (r!=NULL)
  {
    prn_ramo(r);
    r=r->seguinte;
  }
}
```

```
void prn_layout(layout)
/*
- mostra no stdout a informacao relativa a um layout de AGV's
*/
  ap_no_l_nos layout;
{
  prn_string("\nSGAGV: LAYOUT\n");
  while (layout!=NULL)
  {
    prn_no(layout);
    layout=layout->seguinte;
  }
}


void le_f_pos_rel_pcd(nome_ficheiro, layout)
/*
- le um ficheiro de configuracao de pontos de carga/descarga,
actualizando o
  campo pos_rel_mesa dos nos do layout
*/
 char *nome_ficheiro;
 ap_no_l_nos layout;
{
  FILE *f;
  int c, pos;
  ulint id_no, linha=1;
  ap_no_l_nos no;

  sprintf(s, "\n\nSGAGV: leitura do ficheiro '%s' com a posicao relativa
das mesas...\n", nome_ficheiro);
  prn_string(s);
  f=fopen(nome_ficheiro,"r");
  if (f!=NULL)
  {
    do{
      c=procura_prox_linha(f, 'M', &linha);
      if (c!=EOF)
      {
     sprintf(s, "linha %u (mesa ", linha);
     prn_string(s);
     if (isdigit((c=fgetc(f))))
     {
       id_no=le_ulint(f, c-'0');
       sprintf(s, "%u)\n", id_no);
       prn_string(s);
       if (fgetc(f)==',')
       {
         if ((c=fgetc(f))=='-') {c=fgetc(f); pos=-1;} else pos=1;
         if (isdigit(c))
         {
           pos*=(char) le_ulint(f, c-'0');
           if (pos==-1 || pos==1)
           {
            procura_caracter(f, '\n', &linha);
            no=get_no(layout, id_no);
            if (no!=NULL)
```

```
            if (no->tipo_no!=TRANSITO) no->pos_rel_mesa=pos;
            else { sprintf(s, " erro: mesa associada ao no de TRANSITO
%u invalida; linha %u ignorada\n",
                 no->id_no, linha);
                 prn_string(s); }
          else { sprintf(s, " erro: no %u inexistente; linha
igmorada\n", id_no);
                 prn_string(s); }
        }
        else { sprintf(s, " erro: posicao %d invalida\n", (int) pos);
                 prn_string(s); break; }
      }
      else { prn_string(" erro: leitura da posicao relativa da
mesa\n"); break; }
    }
    else { prn_string(" erro: caracter ',' nao encontrado entre id_no
e pos_rel\n"); break; }
  }
  else { prn_string("??)\n erro: leitura do id_no\n"); break; }
    }
    else { sprintf(s, "Fim do ficheiro na linha %u\n", linha);
prn_string(s); }
  } while (c!=EOF);
  fclose(f);
 }
 else { sprintf(s, " erro: abertura do ficheiro %s\n", nome_ficheiro);
prn_string(s); }
}

int pos_mesas_conhecidas(layout)
/*
- verifica se todos os nos que nao sao do tipo TRANSITO conhecem a
posicao das
  respectivas mesas
*/
ap_no_l_nos layout;
{
  int todas_conh=1;

  prn_string("\n\nSGAGV: verificacao das posicoes das mesas...\n");
  while (layout!=NULL)
  {
    if (layout->tipo_no!=TRANSITO)
      if (layout->pos_rel_mesa==0)
      {
    todas_conh=0;
    sprintf(s, " erro: nao e conhecida a posicao da mesa %u em relacao
ao respectivo no\n", layout->id_no);
    prn_string(s);
      }
      else { sprintf(s, " mesa %u tem posicao %d em relacao ao
respectivo no\n", layout->id_no, (int) layout->pos_rel_mesa);
           prn_string(s); }
    layout=layout->seguinte;
  }
  prn_string("\n");
  return todas_conh;
}
```

A.2.5 mens_log.c

```
/* mens_log.c */

/* define funcoes que permitem fazer o display de mensagens no stdout e,
opcionalmente, num ficheiro de log
*/

#include "mens_log.h"
#include <stdio.h>
#include <string.h>
#include <time.h>


/* variavel estatica que guarda um apontador para o ficheiro de log */
FILE *f=NULL;

void preenche_str_data_hora(s)
/*
- preenche a string 's', com a data e hora actual
*/
  char *s;
{
  char s1[15];
  time_t t;
  struct tm *data_hora;

  t=time(NULL);
  data_hora=localtime(&t);

  switch (data_hora->tm_wday)
  {
    case 1: sprintf(s1, "Segunda-feira"); break;
    case 2: sprintf(s1, "Terca-feira"); break;
    case 3: sprintf(s1, "Quarta-feira"); break;
    case 4: sprintf(s1, "Quinta-feira"); break;
    case 5: sprintf(s1, "Sexta-feira"); break;
    case 6: sprintf(s1, "Sabado"); break;
    case 0: sprintf(s1, "Domingo");
  }
  sprintf(s, "%02d/%02d/%02d, %s, %02d:%02d:%02d", data_hora->tm_year,
    data_hora->tm_mon+1, data_hora->tm_mday, s1, data_hora->tm_hour,
    data_hora->tm_min, data_hora->tm_sec);
}

void abre_fich_log(nome, modo)
/*
- abre o ficheiro de log e escreve no cabecalho a data e hora
*/
 char *nome;
 char *modo;
{
  char s[255];

  f=fopen(nome, modo);
  if (f==NULL) printf("\n'abre_fich_log()': erro na abertura do ficheiro
de log '%s'.\n", nome);
  else
  {
```

```
    fprintf(f,
"_____________________________________________________\n");
    preenche_str_data_hora(s);
    fprintf(f, "Ficheiro aberto em %s\n\n", s);
    printf("\n'abre_fich_log()': ficheiro de log '%s' aberto com
sucesso.\n", nome);
  }
}

void prn_string(s)
/*
- imprime no stdout a string s
- se f for < > de NULL imprime-a tambem no ficheiro de log
*/
 char *s;
{
 printf("%s", s);
 if (f!=NULL) fprintf(f, "%s", s);
}

void fecha_fich_log()
/*
- fecha o ficheiro de log
*/
{
 char s[255];

 if (f!=NULL)
 {
  fprintf(f,
"\n_____________________________________________________\n");
  preenche_str_data_hora(s);
  fprintf(f, "Ficheiro fechado em %s\n\n\n\n", s);
  fclose(f);
  f=NULL;
  printf("\n'fecha_fich_log()': o ficheiro de log foi fechado.\n");
 }
}
```

A.2.6 sgm_sga.c

```
/* sgm_sga.c */

#imclude <stdio.h>
#imclude <string.h>
#imclude "mens_log.h"
#imclude "ulint.h"
#imclude "sgm_sga.h"
#imclude "sistagvs.h"


char s[255]; /* buffer utilizado para chamar 'prn_string()' */


/* funcao utilizada para imprimir no stdout o conteudo de uma mensagem
SGM<->SGAGV */
void prn_m_SGM_SGAGV(t, mens)
  double t;
```

```
  mensag_SGM_SGAGV mens;
{
  char s1[15];

  if (t>=0.0)
  {
   sprintf(s, "%10.3f:", t);
   prn_string(s);
  }
  prn_string("(id_mens,tipo_mens,p1,p2,p3,p4)=(");
  switch (mens.tipo_mens)
  {
    case ENABLE_AGV:
    sprintf(s, "%u,ENABLE_AGV,%u,", mens.id_mens, mens.p1);
    prn_string(s);
    switch (mens.p2){
      case ALOCA_AGV:
           prn_string("ALOCA_AGV"); break;
      case NAO_ALOCA_AGV:
           prn_string("NAO_ALOCA_AGV"); break;
      default:
           prn_string("???");
    }
    prn_string(",-,-)\n");
    break;
    case DISABLE_AGV:
    sprintf(s, "%u,DISABLE_AGV,%u,-,-,-)\n", mens.id_mens, mens.p1);
    prn_string(s);
    break;
    case TAREFA_NORMAL:
    if (mens.p1==QUALQUER)
      sprintf(s, "%u,TAREFA_NORMAL,QUALQUER,%u,%u,%u)\n", mens.id_mens,
           mens.p2, mens.p3, mens.p4);
    else sprintf(s, "%u,TAREFA_NORMAL,%u,%u,%u,-)\n", mens.id_mens,
           mens.p1, mens.p2, mens.p3);
    prn_string(s);
    break;
    case TAREFA_POSICION:
    sprintf(s, "%u,TAREFA_POSICION,%u,%u,-,-)\n", mens.id_mens,
           mens.p1, mens.p2);
    prn_string(s);
    break;
    case TAREFA_CARGA:
    if (mens.p1==QUALQUER)
      sprintf(s, "%u,TAREFA_CARGA,QUALQUER,%u,-,%u)\n", mens.id_mens,
           mens.p2, mens.p4);
    else sprintf(s, "%u,TAREFA_CARGA,%u,%u,-,-)\n", mens.id_mens,
           mens.p1, mens.p2);
    prn_string(s);
    break;
    case TAREFA_DESCARGA:
    sprintf(s, "%u,TAREFA_DESCARGA,%u,%u,-,-)\n", mens.id_mens,
           mens.p1, mens.p2);
    prn_string(s);
    break;
    case CANCELA_TAREFA:
    sprintf(s, "%u,CANCELA_TAREFA,", mens.id_mens);
    prn_string(s);
    switch (mens.p1)
```

```
{
  case UMA_TAREFA:
       sprintf(s, "UMA_TAREFA,%u,-,-)\n", mens.p2); break;
  case TODAS_TAR_EXEC:
       sprintf(s, "TODAS_TAR_EXEC,-,-,-)\n"); break;
  case TODAS_TAR:
       sprintf(s, "TODAS_TAR,-,-,-)\n"); break;
  default:
       sprintf(s, "???,%u,-,-)\n", mens.p2); break;
}
prn_string(s);
break;
case CONFIRMACAO:
sprintf(s, "%u,CONFIRMACAO,", mens.id_mens);
prn_string(s);
switch (mens.p1)
{
  case SUCESSO:
       prn_string("SUCESSO,-,-,-)\n"); break;
  case INSUCESSO:
       prn_string("INSUCESSO,");
       switch (mens.p2)
       {
         case AGV_INEX:
              prn_string("AGV_INEX"); break;
         case AGV_INDISPON:
              prn_string("AGV_INDISPON"); break;
         case AGV_CARR:
              prn_string("AGV_CARR"); break;
         case AGV_DESCARR:
              prn_string("AGV_DESCARR"); break;
         case TAREFA_INEX:
              prn_string("TAREFA_INEX"); break;
         case ORIGEM_INVAL:
              prn_string("ORIGEM_INVAL"); break;
         case DESTINO_INVAL:
              prn_string("DESTINO_INVAL"); break;
         case NO_INVALIDO:
              prn_string("NO_INVALIDO"); break;
         case CAMINHO1_INEX:
              prn_string("CAMINHO1_INEX"); break;
         case CAMINHO2_INEX:
              prn_string("CAMINHO2_INEX"); break;
         case OP_CAR_IMP:
              prn_string("OP_CAR_IMP"); break;
         case OP_DES_IMP:
              prn_string("OP_DES_IMP"); break;
         default:
              prn_string("???");
       }
       prn_string(",-,-)\n");
       break;
  default:
       sprintf(s, "???,%u,-,-)\n", mens.p2);
       prn_string(s);
       break;
}
break;
case TAREFA_CANCEL:
```

```
      escr_str_est_tarefa(s1, mens.p1);
      sprintf(s, "%u,TAREFA_CANCEL,%s,%u,", mens.id_mens, s1, mens.p2);
      prn_string(s);
      switch (mens.p3)
      {
        case ORDEM_SGM:
             prn_string("ORDEM_SGM"); break;
        case AGV_DESACTIVADO:
             prn_string("AGV_DESACTIVADO"); break;
        case DEADLOCK:
             prn_string("DEADLOCK"); break;
        default:
             prn_string("???");
      }
      prn_string(",-)\n");
      break;
      case ESTADO_AGV:
      escr_str_est_movim_agv(s1, mens.p2);
      sprintf(s, "%u,ESTADO_AGV,%u,%s,", mens.id_mens, mens.p1, s1);
      prn_string(s);
      escr_str_est_aloc_agv(s1, mens.p2, mens.p3);
      sprintf(s, "%s,", s1);
      prn_string(s);
      escr_str_est_carga_agv(s1, mens.p2, mens.p4);
      sprintf(s, "%s)\n", s1);
      prn_string(s);
      break;
      case ESTADO_TAREFA:
      escr_str_est_tarefa(s1, mens.p2);
      sprintf(s, "%u,ESTADO_TAREFA,%u,%s,%u,-)\n", mens.id_mens, mens.p1,
s1, mens.p3);
      prn_string(s);
      break;
      case T_POSIC_ESPEC:
      sprintf(s, "%u,T_POSIC_ESPEC,%u,%u,%u,-)\n", mens.id_mens, mens.p1,
mens.p2, mens.p3);
      prn_string(s);
      break;
     default:
      sprintf(s, "%u,???,%u,%u,%u,%u)\n", mens.id_mens, mens.p1, mens.p2,
mens.p3, mens.p4);
      prn_string(s);
   }
}
```

A.2.7 sga_spa.c

```
/* sga_spa.c */

#include <stdio.h>
#include <string.h>
#include "mens_log.h"
#include "ulint.h"
#include "sga_spa.h"
#include "sistagvs.h"


char s[255]; /* buffer utilizado para chamar 'prn_string()' */
```

```
/* funcao utilizada para imprimir no stdout o conteudo de uma mensagem
SGAGV<->SPAGV */
void prn_m_SGAGV_SPAGV(t, mens)
  double t;
  mensag_SGAGV_SPAGV mens;
{
  char s1[15];

  if (t>=0.0)
  {
   sprintf(s, "%10.3f:", t);
   prn_string(s);
  }
  prn_string("(id_agv,tipo_mens,p1,p2)=(");
  switch (mens.tipo_mens){
    case MOVIMENTO:
    sprintf(s, "%u,MOVIMENTO,", mens.id_agv);
    prn_string(s);
    switch (mens.p1){
      case MOV_TRANSL:
           prn_string("MOV_TRANSL");
           break;
      case MOV_CARGA:
           prn_string("MOV_CARGA");
           break;
      case MOV_DESCARGA:
           prn_string("MOV_DESCARGA");
           break;
      default:
           prn_string("???");
    }
    sprintf(s, ",%u)\n", mens.p2);
    prn_string(s);
    break;
    case INF_EST_AGV:
    escr_str_est_movim_agv(s1, mens.p1);
    sprintf(s, "%u,INF_EST_AGV,%s,-)\n", mens.id_agv, s1);
    prn_string(s);
    break;
    case ESTADO_INTR:
    sprintf(s, "%u,ESTADO_INTR,", mens.id_agv);
    prn_string(s);
    switch (mens.p1){
      case AGV_INTROD:
           prn_string("AGV_INTROD");
           break;
      case AGV_NOT_ENABLED:
           prn_string("AGV_NOT_ENABLED");
           break;
      case NO_INEX:
           prn_string("NO_INEX");
           break;
      case NO_OCUPADO:
           prn_string("NO_OCUPADO");
           break;
      default:
           prn_string("???");
```

```
        }
        prn_string(",-)\n");
        break;
        case CANCEL_MOV_AVAN:
        sprintf(s, "%u,CANCEL_MOV_AVAN,-,-)\n", mens.id_agv);
        prn_string(s);
        break;
        case FIM_MOVIMENTO:
        sprintf(s, "%u,FIM_MOVIMENTO,-,-)\n", mens.id_agv);
        prn_string(s);
        break;
        case INTRODUCAO:
        sprintf(s, "%u,INTRODUCAO,%u,%d)\n", mens.id_agv, mens.p1, mens.p2);
        prn_string(s);
        break;
        case ERRO_AGV:
        sprintf(s, "%u,ERRO_AGV,-,-)\n", mens.id_agv);
        prn_string(s);
        break;
        case FIM_ERRO_AGV:
        sprintf(s, "%u,FIM_ERRO_AGV,-,-)\n", mens.id_agv);
        prn_string(s);
        break;
        case AGV_MANUAL:
        sprintf(s, "%u,AGV_MANUAL,-,-)\n", mens.id_agv);
        prn_string(s);
        break;
        case MOV_AVAN_CANCEL:
        sprintf(s, "%u,MOV_AVAN_CANCEL,-,-)\n", mens.id_agv);
        prn_string(s);
        break;
        default:
        sprintf(s, "%u,???,%u,%u)\n", mens.id_agv, mens.p1, mens.p2);
        prn_string(s);
    }
}
```

A.2.8 sistagvs.c

```
/* sistagvs.c */

#include <stdio.h>
#include <string.h>
#include "ulint.h"
#include "sistagvs.h"



/* definicao de funcoes utilizaveis em todo o sistema */

void escr_str_est_movim_agv(s, est_movim)
/*
- preenche uma string com o estado de movimentacao de um AGV
*/
  char *s;
  ulint est_movim;
{
  switch (est_movim){
```

```
    case DISABLED:sprintf(s,"DISABLED"); break;
    case ENABLED: sprintf(s,"ENABLED"); break;
    case PARADO:  sprintf(s,"PARADO"); break;
    case EX_MOVIM:sprintf(s,"EX_MOVIM"); break;
    case CARGA_AGV:      sprintf(s,"CARGA_AGV"); break;
    case DESCARGA_AGV:sprintf(s,"DESCARGA_AGV"); break;
    case BLOQUEADO:      sprintf(s,"BLOQUEADO"); break;
    case ERRO:    sprintf(s,"ERRO"); break;
    default:      sprintf(s,"???");
    }
}

void escr_str_est_aloc_agv(s, est_movim, est_aloc)
/*
- preenche uma string com o estado de alocacao de um AGV
*/
  char *s;
  ulint est_movim;
  ulint est_aloc;
{
  if (est_movim!=DISABLED)
    switch (est_aloc)
    {
      case NAO_ALOCADO:   sprintf(s,"NAO_ALOCADO"); break;
      case ALOCADO:       sprintf(s,"ALOCADO"); break;
      default:            sprintf(s,"???");
    }
  else sprintf(s,"-");
}

void escr_str_est_carga_agv(s, est_movim, est_carga)
/*
- preenche uma string com o estado de carga de um AGV
*/
  char *s;
  ulint est_movim;
  ulint est_carga;
{
  if (est_movim!=DISABLED && est_movim!=ENABLED)
    switch (est_carga)
    {
      case DESCARREGADO:sprintf(s,"DESCARREGADO"); break;
      case CARREGADO:    sprintf(s,"CARREGADO"); break;
      default:           sprintf(s,"???");
    }
  else sprintf(s,"-");
}

void escr_str_tipo_tarefa(s, tipo_tarefa)
/*
- preenche uma string com o tipo de uma Tarefa
*/
  char *s;
  ulint tipo_tarefa;
{
  switch (tipo_tarefa)
  {
    case NORMAL:   sprintf(s,"NORMAL"); break;
    case NORMAL_AGV:      sprintf(s,"NORMAL_AGV"); break;
```

```
    case POSICION: sprintf(s,"POSICION"); break;
    case T_CARGA:  sprintf(s,"T_CARGA"); break;
    case CARGA_AGV:       sprintf(s,"CARGA_AGV"); break;
    case T_DESCARGA:      sprintf(s,"T_DESCARGA"); break;
    case POSIC_ESPEC:     sprintf(s,"POSIC_ESPEC"); break;
    default:              sprintf(s,"???");
  }
}

void escr_str_est_tarefa(s, est_tarefa)
/*
- preenche uma string com o estado de uma Tarefa
*/
  char *s;
  ulint est_tarefa;
{
  switch (est_tarefa)
  {
    case FILA:            sprintf(s,"FILA"); break;
    case PASSO1:   sprintf(s,"PASSO1"); break;
    case PASSO2:   sprintf(s,"PASSO2"); break;
    case PASSO3:   sprintf(s,"PASSO3"); break;
    case PASSO4:   sprintf(s,"PASSO4"); break;
    case CONCLUIDA:       sprintf(s,"CONCLUIDA"); break;
    case REJEITADA:       sprintf(s,"REJEITADA"); break;
    default:              sprintf(s,"???");
  }
}
```

A.2.9 sgm.c

```
/* sgm.c */

#include <stdio.h>
#include <string.h>
#include <ctype.h> /* isdigit() */
#include "mens_log.h"
#include "listas.h"
#include "sgm_sga.h"
#include "sgm.h"
#include "lefich.h"
#include "fun_aux.h"
#include "sistagvs.h"
#include "rand_fun.h"


char s[255]; /* buffer utilizado para chamar 'prn_string()' */


/* definicao das funcoes para leitura e validacao
do ficheiro de configuracao do SGM */


ap_no_l_uid le_f_nos_tarefas(nome_ficheiro)
/*
- le um ficheiro de configuracao de nos e respectivas probabilidades de
  ocorrencia de posicionamentos
```

```
- retorna uma lista em que cada elemento contem a probabilidade do
respectivo no
  participar num posicionamento
- significado dos campos da estrutura:
  i  : identificador do no
  d : p_posic
*/
  char *nome_ficheiro;
{
  FILE *f;
  int c;
  ulint id_no, linha=1;
  double p_intr, p_posic;
  ap_no_l_uid h=NULL, u=NULL;

  sprintf(s, "\n\nSGM: leitura do ficheiro '%s' c/ probabilidades dos
nos...\n",
     nome_ficheiro);
  prn_string(s);
  f=fopen(nome_ficheiro,"r");
  if (f!=NULL)
  {
    do{
      c=procura_prox_linha(f, 'N', &linha);
      if (c!=EOF)
      {
     sprintf(s, "linha %u (no ", linha);
     prn_string(s);
     if (isdigit((c=fgetc(f))))
     {
       id_no=le_ulint(f, c-'0');
       sprintf(s, "%u)\n", id_no);
       prn_string(s);
       if (fgetc(f)==',')
       {
         le_double(f, &p_intr); /* p_introd (ignorado) */
           if (fgetc(f)==',')
           {
            le_double(f, &p_posic);
              if ((c=fgetc(f))=='\n')
              {
                linha++;
                u=insere_uid(h, u, id_no, p_posic, 0);
                if (h==NULL) h=u;
              }
              else {sprintf(s, "erro: caracter '%c' invalido\n",c);
                    prn_string(s); break;}
           }
           else {prn_string("erro: caracter ',' nao encontrado entre
  p_introd e p_posic\n");break;}
       }
       else {prn_string("erro: caracter ',' nao encontrado entre id_no e
  p_introd\n");break;}
     }
     else { prn_string("??)\n erro: leitura do id_no\n"); break; }
       }
       else { sprintf(s, " Fim do ficheiro na linha %u\n", linha);
              prn_string(s); }
     } while (c!=EOF);
```

```
    fclose(f);
  }
  else { sprintf(s, " erro: abertura do ficheiro %s\n", nome_ficheiro);
       prn_string(s); }
  if (c!=EOF) h=limpa_1_uid(h);
  return h;
}


int som_prob_nos_unit(nos_tar)
/*
- verifica se o somatorio das probabilidades dos nos das tarefas e 1.0
  para as cargas, descargas e posicionamentos
- se for retorna 1 senao retorna 0
*/
  ap_no_1_uid nos_tar;
{
  double s_pos=0.0;

  prn_string("\n\nSGM: verifica somatorio das probab. dos nos de
tarefas...\n");
  while (nos_tar!=NULL && s_pos<(1.0+TOLER_IGUAL_DOUBLES) )
  {
    if (nos_tar->d>=0.0 && nos_tar->d<=1.0) s_pos+=nos_tar->d;
    else
    {
      sprintf(s, " erro: no %u tem uma probab. de posicionamento
invalida =%.4f\n",
    nos_tar->i,nos_tar->d);
    prn_string(s);
      s_pos=2.0;
    }

    nos_tar=nos_tar->seguinte;
  }
  prn_string("Verificacao concluida.\n");
  if (!d1iguald2(s_pos,1.0,TOLER_IGUAL_DOUBLES))
    prn_string("Erro nas probabilidades de op. de posicionamento\n");
  if (d1iguald2(s_pos,1.0,TOLER_IGUAL_DOUBLES))
    return 1; else return 0;
}


/* definicao de funcoes para manipulacao das listas de AGV's */

int insere_AGV(agv, p, u)
/*
- insere um novo elemento no fim da lista ligada
- actualiza os apontadores *p (primeiro) e *u (ultimo)
- retorna 1 quando bem sucedida ou 0 se ocorrer algum erro
*/
  ap_AGV agv;
  ap_AGV *p, *u;
{
  ap_AGV novo;

  novo= (ap_AGV) malloc(sizeof(AGV));
  if (novo!=NULL)
  {
```

```
      novo->id_agv=agv->id_agv;
      novo->estado_movim=agv->estado_movim;
      novo->estado_alocacao=agv->estado_alocacao;
      novo->estado_carga=agv->estado_carga;
      novo->tarefa_agv=agv->tarefa_agv;
      novo->t_enable=agv->t_enable;
      novo->t_disable=agv->t_disable;
      novo->t_mud_est_aloc=agv->t_mud_est_aloc;
      novo->seguinte=NULL;
      novo->anterior=*u;
      if (*u!=NULL) (*u)->seguinte=novo;
    . *u=novo;
      if (*p==NULL) *p=novo;
      return 1;
    }
    else prn_string("\"insere_AGV()\": erro de alocacao de memoria\n");
    return 0;
}}

iint remove_AGV(agv, p, u)
//*
-- remove um elemento da lista de AGV's
-- actualiza *p e *u quando isso for necessario
-- retorna 1 quando e bem sucedida ou 0 no caso contrario
**/
    ap_AGV agv;
    ap_AGV *p, *u;
{{
    if (agv!=NULL)
    {
      if (agv->anterior!=NULL) agv->anterior->seguinte=agv->seguinte;
      else *p=agv->seguinte;
      if (agv->seguinte!=NULL) agv->seguinte->anterior=agv->anterior;
      else *u=agv->anterior;
      free(agv);
      return 1;
    }
    else return 0;

//*  ap_AGV e=*p, ant=NULL, aux;
    int r=0;

    while (e!=NULL)
    {
      if (e->id_agv==id)
      {
        if (ant!=NULL) ant->seguinte=e->seguinte;
      else *p=e->seguinte;
        if (e->seguinte==NULL) *u=ant;
        aux=e->seguinte;
        free(e);
        e=aux;
        r=1;
      }
      else
      {
        ant=e;
        e=e->seguinte;
      }
```

```
  }
  return r;*/
}

ap_AGV obtem_AGV(id, p)
/*
- retorna o 1. elemento da lista com id_agv==id, percorrendo-a no
sentido
cabeca->cauda
- retorna NULL se nao for encontrado nenhum elemento
- p e cabeca da lista ligada
*/
 ulint id;
 ap_AGV p;
{
 while (p!=NULL && p->id_agv!=id) p=p->seguinte;
 return p;
}

void limpa_l_AGVs(p)
/*
- destroi uma lista, comecando pela cabeca
*/
  ap_AGV p;
{
  ap_AGV prox;

  while (p!=NULL)
  {
    prox=p->seguinte;
    free((void *) p);
    p=prox;
  }
}

void prn_agv(agv)
/*
- imprime a informacao relativa a um AGV
*/
  ap_AGV agv;
{
  char s1[15];

  if (agv!=NULL)
  {
    sprintf(s, "AGV %u\tid_tarefa=", agv->id_agv);
    prn_string(s);

    if (agv->tarefa_agv!=NULL) sprintf(s, "%u", agv->tarefa_agv-
>id_tar);
     else sprintf(s, "---");
    prn_string(s);

    escr_str_est_movim_agv(s1, agv->estado_movim);
    sprintf(s, "\n estado=(%s,",s1);
    prn_string(s);

    escr_str_est_aloc_agv(s1, agv->estado_movim, agv->estado_alocacao);
    sprintf(s, "%s,",s1);
```

```
      prn_string(s);

      escr_str_est_carga_agv(s1, agv->estado_movim, agv->estado_carga);
      sprintf(s, "%s)\n t_enable=%.2f, t_disable=%.2f,
t_mmud_est_aloc=%.2f\n",
      s1, agv->t_enable, agv->t_disable, agv->t_mud_est_aloc);
      prn_string(s);
      sprintf(s, "ult_destino=%u\n", agv->ult_destino);
      prn_string(s);
   }}
   eelse prn_string("(NULL)\n");
}

voiid prn_l_AGVs(p)
/*
- iimprime o conteudo de uma lista de AGV's, percorrendo-a no sentido
cabbeca->cauda
*/
   aap_AGV p;
{
   wwhile (p!=NULL)
   {{
      prn_agv(p);
      p=p->seguinte;
   }}
}


/*  definicao de funcoes para manipulacao das listas de Tarefas */

intt insere_Tarefa(tarefa, p, u)
/*
- iinsere um novo elemento no fim da lista ligada
- aactualiza os apontadores *p (primeiro) e *u (ultimo)
- rretorna 1 quando bem sucedida ou 0 se ocorrer algum erro
*/
   aap_Tarefa tarefa;
   aap_Tarefa *p, *u;
{
   aap_Tarefa novo;

   nnovo= (ap_Tarefa) malloc(sizeof(Tarefa));
   iif (novo!=NULL)
   {{
      novo->id_tar=tarefa->id_tar;
      novo->tipo_tar=tarefa->tipo_tar;
      novo->prior=tarefa->prior;
      novo->no1=tarefa->no1;
      novo->no2=tarefa->no2;
      novo->estado_tar=tarefa->estado_tar;
      novo->agv_tarefa=tarefa->agv_tarefa;
      novo->seguinte=NULL;
      novo->anterior=*u;
      if (*u!=NULL) (*u)->seguinte=novo;
      *u=novo;
      if (*p==NULL) *p=novo;
      return 1;
   }}
   eelse prn_string("\"insere_Tarefa()\": erro de alocacao de memoria\n");
```

```
  return 0;
}

int remove_Tarefa(tarefa, p, u)
/*
- remove um elemento da lista de Tarefas
- actualiza *p e *u quando isso for necessario
- retorna 1 quando e bem sucedida ou 0 no caso contrario
*/
  ap_Tarefa tarefa;
  ap_Tarefa *p, *u;
{
  if (tarefa!=NULL)
  {
    if (tarefa->anterior!=NULL) tarefa->anterior->seguinte=tarefa-
>seguinte;
    else *p=tarefa->seguinte;
    if (tarefa->seguinte!=NULL) tarefa->seguinte->anterior=tarefa-
>anterior;
    else *u=tarefa->anterior;
    free(tarefa);
    return 1;
  }
  else return 0;

/*  ap_Tarefa e=*p, ant=NULL, aux;
  int r=0;

  while (e!=NULL)
  {
    if (e->id_tar==id)
    {
      if (ant!=NULL) ant->seguinte=e->seguinte;
    else *p=e->seguinte;
      if (e->seguinte==NULL) *u=ant;
      aux=e->seguinte;
      free(e);
      e=aux;
      r=1;
    }
    else
    {
      ant=e;
      e=e->seguinte;
    }
  }
  return r;*/
}

ap_Tarefa obtem_Tarefa(id, p)
/*
- retorna o 1. elemento da lista de Tarefas com id_tar==id, percorrendo-
a no sentido
cabeca->cauda
- retorna NULL se nao for encontrado nenhum elemento
- p e cabeca da lista ligada
*/
 ulint id;
 ap_Tarefa p;
```

```
{
 while (p!=NULL && p->id_tar!=id) p=p->seguinte;
 return p;
}

void limpa_l_Tarefas(p)
/*
- destroi uma lista, comecando pela cabeca
*/
  apo_Tarefa p;
{
  apo_Tarefa prox;

  while (p!=NULL)
  {
    prox=p->seguinte;
    free((void *) p);
    p=prox;
  }
}

void prn_Tarefa(tar)
/*
- imprime a informacao relativa a uma Tarefa
*/
  apo_Tarefa tar;
{
  char s1[15];

  if (tar!=NULL)
  {
    sprintf(s, "Tarefa %u\tid_agv=", tar->id_tar);
    prn_string(s);

    if (tar->agv_tarefa!=NULL) sprintf(s, "%u", tar->agv_tarefa-
>id_agv);
      else sprintf(s, "-");
    prn_string(s);
    escr_str_tipo_tarefa(s1, tar->tipo_tar);
    sprintf(s, "\n tipo_tar=%s, prior=%u, ", s1, tar->prior);
    prn_string(s);

      switch (tar->tipo_tar)
      {
        case NORMAL: case NORMAL_AGV:
      sprintf(s, "origem=%u, destino=%u", tar->no1, tar->no2); break;
        case POSICION: case T_CARGA: case CARGA_AGV: case POSIC_ESPEC:
      sprintf(s, "ponto=%u", tar->no1); break;
        case T_DESCARGA: sprintf(s, "ponto=%u", tar->no2);
      }
      prn_string(s);

      escr_str_est_tarefa(s1, tar->estado_tar);
      sprintf(s, "\n estado_tar=%s\n", s1);
      prn_string(s);
   }
   else prn_string("(NULL)\n");
 }
```

```
void prn_l_Tarefas(p)
/*
- imprime o conteudo de uma lista de Tarefas, percorrendo-a no sentido
cabeca->cauda
*/
  ap_Tarefa p;
{
  while (p!=NULL)
  {
    prn_Tarefa(p);
    p=p->seguinte;
  }
}


/* definicao de outras funcoes utilizadas na RP do SGM */

/*void proc_estado_agv(t, m, p_agvs)*/
/*
- processa uma mensagem do tipo ESTADO_AGV, alterando o estado do
respectivo
AGV
*/
/*  double t;
  mensag_SGM_SGAGV *m;
  ap_AGV p_agvs;
{
  p_agvs=obtem_AGV(m->p1, p_agvs);
  if (p_agvs!=NULL)
  {
    p_agvs->estado_movim=m->p2;
    p_agvs->estado_alocacao=m->p3;
    p_agvs->estado_carga=m->p4;
    sprintf(s, "\nSGM: alteracao do estado do AGV %u com sucesso apos a
recepcao de:\n", m->p1);
  }
  else sprintf(s, "\nSGM: AGV %u inexistente apos a recepcao de:\n", m-
>p1);
  prn_string(s);
  prn_m_SGM_SGAGV(t, *m);
}*/

void proc_estado_tarefa(t, m, tar, p_agvs)
/*
- processa uma mensagem do tipo ESTADO_TAREFA, alterando o estado da
respectiva
tarefa e o AGV afectado a mesma
*/
  double t;
  mensag_SGM_SGAGV *m;
  ap_Tarefa tar;
  ap_AGV p_agvs;
{
  char s1[12];

  tar->estado_tar=m->p2;

  if (tar->agv_tarefa!=NULL && tar->agv_tarefa->id_agv!=m->p3 &&
    tar->agv_tarefa->tarefa_agv==tar)
```

```
    tar->agv_tarefa->tarefa_agv=NULL;

  if (tar->agv_tarefa==NULL || tar->agv_tarefa->id_agv!=m->p3)
  {
    tar->agv_tarefa=obtem_AGV(m->p3, p_agvs);
    if (tar->agv_tarefa!=NULL) tar->agv_tarefa->tarefa_agv=tar;
  }
  escr_str_est_tarefa(s1, tar->estado_tar);
  sprintf(s, "\nSGM: Tarefa %u passa ao estado %s, apos a recepcao
de:\m", m->p1, s1);
  prm_string(s);
  prm_m_SGM_SGAGV(t, *m);
}

ap_Tarefa proc_t_posic_espec(t, m, p_tars, u_tars, p_agvs)
/*
- processa uma mensagem do tipo T_POSIC_ESPEC, inserindo a nova tarefa
na fila
*/
  double t;
  mensag_SGM_SGAGV *m;
  ap_Tarefa *p_tars;
  ap_Tarefa *u_tars;
  ap_AGV p_agvs;
{
  Tarefa tar;

  tar.id_tar=m->p1;
  tar.tipo_tar=POSIC_ESPEC;
  tar.prior=0;
  tar.no1=m->p3;
  tar.estado_tar=PASSO1;
  tar.agv_tarefa=obtem_AGV(m->p2, p_agvs);

  if (insere_Tarefa(&tar, p_tars, u_tars))
  {
    if (tar.agv_tarefa!=NULL) tar.agv_tarefa->tarefa_agv=*u_tars;
    sprintf(s, "\nSGM: insercao da tarefa %u na fila apos a recepcao
de:\rn", m->p1);
    prm_string(s);
    prm_m_SGM_SGAGV(t, *m);
    return (*u_tars);
  }
  else
  {
    sprintf(s, "\nSGM: erro na insercao da tarefa %u na fila apos a
recepcao de:\n", m->p1);
    prm_string(s);
    prm_m_SGM_SGAGV(t, *m);
    return NULL;
  }
}

double calc_carga(n_tars, n_agvs_activos)
/*
- determina o valor da carga do sistema
- a carga do sistema e definida como sendo o quociente entre o numero de
tarefas
```

```
pendentes e o numero de AGV's num estado diferente de {DISABLED,
ENABLED},
admitindo um denominador diferente de zero
- se o denominador for igual a zero, e retornado o valor MAX_DOUBLE
(valor muito elevado)
*/
  ulint n_tars;
  ulint n_agvs_activos;
{
  if (n_agvs_activos>0) return( ((double) n_tars)/((double)
n_agvs_activos) );
    else return(MAX_DOUBLE);
}

int existe_AGV_nova_tarefa(p_agvs, p_esc_sgm)
/*
- retorna 1 se houver 1 AGV passivel de ser afectado a uma nova tarefa,
ou 0
se for decidido que o escalonamento nao deve ser feito ao nivel do SGM
ou se nao
for encontrado nenhum AGV disponivel (PARADO/ALOCADO ou
PARADO/CARREGADO)
*/
  ap_AGV p_agvs;
  double p_esc_sgm;
{
  if (rnd_binomial(p_esc_sgm))
  {
    while (p_agvs!=NULL && (p_agvs->estado_movim!=PARADO ||
    p_agvs->tarefa_agv!=NULL ||
    (p_agvs->estado_carga!=CARREGADO && p_agvs-
>estado_alocacao!=ALOCADO)))
          p_agvs=p_agvs->seguinte;

    if (p_agvs!=NULL) return 1;
  }
  return 0;
}

ap_AGV deter_AGV_nova_tarefa(p_agvs)
/*
- retorna o AGV a afectar a uma nova tarefa
*/
  ap_AGV p_agvs;
{
  ap_AGV p_agv_aloc=NULL; /* apontador para um AGV PARADO/ALOCADO */

  while (p_agvs!=NULL &&
    (p_agvs->estado_movim!=PARADO || p_agvs->tarefa_agv!=NULL ||
          p_agvs->estado_carga!=CARREGADO))
  {
    if (p_agvs->estado_movim==PARADO && p_agvs->tarefa_agv==NULL &&
    p_agvs->estado_alocacao==ALOCADO)
    p_agv_aloc=p_agvs;
    p_agvs=p_agvs->seguinte;
  }
  if (p_agvs!=NULL) return p_agvs;
    else return p_agv_aloc;
}
```

```
ulint deter_tipo_tarefa(agv_tarefa, p_tar_carga, p_posicion,
p_tar_carga_agv)
/*
- determina o tipo de uma nova tarefa, tomando como base se a tarefa e
escalonada
pelo SGM ou nao, e as probabilidades definidas para o efeito
*/
  ap_AGV agv_tarefa;
  double p_tar_carga;
  double p_posicion;
  double p_tar_carga_agv;
{
  if (agv_tarefa==NULL)
  {{
     if (rnd_binomial(p_tar_carga)) return T_CARGA;
     else return NORMAL;
  }}
  else
  {{
     if (rnd_binomial(p_posicion)) return POSICION;
     else if (agv_tarefa->estado_carga==CARREGADO) return T_DESCARGA;
      else if (rnd_binomial(p_tar_carga_agv)) return CARGA_AGV;
            else return NORMAL_AGV;
  }}
}

ulint deter_prior_tar(p_tar_prior, prior_tar_max)
/**
- determina a prioridade de uma nova tarefa do tipo NORMAL ou T_CARGA
- se a tarefa for considerada prioritaria (com base na probabilidade
p_tar_prior)
a prioridade valera entre 1 e prior_tar_max, senao valera 0
*//
  double p_tar_prior;
  ulint prior_tar_max;
{
  if (rnd_binomial(p_tar_prior))
     return ( ( (ulint) (random()*((double) prior_tar_max)) )+1);
  else return 0;
}

ulint decide_no_tarefa(nos_tarefas)
/**
- determina um no do layout associado a uma tarefa de Posicionamento com
base na probabilidade associada a cada no para onde possa ocorrer o mov.
*//
  ap_no_l_uid nos_tarefas;
{
  double som_p_ant=0.0, p=random(), p_no;
  ap_no_l_uid anterior=NULL;

  while (nos_tarefas!=NULL)
  {
    p_no=nos_tarefas->d;
    if ((som_p_ant+p_no)>=p && p_no>TOLER_IGUAL_DOUBLES)
    return nos_tarefas->i;
    som_p_ant+=p_no;
    if (p_no>TOLER_IGUAL_DOUBLES) anterior=nos_tarefas;
```

```
    nos_tarefas=nos_tarefas->seguinte;
  }

  if (anterior!=NULL) return anterior->i;
    else return 0;
}

ulint decide_tipo_cancel_tar(p_1_tar, p_todas_exec)
/*
- decide o tipo de cancelamento de tarefas a efectuar com base nas
probabilidades
correspondentes a UMA_TAREFA, TODAS_TAR_EXEC e TODAS_TAR
*/
  double p_1_tar;
  double p_todas_exec;
{
  double p=random();

  if (p<p_1_tar) return UMA_TAREFA;
    else if (p<(p_1_tar+p_todas_exec)) return TODAS_TAR_EXEC;
      else return TODAS_TAR;
}

ulint decide_tar_a_cancelar(p_tars, n_tars)
/*
- escolhe aleatoriamente uma tarefa a cancelar dentre as primeiras
n_tars
- retorna 0 se a tarefa escolhida nao for encontrada
*/
  ap_Tarefa p_tars;
  ulint n_tars;
{
  ulint i=( (ulint) random()*((double) n_tars) )+1;
  ulint j=1;

  while (p_tars!=NULL && j++<i) p_tars=p_tars->seguinte;

  if (p_tars!=NULL) return p_tars->id_tar; else return 0;
}
```

A.2.10 dettraj.c

```
/* dettraj.c */

#include <stdio.h>
#include <string.h>
#include "mens_log.h"
#include "listas.h"
#include "layout.h"
#include "sgagv.h"



char s[255]; /* buffer utilizado para chamar 'prn_string()' */


/* definicao das funcoes para manipulacao de listas de apont. para nos
do layout */
```

```
ap_no_l_apnodtraj procura_apnodtraj(no, h)
/*
- retorna o 1. elemento encontrado ou NULL se nao for encontrado nenhum
- h e cabeca da lista ligada
*/
 ap_No no;
 ap_no_l_apnodtraj h;
{
 while (h!=NULL && h->no!=no) h=h->seguinte;
 return h;
}


int insere_apnodtraj(no, h, u, repetidos)
/*
- insere um novo elemento no fim da lista ligada
- permite ou nao repetidos consoante o valor passado em 'repetidos'
- actualiza os apontadores h e u
- retorna 1 quando bem sucedida ou 0 se ocorrer algum erro
*/
  ap_No no;
  ap_no_l_apnodtraj *h, *u;
  char repetidos;
{
  ap_no_l_apnodtraj novo;

  if (repetidos || procura_apnodtraj(no, *h)==NULL)
  {
    novo= (ap_no_l_apnodtraj) malloc(sizeof(no_l_apnodtraj));
    if (novo!=NULL)
    {
      novo->no=no;
      novo->seguinte=NULL;
      novo->anterior=*u;
      if (*u!=NULL) (*u)->seguinte=novo;
      *u=novo;
      if (*h==NULL) *h=novo;
      return 1;
    }
    else prn_string("\"insere_apnodtraj()\": erro de alocacao de
memoria\n");
  }
  /*else printf("\"insere_apnodtraj()\": elemento repetido nao
inserido\n");*/
  return 0;
}


int remove_apnodtraj(no, h, u)
/*
- remove da lista o elemento no
- actualiza *h e *u quando isso for necessario
- retorna 1 quando e bem sucedida ou 0 se ocorrer um erro
*/
  ap_No no;
  ap_no_l_apnodtraj *h, *u;
{
```

```
  ap_no_l_apnodtraj e;

  if ((e=procura_apnodtraj(no, *h))!=NULL)
  {
    if (e!=*h) e->anterior->seguinte=e->seguinte;
    else *h=e->seguinte;
    if (e!=*u) e->seguinte->anterior=e->anterior;
    else *u=e->anterior;
    free((void *) e);
    return 1;
  }
  else return 0;
}


void limpa_l_apnodtraj(h)
/*
- destroi uma lista, comecando pela cabeca
*/
  ap_no_l_apnodtraj h;
{
  ap_no_l_apnodtraj prox;

  while (h!=NULL)
  {
    prox=h->seguinte;
    free((void *) h);
    h=prox;
  }
}



/* definicao das funcoes para manipulacao de listas representando
trajectos */


int insere_a_cabeca_l_apno(no, custo, h)
/*
- insere `a cabeca um novo elemento na lista ligada, actualizando *h
- retorna 1 em caso de sucesso, ou 0 em caso de falha
*/
  ap_no_l_nos no;
  ulint custo;
  ap_no_l_apno *h;
{
  ap_no_l_apno novo=(ap_no_l_apno) malloc(sizeof(no_l_apno));

  if (novo!=NULL)
  {
    novo->no=no;
    novo->custo=custo;
    novo->anterior=NULL;
    novo->seguinte=*h;
    if (*h!=NULL) (*h)->anterior=novo;
    *h=novo;
    return 1;
  }
```

```
  else
  {{
     prn_string("\"insere_a_cabeca_l_apno()\": erro de alocacao de
memoria\n");
     return 0;
  }}
}


voiid remove_cabeca_l_apno(h)
/*
- remove o 1. elemento da lista, se existir
- actualiza o apontador *h, colocando-o a apontar para a nova cabeca da
lissta
*/
  aap_no_l_apno *h;
{
  aap_no_l_apno p;

  iif (*h!=NULL)
  {{
     p=(*h)->seguinte;
     free(*h);
     *h=p;
     if (*h!=NULL) (*h)->anterior=NULL;
  }}
}


voiid limpa_l_apno(h)
/*
- llimpa toda a lista
*/
  aap_no_l_apno *h;
{ while (*h!=NULL) remove_cabeca_l_apno(h); }


voiid prn_l_apno(h)
/*
- iimprime os elementos da lista
*/
  aap_no_l_apno h;
{
  iif (h==NULL) prn_string("(NULL)");
  while (h!=NULL)
   {
     if (h->anterior!=NULL) prn_string(", ");
     sprintf(s, "%u(%u)", h->no->id_no, h->custo);
     prn_string(s);
     h=h->seguinte;
   }
  prn_string("\n");
}



/* definicao das funcoes para determinacao do trajecto optimo entre 2
nos */
```

```
ap_No obtem_no_lay_dtraj(id, h)
/*
- retorna um apontador para o elemento cujo campo id_no==id
- se nao existir, retorna NULL
*/
  ulint id;
  ap_No h;
{
  while (h!=NULL)
  {
    if (h->id_no==id) return h;
    h=h->seguinte;
  }
  return h;
}


void limpa_ramos_no_dtraj(h)
/*
- limpa a lista de ramos de um no do layout utilizado durante a
determinacao do traj.
*/
  ap_Ramo h;
{
  ap_Ramo prox_ramo;

  while (h!=NULL)
  {
    prox_ramo=h->seguinte;
    free(h);
    h=prox_ramo;
  }
}


void limpa_lay_dtraj(h)
/*
- limpa a lista que representa o layout durante a determinacao do
trajecto
*/
  ap_No h;
{
  ap_No prox_no;

  while (h!=NULL)
  {
    limpa_ramos_no_dtraj(h->ramos);
    prox_no=h->seguinte;
    free(h);
    h=prox_no;
  }
}


void elimina_nos_isolados(h, origem)
/*
- elimina nos isolados do layout utilizado para determinar o trajecto
```

```
- considera-se que um no esta isolado se nao houver ramos para esse no,
comm a
restricao de nao se tratar do no origem
*/
  aap_No *h;
  aap_no_l_nos origem;
{
  aap_No u_no=NULL, no1=*h, no2, aux;
  aap_Ramo r_no1, r;

  while (no1!=NULL)
     if (no1->id_no!=origem->id_no)
     {
       no2=*h; r_no1=NULL;
       while (no2!=NULL && r_no1==NULL)
       {
     if (no2!=no1)
     {
       r=no2->ramos;
       while (r!=NULL && r_no1==NULL)
       {
         if (r->cabeca==no1) r_no1=r;
         r=r->seguinte;
       }
     }
     no2=no2->seguinte;
       }

       if (r_no1==NULL)
       {
     if (u_no!=NULL) u_no->seguinte=no1->seguinte;
            else *h=no1->seguinte;
     limpa_ramos_no_dtraj(no1->ramos);
     aux=no1->seguinte;
     free(no1);
     no1=aux;
       }
       else
       {
     u_no=no1;
     no1=no1->seguinte;
       }
     }
     else
     {
       u_no=no1;
       no1=no1->seguinte;
     }
}


ap_No cria_lay_dtraj(layout, origem, destino, nos_a_ignorar,
     ramo_a_ignorar, poss_acesso)
/*
- cria uma lista que representa de forma eficiente o layout, durante o
processo de
determinacao do trajecto optimo
*/
  ap_no_l_nos layout;
```

```
  ap_no_l_nos origem;
  ap_no_l_nos destino;
  ap_no_l_ulint nos_a_ignorar;
  ap_no_l_ramos ramo_a_ignorar;
  int poss_acesso; /* possibilidade de acesso por parte dos AGV's */
{
  ap_no_l_nos no_lay;
  ap_No no, h=NULL, u=NULL;
  ap_no_l_ramos r_lay;
  ap_Ramo u_r, r;
  int noerrors=1, int_vazio;

  no_lay=layout;

  while (no_lay!=NULL && noerrors)
  {
    if (procura_ulint(no_lay->id_no, nos_a_ignorar)==NULL)
    {
      no=(ap_No) malloc(sizeof(No));
      if (no!=NULL)
      {
        no->id_no=no_lay->id_no;
        if (no_lay==origem) no->w=0; else no->w=MAXINT;
        no->pn=NULL;
        no->ramos=NULL;
        no->seguinte=NULL;

        if (h==NULL) h=no;
        if (u!=NULL) u->seguinte=no;
        u=no;
      }
      else
      {
        prn_string("\"cria_lay_dtraj()\": erro na alocacao de um novo
no\n");
        noerrors=0;
      }
    }
    no_lay=no_lay->seguinte;
  }

  no=h; no_lay=layout;
  while (no!=NULL && noerrors)
  {
    while (no_lay->id_no!=no->id_no) no_lay=no_lay->seguinte;
    r_lay=no_lay->ramos; u_r=NULL;
    while (r_lay!=NULL && noerrors)
    {
    if (r_lay!=ramo_a_ignorar)
    {
      int_vazio=inters_igual_vazio(nos_a_ignorar, r_lay->bloqueios);
        if ( int_vazio &&
          ((r_lay->destino!=origem->id_no && r_lay->destino!=destino-
>id_no &&
            r_lay->tipo_ramo!=TERMINAL) ||
           (r_lay->destino==destino->id_no &&
              (r_lay->tipo_ramo!=PASSAGEM || destino->tipo_no==TRANSITO)
&&
              (destino->tipo_no==TRANSITO || poss_acesso==0 ||
```

```
                destino->pos_rel_mesa*r_lay->orient_final==poss_acesso)
            )
          )
        )
          {
        r=(ap_Ramo) malloc(sizeof(Ramo));
        if (r!=NULL)
        {
          r->cabeca=obtem_no_lay_dtraj(r_lay->destino, h);

          r->custo=r_lay->custo;
          r->seguinte=NULL;
          if (no->ramos==NULL) no->ramos=r;
          if (u_r!=NULL) u_r->seguinte=r;
          u_r=r;
        }
        else
        {
          prn_string("\"cria_lay_dtraj()\": erro na alocacao de memoria de
um novo ramo\n");
          noerrors=0;
        }
          }
      }
        r_lay=r_lay->seguinte;
      }
      no=no->seguinte;
  }}

  if (!noerrors) { limpa_lay_dtraj(h); h=NULL; }
     else elimina_nos_isolados(&h, origem);
  return h;
}


void general_labeling_algorithm(nos, origem)
/**
- implementa o algoritmo de Dijkstra para redes ciclicas e aciclicas,
calculando os
parametros w e pn para todos os nos atingiveis desde a origem
*//
  ap_No nos;
  ap_No origem;
{
  ap_no_l_apnodtraj S_h=NULL, S_u=NULL;
  ap_No no;
  ap_Ramo r;

  insere_apnodtraj(origem, &S_h, &S_u, 0);
  do{
    no=S_h->no;
    remove_apnodtraj(no, &S_h, &S_u);
    r=no->ramos;
    while (r!=NULL)
    {
      if ( (no->w + r->custo) < r->cabeca->w )
      {
    r->cabeca->w = no->w + r->custo;
    r->cabeca->pn = no;
```

```
      insere_apnodtraj(r->cabeca, &S_h, &S_u, 0);
        }
        r=r->seguinte;
      }
    } while (S_h!=NULL);
}


ap_no_l_apno obtem_trajecto(destino, layout)
/*
- retorna uma lista representativa de um trajecto resultante da execucao
do GLA,
percorrendo o layout a partir do no destino, atraves do campo pn
- retorna NULL se ocorrer algum erro
*/
  ap_No destino;
  ap_no_l_nos layout;
{
  ap_no_l_apno h=NULL;
  int noerror=1;

  while (destino->pn!=NULL && noerror)
  {
    noerror=insere_a_cabeca_l_apno(get_no(layout, destino->id_no),
    destino->w-destino->pn->w, &h);
    destino=destino->pn;
  }

  if (!noerror) limpa_l_apno(&h);

  return h;
}


ulint traj_optimo(layout, origem, destino,
    nos_a_ignorar, ramo_a_ignorar, poss_acesso, traj_opt)
/*
- determina o trajecto optimo entre 2 pontos e o custo associado
- retorna o custo do trajecto optimo e um apontador para a cabeca
da lista que o representa atrves de '*traj_optimo'
*/
  ap_no_l_nos layout;
  ap_no_l_nos origem;
  ap_no_l_nos destino;
  ap_no_l_ulint nos_a_ignorar;
  ap_no_l_ramos ramo_a_ignorar;
  int poss_acesso;
  ap_no_l_apno *traj_opt;
{
  ap_No lay_dtraj, or_dtraj, de_dtraj;
  ulint c=MAXINT;

  *traj_opt=NULL;

  lay_dtraj=cria_lay_dtraj(layout, origem, destino,
    nos_a_ignorar, ramo_a_ignorar, poss_acesso);
  if ((or_dtraj=obtem_no_lay_dtraj(origem->id_no, lay_dtraj))!=NULL &&
      (de_dtraj=obtem_no_lay_dtraj(destino->id_no, lay_dtraj))!=NULL)
  {
```

```
      general_labeling_algorithm(lay_dtraj, or_dtraj);
      *traj_opt=obtem_trajecto(de_dtraj, layout);
      c=de_dtraj->w;
    }
/**  else prn_string("\"traj_optimo()\": no origem ou no destino nao
encontrado\n");*/

  limpa_lay_dtraj(lay_dtraj);
  return c;
}


ullint custo_traj_optimo(layout, origem, destino,
     nos_a_ignorar, ramo_a_ignorar, poss_acesso)
/**
- determina o custo associado ao trajecto optimo entre 2 pontos
-  retorna MAXINT se ocorrer algum erro
*//
   ap_no_l_nos layout;
   ap_no_l_nos origem;
   ap_no_l_nos destino;
   ap_no_l_ulint nos_a_ignorar;
   ap_no_l_ramos ramo_a_ignorar;
   int poss_acesso;
{
   ap_No lay_dtraj, or_dtraj, de_dtraj;
   ulint c=MAXINT;

/**  prn_string("\n'custo_traj_optimo': vou formar estrutura do
laayout\n");*/
   lay_dtraj=cria_lay_dtraj(layout, origem, destino,
     nos_a_ignorar, ramo_a_ignorar, poss_acesso);
   if ((or_dtraj=obtem_no_lay_dtraj(origem->id_no, lay_dtraj))!=NULL &&
       (de_dtraj=obtem_no_lay_dtraj(destino->id_no, lay_dtraj))!=NULL)
   {
/**   prn_string("\n'custo_traj_optimo': vou chamar
'ggeneral_labeling_algorithm'\n");*/
     general_labeling_algorithm(lay_dtraj, or_dtraj);
     c=de_dtraj->w;
   }
/**  else prn_string("\"custo_traj_optimo()\": no origem ou no destino
naao encontrado\n");*/

   limpa_lay_dtraj(lay_dtraj);
   return c;
}
```

A.2..11 esc_desp.c

```
/** sgagv.c */

#iinclude <stdio.h>
#iinclude <string.h>
#iinclude "mens_log.h"
#iinclude "listas.h"
#iinclude "layout.h"
#iinclude "fun_aux.h"
#iinclude "sgagv.h"
#iinclude "sistagvs.h"
```

```
char s[255]; /* buffer utilizado para chamar 'prn_string()' */


/* funcoes para manipulacao de listas de apontadores para AGV's */


int insere_no_l_ap_AGVs(agv, h, t)
/*
- insere um novo apontador na cauda da lista, actualizando *h e *t
- retorna 1 em caso de sucesso ou 0 em caso de erro
*/
  ap_AGV agv;
  ap_no_l_ap_AGVs *h;
  ap_no_l_ap_AGVs *t;
{
  ap_no_l_ap_AGVs novo;

  novo=(ap_no_l_ap_AGVs) malloc(sizeof(no_l_ap_AGVs));
  if (novo!=NULL)
  {
    novo->agv=agv;
    novo->seguinte=NULL;
    if (*h==NULL) *h=novo;
    if (*t!=NULL) (*t)->seguinte=novo;
    *t=novo;
    return 1;
  }
  else
  {
    prn_string("\"insere_no_l_ap_AGVs()\": erro de alocacao de
memoria\n");
    return 0;
  }
}

void remove_cab_l_ap_AGVs(h)
/*
- remove o 1. elemento da lista e actualiza *h
*/
  ap_no_l_ap_AGVs *h;
{
  ap_no_l_ap_AGVs p;

  if (*h!=NULL)
  {
    p=(*h)->seguinte;
    free(*h);
    *h=p;
  }
}

void limpa_l_ap_AGVs(h)
/*
- limpa toda a lista, colocando *h=NULL
*/
  ap_no_l_ap_AGVs *h;
```

```
{
  ap_no_l_ap_AGVs p;

  while (*h!=NULL)
  {
    p=(*h)->seguinte;
    free(*h);
    *h=p;
  }
}


ap_AGV obtem_ap_agv_pos(h, pos)
/*
- retorna o campo agv do elemento da lista colocado na posicao pos
- se nao existir, retorna NULL
*/
  ap_no_l_ap_AGVs h;
  ulint pos;
{
  ulint i=1;

  while (h!=NULL && i++ < pos) h=h->seguinte;

  if (h!=NULL) return h->agv; else return NULL;
}

void prn_l_ap_AGVs(h)
/*
- imprime no stdout os identificadores das tarefas apontadas pelos
elelementos
daa lista ligada
*/
  ap_no_l_ap_AGVs h;
{
  ap_no_l_ap_AGVs p=h;

  prn_string("{");
  while (p!=NULL)
  {
    if (p!=h)
    prn_string(",");
    if (p->agv!=NULL) sprintf(s, "%u", p->agv->id_agv);
    else sprintf(s, "(NULL)");
    prn_string(s);
    p=p->seguinte;
  }
  prn_string("}\n");
}


/* funcoes para manipulacao de listas de apontadores para Tarefas */


imt existe_no_l_ap_Tarefas(tarefa, h)
/*
- retorna 1 se existir um elemento a apontar para tarefa ou 0 no caso
contrario
```

```
*/
  ap_Tarefa tarefa;
  ap_no_l_ap_Tarefas h;
{
  while (h!=NULL)
  {
    if (h->tarefa==tarefa) return 1;
    h=h->seguinte;
  }
  return 0;
}

int insere_no_l_ap_Tarefas(tarefa, h, t)
/*
- insere um novo apontador na cauda da lista, actualizando *h e *t
- retorna 1 em caso de sucesso ou 0 em caso de erro
*/
  ap_Tarefa tarefa;
  ap_no_l_ap_Tarefas *h;
  ap_no_l_ap_Tarefas *t;
{
  ap_no_l_ap_Tarefas novo;

  novo=(ap_no_l_ap_Tarefas) malloc(sizeof(no_l_ap_Tarefas));
  if (novo!=NULL)
  {
    novo->tarefa=tarefa;
    novo->seguinte=NULL;
    if (*h==NULL) *h=novo;
    if (*t!=NULL) (*t)->seguinte=novo;
    *t=novo;
  4 return 1;
  }
  else
  {
    prn_string("\"insere_no_l_ap_Tarefas()\": erro de alocacao de
memoria\n");
    return 0;
  }
}

void remove_cab_l_ap_Tarefas(h)
/*
- remove o 1. elemento da lista e actualiza *h
*/
  ap_no_l_ap_Tarefas *h;
{
  ap_no_l_ap_Tarefas p;

  if (*h!=NULL)
  {
    p=(*h)->seguinte;
    free(*h);
    *h=p;
  }
}

void limpa_l_ap_Tarefas(h)
/*
```

```
- limpa toda a lista, colocando *h=NULL
*/
  ap_no_l_ap_Tarefas *h;
{
  ap_no_l_ap_Tarefas p;

  while (*h!=NULL)
  {
    p=(*h)->seguinte;
    free(*h);
    *h=p;
  }
}


ap_Tarefa obtem_ap_tarefa_pos(h, pos)
/*
- retorna o campo tarefa do elemento da lista colocado na posicao pos
- se nao existir, retorna NULL
*/
  ap_no_l_ap_Tarefas h;
  ulint pos;
{
  ulint i=1;

  while (h!=NULL && i++ < pos) h=h->seguinte;

  if (h!=NULL) return h->tarefa; else return NULL;
}

void prn_l_ap_Tarefas(h)
/*
- imprime no stdout os identificadores das tarefas apontadas pelos
elementos
da lista ligada
*/
  ap_no_l_ap_Tarefas h;
{
  ap_no_l_ap_Tarefas p=h;

  prn_string("{");
  while (p!=NULL)
  {
    if (p!=h)
    prn_string(",");
    if (p->tarefa!=NULL) sprintf(s, "%u", p->tarefa->id_tar);
    else sprintf(s, "(NULL)");
    prn_string(s);
    p=p->seguinte;
  }
  prn_string("}\n");
}


/* funcoes para manipular a matriz de eficiencia */


uint* aloca_matriz_quad(dim)
```

```
/*
- aloca espaco de memoria para guardar uma matriz dim x dim (dim>0)
- retorna NULL se ocorrer um erro
*/
  ulint dim;
{ return( (ulint *) malloc(sizeof(ulint)*dim*dim) ); }

void escr_elem(m, dim, l, c, i)
/*
- escreve o valor i na posicao (l,c) da matriz ( 1<=l<=dim e 1<=c<=dim )
*/
  ulint *m;
  ulint dim;
  ulint l;
  ulint c;
  ulint i;
{ *(m+(l-1)*dim+c-1)=i; }

ulint le_elem(m, dim, l, c)
/*
- retorna o elemento (l,c) da matriz ( 1<=l<=dim e 1<=c<=dim )
*/
  ulint *m;
  ulint dim;
  ulint l;
  ulint c;
{ return( *(m+(l-1)*dim+c-1) ); }

void prn_matriz(m, dim)
/*
- imprime a matriz no stdout
*/
  ulint *m;
  ulint dim;
{
  ulint l, c;

  prn_string("\n");
  for (l=1; l<=dim; l++)
  {
    for (c=1; c<=dim; c++)
    {
    sprintf(s, " %10u", le_elem(m, dim, l, c));
    prn_string(s);
    }
    prn_string("\n");
  }
}

void destroi_matriz(m)
/*
- liberta o espaco de memoria ocupado pela matriz
*/
  ulint *m;
{ free(m); }


/* definicao das funcoes de escalonamento e de despacho */
```

```
ulint deter_num_af(tarefas, agvs, num_max_af, reaf_dinam)
/**
-- determina o numero de afectacoes a efectuar pelo escalonador
**/
   ap_Tarefa tarefas;
   ap_AGV agvs;
   ulint num_max_af;
   int reaf_dinam;
{{
   ulint num_tar=0, num_agvs=0;

   while (tarefas!=NULL)
   {
     if (
      (tarefas->tipo_tar==NORMAL || tarefas->tipo_tar==T_CARGA) &&
      (tarefas->estado_tar==FILA || (tarefas->estado_tar==PASSO1 &&
            tarefas->agv_tarefa->estado_alocacao==NAO_ALOCADO &&
reaf_dinam)
      )
        ) num_tar++;
     tarefas=tarefas->seguinte;
   }

   while (agvs!=NULL)
   {
     if (
      agvs->estado_alocacao==NAO_ALOCADO &&
      agvs->estado_carga==DESCARREGADO &&
      ((agvs->estado_movim==PARADO && agvs->tarefa_agv==NULL) ||
       (reaf_dinam && agvs->tarefa_agv!=NULL && agvs->tarefa_agv-
>estado_tar==PASSO1 &&
        (agvs->tarefa_agv->tipo_tar==NORMAL || agvs->tarefa_agv-
>tipo_tar==T_CARGA)
       )
         )
        ) num_agvs++;
     agvs=agvs->seguinte;
   }

   return min_ulint(num_max_af, max_ulint(num_tar, num_agvs));
//*    return max_ulint(num_agvs, min_ulint(num_max_af, num_tar));*/
}}

iint forma_l_tar_e_agvs_af(num_af, tarefas, agvs, reaf_dinam,
     tar_af, num_tar_af, agvs_af, num_agvs_af)
//*
-- forma uma lista de apontadores para tarefas e uma lista de AGV's
iintervenientes no processo de afectacao
-- retorna 0 em caso de erro ou 1 em caso de sucesso
**/
   ulint num_af;
   ap_Tarefa tarefas;
   ap_AGV agvs;
   int reaf_dinam;
   ap_no_l_ap_Tarefas *tar_af;
   ulint *num_tar_af;
   ap_no_l_ap_AGVs *agvs_af;
```

```
  ulint *num_agvs_af;
{
  ap_Tarefa tar, tar_sel;
  ap_no_l_ap_Tarefas u_tar_af=NULL, h_tar_sel=NULL, u_tar_sel=NULL;
  ap_no_l_ap_AGVs u_agv_af=NULL;
  ulint pmax;
  char noerrors=1;
  ulint num_tar_fila=0;

  *tar_af=NULL;
  *num_tar_af=0;
  *agvs_af=NULL;
  *num_agvs_af=0;

  /* insere na lista de tarefas as tarefas no passo 1 do tipo NORMAL ou
T_CARGA,
     e insere na lista de AGV's os respectivos AGV's */
  tar=tarefas;
  while (tar!=NULL && *num_tar_af<num_af && *num_agvs_af<num_af &&
noerrors)
  {
    if (reaf_dinam && tar->estado_tar==PASSO1 &&
    (tar->tipo_tar==NORMAL || tar->tipo_tar==T_CARGA) &&
    tar->agv_tarefa->estado_alocacao==NAO_ALOCADO
       )
    {
      if (insere_no_l_ap_Tarefas(tar, tar_af, &u_tar_af))
      {
    tar->est_antes_desp=PASSO1;
    (*num_tar_af)++;
      }
    else noerrors=0;
      if (insere_no_l_ap_AGVs(tar->agv_tarefa, agvs_af, &u_agv_af))
(*num_agvs_af)++;
    else noerrors=0;
    }
    if (tar->estado_tar==FILA && (tar->tipo_tar==NORMAL || tar-
>tipo_tar==T_CARGA))
    num_tar_fila++; /* em principio, basta verificar a primeira
condicao,
                    porque so tarefas do tipo NORMAL ou T_CARGA podem
estar
                    no estado FILA */
    tar=tar->seguinte;
  }

  /* se ainda nao tiver sido atingido o limite num_af e se nao tiver
ocorrido
  qualquer erro, num_tar_fila vale o numero de tarefas em fila de espera
*/

  /* insere tarefas no estado FILA na lista de tarefas ate perfazer
num_af ou
     ate esgotar as tarefas, escolhendo as tarefas pela sua prioridade */
  while (*num_tar_af<num_af && num_tar_fila>0 && noerrors)
  {
    tar=tarefas; pmax=0; tar_sel=NULL;
    while (tar!=NULL)
    {
```

```
        if (tar->estado_tar==FILA &&
        (tar->prior>pmax || (tar->prior==pmax && tar_sel==NULL)) &&
        !existe_no_l_ap_Tarefas(tar, h_tar_sel))
        { tar_sel=tar; pmax=tar->prior; }
        tar=tar->seguinte;
      }

      if (insere_no_l_ap_Tarefas(tar_sel, &h_tar_sel, &u_tar_sel) &&
      insere_no_l_ap_Tarefas(tar_sel, tar_af, &u_tar_af))
      {
        tar_sel->est_antes_desp=FILA;
        (*num_tar_af)++;
        num_tar_fila--;
      }
      else noerrors=0;
    }
    limpa_l_ap_Tarefas(&h_tar_sel);

    /* insere agvs no estado PARADO e NAO_ALOCADO na lista de AGV's ate
  peerfazer
       num_af ou ate esgotar os AGV's */
    while (agvs!=NULL && *num_agvs_af<num_af && noerrors)
    {
      if (agvs->estado_movim==PARADO && agvs->tarefa_agv==NULL &&
      agvs->estado_alocacao==NAO_ALOCADO &&
      agvs->estado_carga==DESCARREGADO)
      if (insere_no_l_ap_AGVs(agvs, agvs_af, &u_agv_af)) (*num_agvs_af)++;
      else noerrors=0;
      agvs=agvs->seguinte;
    }

    if (noerrors) return 1;
    else
    {
      limpa_l_ap_Tarefas(tar_af);
      *num_tar_af=0;
      limpa_l_ap_AGVs(agvs_af);
      *num_agvs_af=0;
      return 0;
    }
  }

  ullint* forma_matr_efic(num_af, tarefas, agvs, layout, poss_acesso)
  /**
  - calcula a matriz de eficiencia para o Algoritmo Hungaro
  *//
    ulint num_af;
    ap_no_l_ap_Tarefas tarefas;
    ap_no_l_ap_AGVs agvs;
    ap_no_l_nos layout;
    int poss_acesso;
  {
    ulint *m; /* matriz quadrada num_af * num_af elementos */
    ulint ia; /* indice de AGV */
    ulint it; /* indice de Tarefa */
    ap_no_l_ap_Tarefas t;
    ap_no_l_nos pos;
```

```
/*  prn_string("\n'forma_matr_efic': vou alocar memoria para a
matriz\n");*/
  if ((m=aloca_matriz_quad(num_af))!=NULL)
    for (ia=1; ia<=num_af; ia++)
    {
      t=tarefas;
      for (it=1; it<=num_af; it++)
      {
    if (agvs!=NULL)
      if (t!=NULL)
      {
/*      if (agvs->agv->tarefa_agv!=NULL && agvs->agv-
>estado_movim=MOVIMENTO)
          pos=agvs->agv->tarefa_agv->traj1->no;
        else */pos=agvs->agv->posicao;
        if (pos!=t->tarefa->no1)
        {
/*          sprintf(s, "'forma_matr_efic': vou calcular custo (AGV %u,
Tarefa %u), %u->%u\n",
                  agvs->agv->id_agv, t->tarefa->id_tar, pos->id_no, t-
>tarefa->no1->id_no);
          prn_string(s);*/
          escr_elem(m, num_af, ia, it, custo_traj_optimo(layout, pos,
                                t->tarefa->no1, NULL, NULL,
poss_acesso));
        }
        else if (poss_acesso==0 ||
                agvs->agv->orient*pos->pos_rel_mesa==poss_acesso)
                  escr_elem(m, num_af, ia, it, 0);
              else escr_elem(m, num_af, ia, it, MAXIlT);
      }
      else escr_elem(m, num_af, ia, it, MAXINT);
    else
      if (t!=NULL) escr_elem(m, num_af, ia, it, MAXINT);
      else escr_elem(m, num_af, ia, it, 0);

    if (t!=NULL) t=t->seguinte;
      }
      if (agvs!=NULL) agvs=agvs->seguinte;
    }
  return m;
}

void alg_hung_p1(m, dim)
/* Passo 1 do Algoritmo Hungaro
- determina o minimo de cada linha da matriz e subtrai-o a todos os
elementos da linha correspondente
*/
  ulint *m;
  ulint dim;
{
  ulint l, c, minimo;

  for (l=1; l<=dim; l++)
  {
    minimo=MAXINT;
    for (c=1; c<=dim; c++)
      if (le_elem(m, dim, l, c)<minimo) minimo=le_elem(m, dim, l, c);
```

```
    for (c=1; c<=dim; c++) escr_elem(m, dim, l, c, le_elem(m, dim, l,
c)-minimo);
  }}
}

intt forma_zl_zc_lzeros(m, dim, zl, zc, lzeros)
/*
- fforma uma lista zl com dim elementos, representando o numero de zeros
da
mattriz m em cada uma das suas dim linhas, e uma lista zc com dim
eleemento s,
reppresentando o numero de zeros da matriz m em cada uma das suas dim
collunas
- llzeros guarda a localizacao dos zeros encontrados na matriz
- rretorna 1 em caso de sucesso ou 0 em caso de insucesso
*/
  uulint *m;
  uulint dim;
  eap_no_l_ulint *zl;
  eap_no_l_ulint *zc;
  eap_no_l_2uint *lzeros;
{
  uulint l, c;
  eap_no_l_ulint u_zl=NULL, u_zc=NULL, p_zc;
  eap_no_l_2uint u_lzeros=NULL;
  iint noerrors=1;

  **zl=NULL;
  **zc=NULL;
  **lzeros=NULL;

  ffor (c=1; (c<=dim && noerrors); c++) noerrors=insere_ulint(0, zc,
&u_zc, 1);

  ffor (l=1; (l<=dim && noerrors); l++)
  {{
     noerrors=insere_ulint(0, zl, &u_zl, 1);
     p_zc=*zc;
     for (c=1; (c<=dim && noerrors); c++)
     {
       if (le_elem(m, dim, l, c)==0)
       {
     (u_zl->i)++;
     (p_zc->i)++;
     noerrors=insere_2uint(l, c, lzeros, &u_lzeros);
       }
       p_zc=p_zc->seguinte;
     }
  }}
  rreturn noerrors;
}

voiid act_zl(m, dim, c, zl)
/*
- aactualiza a lista zl, que guarda o numero de zeros em cada linha de m,
quaando os zeros de uma coluna c sao eliminados
- ppara cada um dos zeros dessa coluna, e decrementado o numero de zeros
da respectiva linha
*/
```

```
  ulint *m;
  ulint dim;
  ulint c;
  ap_no_l_ulint zl;
{
  ulint l;

  for (l=1; l<=dim; l++)
  {
    if (le_elem(m, dim, l, c)==0 && zl->i > 0) (zl->i)--;
    zl=zl->seguinte;
  }
}

void act_zc(m, dim, l, zc)
/*
- actualiza a lista zc, que guarda o numero de zeros em cada coluna de
m,
quando os zeros de uma linha l sao eliminados
- para cada um dos zeros dessa linha, e decrementado o numero de zeros
da respectiva coluna
*/
  ulint *m;
  ulint dim;
  ulint l;
  ap_no_l_ulint zc;
{
  ulint c;

  for (c=1; c<=dim; c++)
  {
    if (le_elem(m, dim, l, c)==0 && zc->i > 0) (zc->i)--;
    zc=zc->seguinte;
  }
}

/*int alg_hung_p2(m, dim, ll, tam_ll, lc, tam_lc)
 Passo 2 do Algoritmo Hungaro
- cobre todos os zeros da matriz de afectacao com o menor numero
possivel de
linhas horizontais (lista de linhas ll) e verticais (lista de linhas lc)

  ulint *m;
  ulint dim;
  ap_no_l_ulint *ll;
  ulint *tam_ll;
  ap_no_l_ulint *lc;
  ulint *tam_lc;
{
  ulint l, max_zl, max_zc, n_max_zl, n_max_zc, n_zeros;
  ap_no_l_ulint u_ll=NULL, u_lc=NULL, zl, zc, p;
  int noerrors;

  *ll=NULL; *tam_ll=0; *lc=NULL; *tam_lc=0;

  noerrors=forma_zl_e_zc(m, dim, &zl, &zc, &n_zeros);
  while (noerrors && n_zeros>0)
  {
    max_l_ulint(zl, &max_zl, &n_max_zl);
```

```
    max_l_ulint(zc, &max_zc, &n_max_zc);

    if ( max_zl>max_zc || (max_zl==max_zc && n_max_zl>n_max_zc) ||
      (max_zl==max_zc && n_max_zl==n_max_zc && *tam_ll<= *tam_lc) )
    {
      p=zl; l=1;
      while (p!=NULL && noerrors)
      {
    if (p->i==max_zl)
    {
      noerrors=insere_ulint(l, ll, &u_ll, 0);
      if (noerrors)
      {
        (*tam_ll)++;
        n_zeros-=p->i;
        p->i=0;
        act_zc(m, dim, l, zc);
      }
    }
    l++;
    p=p->seguinte;
      }
    }
    else
    {
      p=zc; l=1;
      while (p!=NULL && noerrors)
      {
    if (p->i==max_zc)
    {
      noerrors=insere_ulint(l, lc, &u_lc, 0);
      if (noerrors)
      {
        (*tam_lc)++;
        n_zeros-=p->i;
        p->i=0;
        act_zl(m, dim, l, zl);
      }
    }
    l++;
    p=p->seguinte;
      }
    }
  }
  liimpa_l_ulint(zl);
  liimpa_l_ulint(zc);
  reeturn noerrors;
}*/

int  forma_l_zeros(m, dim, lzeros)
/*
- fcorma uma lista de pares ordenados (linha, coluna), que identificam as
posiicoes dos zeros de uma matriz
- a saida da funcao '*lzeros' contem um apontador para a cabeca da lista
- reetorna 1 se for bem sucedida ou 0 se ocorrer algum erro
*/
  ullint *m;
  ullint dim;
  ap_no_l_2uint *lzeros;
```

```
{
  ulint l, c;
  int noerrors=1;
  ap_no_l_2uint u=NULL;

  *lzeros=NULL;

  for (l=1; (l<=dim && noerrors); l++)
    for (c=1; (c<=dim && noerrors); c++)
      if (le_elem(m, dim, l, c)==0) noerrors=insere_2uint(l, c, lzeros,
&u);
  if (!noerrors)
  {
    limpa_l_2uint(*lzeros);
    *lzeros=NULL;
  }
  return noerrors;
}

void limpa_zeros_linha(linha, lzeros)
/*
- limpa todos os elementos da lista de zeros com o campo i1==linha
- actualiza o apontador *lzeros
*/
  ulint linha;
  ap_no_l_2uint *lzeros;
{
  ap_no_l_2uint p=*lzeros, ant=NULL, aux;

  while (p!=NULL)
    if (p->i1==linha)
    {
      aux=p->seguinte;
      if (ant!=NULL) ant->seguinte=aux;
    else *lzeros=aux;
      free(p);
      p=aux;
    }
    else
    { ant=p; p=p->seguinte; }
}

void limpa_zeros_coluna(coluna, lzeros)
/*
- limpa todos os elementos da lista de zeros com o campo i2==coluna
- actualiza o apontador *lzeros
*/
  ulint coluna;
  ap_no_l_2uint *lzeros;
{
  ap_no_l_2uint p=*lzeros, ant=NULL, aux;

  while (p!=NULL)
    if (p->i2==coluna)
    {
      aux=p->seguinte;
      if (ant!=NULL) ant->seguinte=aux;
    else *lzeros=aux;
      free(p);
```

```
        p=aux;
      }
      else
      { ant=p; p=p->seguinte; }
}


/* NOVA VERSAO do PASSO 2 do ALGORITMO HUNGARO */

ulint nit; /* usada para contar o numero de iteracoes de cobre_zeros */

int  cobre_zeros(lzeros, ll, u_ll, tam_ll, lc, u_lc, tam_lc)
/*
- allgoritmo recursivo que cobre os zeros de uma iatriz com o numero
miniimo de tracos horizontais (linhas que constem em ll) e verticais
(collunas que constem em lc)
*/
  app_no_l_2uint lzeros;
  app_no_l_ulint *ll;
  app_no_l_ulint u_ll;
  ullint *tam_ll;
  app_no_l_ulint *lc;
  app_no_l_ulint u_lc;
  ullint *tam_lc;
{
  ullint l, c, tam_ll2, tam_lc2;
  irnt noerrors=1;
  app_no_l_ulint ll2, u_ll2, lc2, u_lc2;
  app_no_l_2uint lheros2;

  iif (lzeros!=NULL)
  {
     l=lzeros->i1;
     c=lzeros->i2;
     remove_cab_l_2uint(&lzeros);

     tam_ll2=*tam_ll;
     noerrors=copia_l_ulint(*ll, &ll2, &u_ll2);
     if (noerrors)
     {
       tam_lc2=*tam_lc;
       noerrors=copia_l_ulint(*lc, &lc2, &u_lc2);
       if (noerrors)
       {
      noerrors=copia_l_2uint(lzeros, &lzeros2);
      if (noerrors)
      {
        noerrors=insere_ulint(l, ll, &u_ll, 1);
        if (noerrors)
        {
          (*tam_ll)++;
          limpa_zeros_linha(l, &lzeros);

          noerrors=insere_ulint(c, &lc2, &u_lc2, 1);
          if (noerrors)
          {
            tam_lc2++;
            limpa_zeros_coluna(c, &lzeros2);
```

```
            if (cobre_zeros(lzeros, ll, u_ll, tam_ll,
                    lc, u_lc, tam_lc) &&
                cobre_zeros(lzeros2, &ll2, u_ll2, &tam_ll2,
                    &lc2, u_lc2, &tam_lc2)
              )
            {
             if ((tam_ll2+tam_lc2)<(*tam_ll+*tam_lc) ||
                 ((tam_ll2+tam_lc2)==(*tam_ll+*tam_lc) &&
                    (tam_ll2*tam_lc2)>((*tam_ll)*(*tam_lc))
                 )
                )
             {
               limpa_l_ulint(*ll);
               limpa_l_ulint(*lc);
               *ll=ll2;
               *lc=lc2;
               *tam_ll=tam_ll2;
               *tam_lc=tam_lc2;
               ll2=NULL;
               lc2=NULL;
             }
            }
            else noerrors=0;
          }
        }
      }
      limpa_l_ulint(lc2);
        }
        limpa_l_ulint(ll2);
      }
    }
    else nit++;

    return noerrors;
}

int alg_hung_p2(m, dim, ll, tam_ll, lc, tam_lc)
/* Passo 2 do Algoritmo Hungaro
- cobre todos os zeros da matriz de afectacao com o menor numero
possivel de
linhas horizontais (lista de linhas ll) e verticais (lista de linhas lc)
*/
    ulint *m;
    ulint dim;
    ap_no_l_ulint *ll;
    ulint *tam_ll;
    ap_no_l_ulint *lc;
    ulint *tam_lc;
{
    ap_no_l_2uint lzeros;
    int noerrors;

    *ll=NULL; *tam_ll=0; *lc=NULL; *tam_lc=0;

    noerrors=forma_l_zeros(m, dim, &lzeros);
    nit=0;
    if (noerrors) noerrors=cobre_zeros(lzeros, ll, NULL, tam_ll,
                    lc, NULL, tam_lc);
```

```
    sprintf(s, "\n    numero de solucoes testadas=%u\n", nit);
    prn_string(s);

    return noerrors;
  }

  ulint alg_hung_p4(m, dim, ll, lc)
  /* Passo 4 do Algoritmo Hungaro
  - determina o elemento minimo dentre os elementos da matriz m que nao
  pertencem as
  linhas na lista ll ou as colunas na lista lc
  */
    ulint *m;
    ulint dim;
    ap_no_l_ulint ll;
    ap_no_l_ulint lc;
  {
    ulint l, c, minimo=MAXINT;

    for (l=1; l<=dim; l++)
      for (c=1; c<=dim; c++)
        if (procura_ulint(l, ll)==NULL && procura_ulint(c, lc)==NULL &&
        le_elem(m, dim, l, c)<minimo) minimo=le_elem(m, dim, l, c);
    return minimo;
  }

  void alg_hung_p5(m, dim, ll, lc, emin)
  /* Passo 5 do Algoritmo Hungaro
  - para todos os elementos da matriz m que nao pertencem a linhas na
  lista ll ou a
  colunas na lista lc, subtrai emin
  - para todos os elementos da matriz m no cruzamento das linhas na lista
  ll com as
  colunas na lista lc, adiciona emin
  */
    uliint *m;
    uliint dim;
    ap_no_l_ulint ll;
    ap_no_l_ulint lc;
    uliint emin;
  {
    uliint l, c;
    ap_no_l_ulint pl, pc;

    for (l=1; l<=dim; l++)
    {
      pl=procura_ulint(l, ll);
      for (c=1; c<=dim; c++)
      {
        pc=procura_ulint(c, lc);
        if (pl==NULL && pc==NULL) escr_elem(m, dim, l, c, le_elem(m, dim,
  l, c)-emin);
        if (pl!=NULL && pc!=NULL) escr_elem(m, dim, l, c, le_elem(m, dim,
  l, c)+emin);
      }
    }
  }

  ap_no_l_2uint alg_hung_p6(m, dim)
```

```
/* Passo 6 do Algoritmo Hungaro
- retorna um apontador para uma lista de afectacoes
- o campo i1 e a linha e o campo i2 e a coluna
*/
  ulint *m;
  ulint dim;
{
  int noerrors;
  ulint tam_sol=0, n, nmin, l, c;
  ap_no_l_ulint zl, zc;
  ap_no_l_2uint sol=NULL, u_sol=NULL, lzeros, pz, pz_min, ulz, aux;

  noerrors=forma_zl_zc_lzeros(m, dim, &zl, &zc, &lzeros);

  while (tam_sol<dim && noerrors && lzeros!=NULL)
  {
/*    prn_string("\nlzeros=");
    prn_l_2uint(lzeros);
    prn_string("\nzl=");
    prn_l_ulint(zl);
    prn_string("\nzc=");
    prn_l_ulint(zc);
*/
    pz=lzeros; pz_min=NULL; nmin=MAXINT;
    while (pz!=NULL)
    {
    n=elem_l_ulint(pz->i1, zl)+elem_l_ulint(pz->i2, zc)-1;
    if (pz_min==NULL || n<nmin)
    {
      pz_min=pz; nmin=n;
    }
/*  sprintf(s, "\n(%u,%u):%u", pz->i1, pz->i2, n); prn_string(s);*/
    pz=pz->seguinte;
    }

    l=pz_min->i1; c=pz_min->i2;
    noerrors=insere_2uint(l, c, &sol, &u_sol);
    if (noerrors)
    {
      tam_sol++;
      act_zc(m, dim, l, zc);
      act_zl(m, dim, c, zl);
      atribui_ulint_pos(0, l, zl);
      atribui_ulint_pos(0, c, zc);
    }

    pz=lzeros; ulz=NULL;
    while (pz!=NULL)
      if (pz->i1==l || pz->i2==c)
      {
    if (pz==lzeros) lzeros=pz->seguinte;
          else ulz->seguinte=pz->seguinte;
    aux=pz->seguinte;
    free(pz);
    pz=aux;
      }
      else
      {
    ulz=pz;
```

```
    pz=pz->seguinte;
      }
  }

  limpa_l_ulint(zl);
  limpa_l_ulint(zc);
  limpa_l_2uint(lzeros);

  if (noerrors) return sol;
  else
  {
    limpa_l_2uint(sol);
    return NULL;
  }
}

ap_no_l_2uint algoritmo_hungaro(m, dim)
/*
- implementa o Algoritmo Hungaro (algoritmo de afectacao optima)
*/
  uint *m;
  uint dim;
{
  ap_no_l_ulint ll, lc;
  uint tam_ll, tam_lc, emin;
  int noerrors;

  alg_hung_p1(m, dim);

  do{
    prn_string("\n   matriz de eficiencia apos uma iteracao do Alg.
Hungaro:");
    prn_matriz(m, dim);

    noerrors=alg_hung_p2(m, dim, &ll, &tam_ll, &lc, &tam_lc);

    prn_string("\n   menor numero de tracos que cobrem os zeros da
matriz (passo 2):\n    linhas={");
    prn_l_ulint(ll);
    prn_string("}; colunas={");
    prn_l_ulint(lc);
    sprintf(s, "}; num_tracos=%u\n\n", tam_ll+tam_lc);
    prn_string(s);

    if (noerrors && (tam_ll+tam_lc)<dim)
    {
       emin=alg_hung_p4(m, dim, ll, lc);
       alg_hung_p5(m, dim, ll, lc, emin);
    }
    limpa_l_ulint(ll);
    limpa_l_ulint(lc);
  }while (noerrors && (tam_ll+tam_lc)<dim);

  if (noerrors) return (alg_hung_p6(m, dim));
    else return NULL;
}

void act_lagvs_e_ftar(l_af, tar_af, num_tar_af, agvs_af, num_agvs_af,
    layout, poss_acesso, tar_mud_est, agvs_reaf, t)
```

```
/*
- actualiza a informacao guardada na lista de AGV's e na fila de
Tarefas, de
acordo com o resultado do algoritmo de escalonamento (lista de
afectacoes
l_af)
*/
  ap_no_l_2uint l_af;
  ap_no_l_ap_Tarefas tar_af;
  ulint num_tar_af;
  ap_no_l_ap_AGVs agvs_af;
  ulint num_agvs_af;
  ap_no_l_nos layout;
  int poss_acesso;
  ap_no_l_ap_Tarefas *tar_mud_est;
  ap_no_l_ap_AGVs *agvs_reaf;
  double t;
{
  ap_AGV agv;
  ap_Tarefa tarefa;
  ap_no_l_ap_Tarefas u_tar_mud_est=NULL;
  ap_no_l_ap_AGVs u_agvs_reaf=NULL;

  while (l_af!=NULL)
  {
    if (l_af->i1 <= num_agvs_af) agv=obtem_ap_agv_pos(agvs_af, l_af-
>i1);
    else agv=NULL;
    if (l_af->i2 <= num_tar_af) tarefa=obtem_ap_tarefa_pos(tar_af, l_af-
>i2);
    else tarefa=NULL;

    if (agv!=NULL) /* AGV nao ficticio */
      if (tarefa!=NULL) /* tarefa nao ficticia */
      {
    if (agv->tarefa_agv!=tarefa) /* se o AGV e afectado a outra tarefa
*/
    {
      if (agv->tarefa_agv!=NULL)
           insere_no_l_ap_AGVs(agv, agvs_reaf, &u_agvs_reaf);

      limpa_l_apno(&tarefa->traj1); /* limpa antigo trajecto 1 se
existir */
      tarefa->ctraj1=MAXINT;
      limpa_l_apno(&tarefa->traj2); /* limpa antigo trajecto 2 se
existir */
      tarefa->ctraj2=MAXINT;

      if (agv->posicao!=tarefa->no1 || poss_acesso==0 || agv-
>orient*agv->posicao->pos_rel_mesa==poss_acesso)
      {
        tarefa->ctraj1=traj_optimo(layout, agv->posicao, tarefa->no1,
                  NULL, NULL, poss_acesso, &tarefa->traj1);
        if (tarefa->tipo_tar!=T_CARGA)
          if (tarefa->no1!=tarefa->no2)
            tarefa->ctraj2=traj_optimo(layout, tarefa->no1, tarefa->no2,
                  NULL, NULL, poss_acesso, &tarefa->traj2);
           else tarefa->ctraj2=0;
```

```
        if ( (agv->posicao!=tarefa->no1 && tarefa->traj1==NULL) ||
             (tarefa->no1!=tarefa->no2 && tarefa->traj2==NULL && tarefa-
>tipoo_tar==NORMAL) )
        /* se a tarefa nao for fasivel ou se ocorrer algum erro na det.
de trajectos */
        {
          act_estat_temporais_tarefa(tarefa, t);
          tarefa->estado_tar=REJEITADA;
          tarefa->agv_tarefa=NULL;
          /* campo prior utilizado para guardar a razao do cancelamento
de uuma tarefa */
          if (agv->posicao!=tarefa->no1 && tarefa->traj1==NULL)
           tarefa->prior=CAMINHO1_INEX;
           else tarefa->prior=CAMINHO2_INEX;
          agv->tarefa_agv=NULL;
        }
        else
        {
           tarefa->agv_tarefa=agv;
           (tarefa->num_af)++;
           agv->tarefa_agv=tarefa;
           if (tarefa->estado_tar==FILA)
           {
             act_estat_temporais_tarefa(tarefa, t);
             tarefa->estado_tar=PASSO1;
            }
        }
      }
      else /* tarefa nao fasivel por este AGV */
      {
          act_estat_temporais_tarefa(tarefa, t);
          tarefa->estado_tar=REJEITADA;
          tarefa->agv_tarefa=NULL;
          /* campo prior utilizado para guardar a razao do cancelamento
de uuma tarefa */
          tarefa->prior=OP_CAR_IMP;
          agv->tarefa_agv=NULL;
      }
      insere_no_l_ap_Tarefas(tarefa, tar_mud_est, &u_tar_mud_est);
    }
    /* se o AGV e afectado a mesma tarefa nao e preciso fazer nada */
      }
      else /* AGV afectado a uma tarefa ficticia */
      {
    if (agv->tarefa_agv!=NULL)
    {
      agv->tarefa_agv=NULL;
      insere_no_l_ap_AGVs(agv, agvs_reaf, &u_agvs_reaf);
    }
      }
    else /* AGV ficticio */
      if (tarefa!=NULL) /* tarefa afectada a um AGV ficticio */
      {
    if (tarefa->estado_tar==PASSO1)
    {
      act_estat_temporais_tarefa(tarefa, t);
      tarefa->estado_tar=FILA;
      tarefa->agv_tarefa=NULL;
```

```
        limpa_l_apno(&tarefa->traj1); /* limpa antigo trajecto 1 se
existir */
        tarefa->ctraj1=MAXINT;
        limpa_l_apno(&tarefa->traj2); /* limpa antigo trajecto 2 se
existir */
        tarefa->ctraj2=MAXINT;
        insere_no_l_ap_Tarefas(tarefa, tar_mud_est, &u_tar_mud_est);
      }
        }
        /* se a tarefa tambem for ficticia nao e preciso fazer nada */
      l_af=l_af->seguinte;
    }
    limpa_l_2uint(l_af);
    limpa_l_ap_Tarefas(&tar_af);
    limpa_l_ap_AGVs(&agvs_af);
}

void escal_desp_VMP_TMCV(tarefas, agvs, layout, poss_acesso, num_max_af,
reaf_dinam,
            tar_mud_est, agvs_reaf, t)
/*
- efectua o escalonamento e despacho de tarefas nao escalonadas pelo SGM
- parte-se do principio que existe pelo menos uma tarefa nao escalonada
e
um agv livre
*/
    ap_Tarefa tarefas;
    ap_AGV agvs;
    ap_no_l_nos layout;
    int poss_acesso;
    ulint num_max_af;
    int reaf_dinam;
    ap_no_l_ap_Tarefas *tar_mud_est;
    ap_no_l_ap_AGVs *agvs_reaf;
    double t;
{
    ap_no_l_ap_Tarefas tar_af;
    ap_no_l_ap_AGVs agvs_af;
    ulint num_af, num_tar_af, num_agvs_af;
    ulint *matr_efic;
    ap_no_l_2uint l_af;

    prn_string("\nescalonador do tipo NV/TMCV:");
    num_af=deter_num_af(tarefas, agvs, num_max_af, reaf_dinam);
    sprintf(s, "\n  numero de afectacoes=%u\n", num_af);
    prn_string(s);

    *tar_mud_est=NULL;
    *agvs_reaf=NULL;

    if (num_af>0)
    {
      prn_string("  forma lista de tarefas e lista de agvs
intervenientes...\n");
      if (forma_l_tar_e_agvs_af(num_af, tarefas, agvs, reaf_dinam,
            &tar_af, &num_tar_af, &agvs_af, &num_agvs_af))
      {
        prn_string("   Lista de Tarefas a Escalonar: ");
        prn_l_ap_Tarefas(tar_af);
```

```
      prn_string("   Lista de AGV's a Escalonar: ");
      prn_l_ap_AGVs(agvs_af);
      prn_string("\n  forma matriz de eficiencia...\n");
      matr_efic=forma_matr_efic(num_af, tar_af, agvs_af, layout,
posss_acesso);
      if (matr_efic!=NULL)
      {
    prn_string("   matriz de eficiencia:");
    prn_matriz(matr_efic, num_af);
    prn_string("\n  executa o algoritmo hungaro...\n");
    l_af=algoritmo_hungaro(matr_efic, num_af);
    prn_string("  lista de afectacoes: ");
    prn_l_2uint(l_af);
    destroi_matriz(matr_efic);
    if (l_af!=NULL)
    {
      prn_string("\n  actualiza a lista de AGVs e a fila de
Tareefas...\n");
      act_lagvs_e_ftar(l_af, tar_af, num_tar_af, agvs_af, num_agvs_af,
           layout, poss_acesso, tar_mud_est, agvs_reaf, t);
    }
    else prn_string("  erro no algoritmo hungaro\n");
      }
      else prn_string("  erro no calculo da matriz de eficiencia\n");
    }
    else prn_string("  erro na formacao das listas\n");
  }
  ellse prn_string("  numero de afectacoes nulo\n");
  prrn_string("Fim do escalonador VMP/TMCV\n");
}

voidd determ_primeiro_AGV_dispon_e_tarefa_pendente(tarefas,
    agvs, tarefa, agv)
/*
- deetermina o primeiro AGV que ficou disponivel e a primeira Tarefa
penddente a ficar no estado FILA;
- essta informacao e utilizada para determinar se a afectacao e do tipo
"dessencadeada pelo centro de trabalho" ou do tipo "desencadeada pelo
veicculo".
*/
  app_Tarefa tarefas;
  app_AGV agvs;
  app_Tarefa *tarefa;
  app_AGV *agv;
{
  *ttarefa=NULL;
  whnile (tarefas!=NULL)
  {
     if (tarefas->estado_tar==FILA &&
     tarefas->agv_tarefa==NULL &&
     (tarefas->tipo_tar==NORMAL ||
      tarefas->tipo_tar==T_CARGA) &&
     (*tarefa==NULL || tarefas->t_mud_estado<(*tarefa)->t_mud_estado)
        ) *tarefa=tarefas;
    tarefas=tarefas->seguinte;
  }

  *agv=NULL;
  while (agvs!=NULL)
```

```
    {
      if (agvs->estado_movim==PARADO &&
      agvs->estado_alocacao==NAO_ALOCADO &&
      agvs->estado_carga==DESCARREGADO &&
      agvs->tarefa_agv==NULL &&
      (*agv==NULL || agvs->t_mud_estado<(*agv)->t_mud_estado)
         ) *agv=agvs;
      agvs=agvs->seguinte;
    }
}

int cria_lista_tarefas_escalonaveis(tarefas, num_max_af,
tar_escalonaveis)
/*
- cria uma lista de apontadores para as tarefas que irao intervir no
despacho desencadeado por um Veiculo que ficou disponivel;
- retorna 0 se ocorrer algum erro ou 1 no caso contrario
*/
  ap_Tarefa tarefas;
  ulint num_max_af;
  ap_no_l_ap_Tarefas *tar_escalonaveis;
{
  ulint num_tar_escalonaveis=0, pmax;
  int ha_mais_tarefas=1, noerrors=1;
  ap_Tarefa tar, tar_sel;
  ap_no_l_ap_Tarefas h_tar_sel=NULL, u_tar_sel=NULL,
u_tar_escalonaveis=NULL;

  *tar_escalonaveis=NULL;

  while (num_tar_escalonaveis<num_max_af && ha_mais_tarefas && noerrors)
  {
    tar=tarefas; pmax=0; tar_sel=NULL;
    while (tar!=NULL)
    {
    if (tar->estado_tar==FILA &&
        tar->agv_tarefa==NULL &&
        (tar->tipo_tar==NORMAL ||
         tar->tipo_tar==T_CARGA) &&
        (tar->prior>pmax || (tar->prior==pmax && tar_sel==NULL)) &&
        !existe_no_l_ap_Tarefas(tar, h_tar_sel)
       )
    {
      tar_sel=tar; pmax=tar->prior;
    }
    tar=tar->seguinte;
    }

    if (tar_sel==NULL) ha_mais_tarefas=0;
    else
     if (insere_no_l_ap_Tarefas(tar_sel, &h_tar_sel, &u_tar_sel) &&
      insere_no_l_ap_Tarefas(tar_sel, tar_escalonaveis,
&u_tar_escalonaveis)
    ) num_tar_escalonaveis++;
     else noerrors=0;
  }

  limpa_l_ap_Tarefas(&h_tar_sel);
  if (noerrors) return 1;
```

```
  else
  {
    limpa_l_ap_Tarefas(tar_escalonaveis);
    return 0;
  }
}

int cria_lista_agvs_escalonaveis(agvs, num_max_af, agvs_escalonaveis)
/*
- cria uma lista de apontadores para os AGV's que irao intervir no
despacho desencadeado por um Centro de Trabalho que pediu um movimento;
- retorna 0 se ocorrer algum erro ou 1 no caso contrario
*/
  ap_AGV agvs;
  ul_int num_max_af;
  ap_no_l_ap_AGVs *agvs_escalonaveis;
{
  ul_int num_agvs_escalonaveis=0;
  ap_no_l_ap_AGVs u_agvs_escalonaveis=NULL;
  in_t noerrors=1;

  *agvs_escalonaveis=NULL;

  while (agvs!=NULL && num_agvs_escalonaveis<num_max_af && noerrors)
  {
    if (agvs->estado_movim==PARADO &&
     agvs->tarefa_agv==NULL &&
     agvs->estado_alocacao==NAO_ALOCADO &&
     agvs->estado_carga==DESCARREGADO)
      if (insere_no_l_ap_AGVs(agvs, agvs_escalonaveis,
&u_agvs_escalonaveis))
     num_agvs_escalonaveis++;
     else noerrors=0;
    agvs=agvs->seguinte;
  }

  if (noerrors) return 1;
  else
  {
    limpa_l_ap_AGVs(agvs_escalonaveis);
    return 0;
  }
}

ap_Tarefa aplica_regra_TMCV(tarefas, agv, layout, poss_acesso)
/*
- aplica a regra de despacho "Tempo Mais Curto de Viagem", utilizada
na afectacao desencadeada por um Veiculo
*/
  ap_no_l_ap_Tarefas tarefas;
  ap_AGV agv;
  ap_no_l_nos layout;
  in_t poss_acesso;
{
  ul_int cmin=MAXINT, c;
  ap_Tarefa tar_sel=NULL;

  while (tarefas!=NULL)
  {
```

```
    if (agv->posicao!=tarefas->tarefa->no1)
    c=custo_traj_optimo(layout, agv->posicao, tarefas->tarefa->no1,
          NULL, NULL, poss_acesso);
    else if (poss_acesso==0 ||
          agv->orient*agv->posicao->pos_rel_mesa==poss_acesso)
                c=0;
          else c=MAXINT;
    if (c<cmin) { cmin=c; tar_sel=tarefas->tarefa; }
    tarefas=tarefas->seguinte;
  }

  return tar_sel;
}

ap_Tarefa aplica_regra_MTFS(tarefas, est_filas)
/*
- aplica a regra de despacho "Maximo Tamanho da Fila de Saida",
utilizada
na afectacao desencadeada por um Veiculo
*/
  ap_no_l_ap_Tarefas tarefas;
  ap_EstadoFila est_filas;
{
  ulint tam_max=0, tam;
  ap_Tarefa tar_sel=NULL;
  ap_EstadoFila fila;

  while (tarefas!=NULL)
  {
    fila=procura_EstadoFila(tarefas->tarefa->no1->id_no, est_filas);
    if (fila!=NULL) tam=fila->Lf;
    else tam=0;
    if (tam>tam_max) { tam_max=tam; tar_sel=tarefas->tarefa; }
    tarefas=tarefas->seguinte;
  }

  return tar_sel;
}

ap_Tarefa aplica_regra_MELFS(tarefas, est_filas)
/*
- aplica a regra de despacho "Minimo Espaco Livre na Fila de Saida",
utilizada na afectacao desencadeada por um Veiculo
*/
  ap_no_l_ap_Tarefas tarefas;
  ap_EstadoFila est_filas;
{
  ulint espaco_min=MAXINT, espaco;
  ap_Tarefa tar_sel=NULL;
  ap_EstadoFila fila;

  while (tarefas!=NULL)
  {
    fila=procura_EstadoFila(tarefas->tarefa->no1->id_no, est_filas);
    if (fila!=NULL) espaco=fila->Cf-fila->Lf;
    else espaco=MAXINT;
    if (espaco<espaco_min) { espaco_min=espaco; tar_sel=tarefas->tarefa;
}
    tarefas=tarefas->seguinte;
```

```
  }

  return tar_sel;
}

ap_Tarefa aplica_regra_PCPSM(tarefas, filas)
/*
- aplica a regra de despacho "Primeira a Chegar, Primeira Servida,
Modificada",
utilizada na afectacao desencadeada por um Veiculo
*/
  ap_no_l_ap_Tarefas tarefas;
  ap_EstadoFila filas;
{
  double t_min=1.0E99;
  ap_Tarefa tar_sel=NULL;

  while (filas!=NULL)
  {
    if (filas->tar_mais_ant!=NULL &&
    filas->t_MFCFS<t_min &&
    existe_no_l_ap_Tarefas(filas->tar_mais_ant, tarefas)
       )
    { tar_sel=filas->tar_mais_ant; t_min=filas->t_MFCFS; }
    filas=filas->seguinte;
  }

  return tar_sel;
}

ap_AGV aplica_regra_VMP(tarefa, agvs, layout, poss_acesso)
/*
- aplica a regra de despacho "Veiculo Mais Perto",
utilizada na afectacao desencadeada por um Centro de Trabalho
*/
  ap_Tarefa tarefa;
  ap_no_l_ap_AGVs agvs;
  ap_no_l_nos layout;
  int poss_acesso;
{
  ulint cmin=MAXINT, c;
  ap_AGV agv_sel=NULL;

  while (agvs!=NULL)
  {
    if (agvs->agv->posicao!=tarefa->no1)
    c=custo_traj_optimo(layout, agvs->agv->posicao, tarefa->no1,
           NULL, NULL, poss_acesso);
    else if (poss_acesso==0 ||
           agvs->agv->orient*agvs->agv->posicao-
>pos_rel_mesa==poss_acesso)
                c=0;
           else c=MAXINT;
    if (c<cmin) { cmin=c; agv_sel=agvs->agv; }
    agvs=agvs->seguinte;
  }

  return agv_sel;
}
```

```
ap_AGV aplica_regra_VMU(agvs, t)
/*
- aplica a regra de despacho "Veiculo Menos Utilizado",
utilizada na afectacao desencadeada por um Centro de Trabalho
*/
  ap_no_l_ap_AGVs agvs;
  double t;
{
  double taxa_min=1.0E99, soma1, soma2;
  ap_AGV agv_sel=NULL;

  while (agvs!=NULL)
  {
    soma1=t-agvs->agv->t_mud_estado+agvs->agv->dt_espera+
    agvs->agv->dt_bloqueado+agvs->agv->dt_erro;
    soma2=agvs->agv->dt_movim_descarr+agvs->agv->dt_carga+
    agvs->agv->dt_movim_carr+agvs->agv->dt_descarga;
    if (soma1+soma2>0.0) agvs->agv->taxa_util=soma2/(soma1+soma2)*100.0;
    else agvs->agv->taxa_util=100.0;

    if (agvs->agv->taxa_util<taxa_min)
    { agv_sel=agvs->agv; taxa_min=agvs->agv->taxa_util; }
    agvs=agvs->seguinte;
  }

  return agv_sel;
}

void afecta_agv_a_tarefa(agv, tarefa, layout, poss_acesso, t,
tar_mud_est)
/*
- afecta um AGV a uma tarefa de acordo com a decisao resultante da
aplicacao
de uma regra de despacho
*/
  ap_AGV agv;
  ap_Tarefa tarefa;
  ap_no_l_nos layout;
  int poss_acesso;
  double t;
  ap_no_l_ap_Tarefas *tar_mud_est;
{
  ap_no_l_ap_Tarefas u_tar_mud_est=NULL;

  *tar_mud_est=NULL;

  act_estat_temporais_tarefa(tarefa, t);

  if (agv->posicao!=tarefa->no1 || poss_acesso==0 ||
      agv->orient*agv->posicao->pos_rel_mesa==poss_acesso)
  {
    tarefa->ctraj1=traj_optimo(layout, agv->posicao, tarefa->no1,
    NULL, NULL, poss_acesso, &tarefa->traj1);
    if (tarefa->tipo_tar!=T_CARGA)
      if (tarefa->no1!=tarefa->no2)
    tarefa->ctraj2=traj_optimo(layout, tarefa->no1, tarefa->no2,
    NULL, NULL, poss_acesso, &tarefa->traj2);
    else { tarefa->ctraj2=0; tarefa->traj2=NULL; }
```

```
    iif ((agv->posicao!=tarefa->no1 && tarefa->traj1==NULL) ||
     (tarefa->no1!=tarefa->no2 && tarefa->traj2==NULL && tarefa-
>tipco_tar==NORMAL)
        )
     {
     tarefa->estado_tar=REJEITADA;
     tarefa->agv_tarefa=NULL;
     /* campo prior utilizado para comunicar a razao da rejeicao da
Tareefa */
     if (agv->posicao!=tarefa->no1 && tarefa->traj1==NULL)
           tarefa->prior=CAMINHO1_INEX;
     else tarefa->prior=CAMINHO2_INEX;
     agv->tarefa_agv=NULL;
     }
    else
     {
     tarefa->agv_tarefa=agv;
     (tarefa->num_af)++;
     agv->tarefa_agv=tarefa;
     tarefa->estado_tar=PASSO1;
     }
  }
  ellse
  {
     tarefa->estado_tar=REJEITADA;
     tarefa->agv_tarefa=NULL;
     /* campo prior utilizado para comunicar a razao da rejeicao da
Tareefa */
     tarefa->prior=OP_CAR_IMP;
     agv->tarefa_agv=NULL;
  }

  innsere_no_l_ap_Tarefas(tarefa, tar_mud_est, &u_tar_mud_est);
}

     /* funcao de escalonamento e de despacho principal */
voidd escalon_despacho(regra_ADCT, regra_ADV, tarefas, agvs, layout,
  pcoss_acesso, num_max_af, reaf_dinam, est_filas, t, tar_mud_est,
agvss_reaf)
/*
- fuuncao de escalonamento e de despacho principal
*/
  irnt regra_ADCT;
  irnt regra_ADV;
  app_Tarefa tarefas;
  app_AGV agvs;
  app_no_l_nos layout;
  irnt poss_acesso;
  ullint num_max_af;
  irnt reaf_dinam;
  app_EstadoFila est_filas;
  double t;
  app_no_l_ap_Tarefas *tar_mud_est;
  app_no_l_ap_AGVs *agvs_reaf;
{
  app_Tarefa tarefa;
  app_AGV agv;
  app_no_l_ap_Tarefas tar_escalonaveis;
```

```
  ap_no_l_ap_AGVs agvs_escalonaveis;

  if (regra_ADCT==VMP && regra_ADV==TMCV)
    escal_desp_VMP_TMCV(tarefas, agvs, layout, poss_acesso,
          num_max_af, reaf_dinam, tar_mud_est, agvs_reaf, t);
  else
  {
    determ_primeiro_AGV_dispon_e_tarefa_pendente(tarefas, agvs, &tarefa,
&agv);
    sprintf(s, "\nEscalonador: 1. AGV disponivel=%u ; 1. Tarefa
pendente=%u\n",
    agv->id_agv, tarefa->id_tar);
    prn_string(s);

    if (tarefa->t_mud_estado<agv->t_mud_estado)
    {
      prn_string(" problema de afectacao do tipo \"Afectacao
Desencadeada por um Veiculo\"\n");
      cria_lista_tarefas_escalonaveis(tarefas, num_max_af,
&tar_escalonaveis);
      prn_string(" Lista de Tarefas Escalonaveis: ");
      prn_l_ap_Tarefas(tar_escalonaveis);

      switch (regra_ADV)
      {
    case TMCV:    prn_string(" aplica a regra TMCV\n");
                  tarefa=aplica_regra_TMCV(tar_escalonaveis, agv,
layout, poss_acesso);
                  break;
    case MTFS:    prn_string(" aplica a regra MTFS\n");
                  tarefa=aplica_regra_MTFS(tar_escalonaveis, est_filas);
                  break;
    case MELFS:   prn_string(" aplica a regra MELFS\n");
                  tarefa=aplica_regra_MELFS(tar_escalonaveis,
est_filas);
                  break;
    default:      prn_string(" aplica a regra PCPSM\n");
                  tarefa=aplica_regra_PCPSM(tar_escalonaveis,
est_filas);
      }
      limpa_l_ap_Tarefas(&tar_escalonaveis);
    }
    else
    {
      prn_string(" problema de afectacao do tipo \"Afectacao
Desencadeada por um Centro de Trabalho\"\n");
      cria_lista_agvs_escalonaveis(agvs, num_max_af,
&agvs_escalonaveis);
      prn_string(" Lista de AGV's Escalonaveis: ");
      prn_l_ap_AGVs(agvs_escalonaveis);

      switch (regra_ADCT)
      {
    case VMU:     prn_string(" aplica a regra VMU\n");
                  agv=aplica_regra_VMU(agvs_escalonaveis, t);
                  break;
    default:      prn_string(" aplica a regra VMP\n");
                  agv=aplica_regra_VMP(tarefa, agvs_escalonaveis,
                        layout, poss_acesso);
```

```
        }
        limpa_l_ap_AGVs(&agvs_escalonaveis);
      }

      sprintf(s, "\n foi decidido afectar o AGV %u a Tarefa %u\n",
      agv->id_agv, tarefa->id_tar);
      prn_string(s);

      afecta_agv_a_tarefa(agv, tarefa, layout, poss_acesso, t,
tar:_mud_est);
      *agvs_reaf=NULL;
    }}

  prn_string("\nLista de Tarefas:");
  prn_tarefas(tarefas);
  prn_string("\nLista de AGV's");
  prn_AGVs(agvs);
  prn_string("\nLista de Tarefas que mudaram de estado: ");
  prn_l_ap_Tarefas(*tar_mud_est);
  prn_string("\nLista de AGV's que foram reafectados: ");
  prn_l_ap_AGVs(*agvs_reaf);
}
```

A.2.1\2 sgm.c

```
/*  sgm.c */

#imclude <stdio.h>
#imclude <string.h>
#imclude <ctype.h> /* isdigit() */
#imclude "mens_log.h"
#imclude "listas.h"
#imclude "sgm_sga.h"
#imclude "sgm.h"
#imclude "lefich.h"
#imclude "fun_aux.h"
#imclude "sistagvs.h"
#imclude "rand_fun.h"


chaar s[255]; /* buffer utilizado para chamar 'prn_string()' */


/*  definicao das funcoes para leitura e validacao
do  ficheiro de configuracao do SGM */


ap_no_l_uid le_f_nos_tarefas(nome_ficheiro)
/*
- lle um ficheiro de configuracao de nos e respectivas probabilidades de
  cocorrencia de posicionamentos
- rretorna uma lista em que cada elemento contem a probabilidade do
re:spectivo no
  participar num posicionamento
- significado dos campos da estrutura:
  :i  : identificador do no
  cd : p_posic
*/
```

```
  char *nome_ficheiro;
{
  FILE *f;
  int c;
  ulint id_no, linha=1;
  double p_intr, p_posic;
  ap_no_l_uid h=NULL, u=NULL;

  sprintf(s, "\n\nSGM: leitura do ficheiro '%s' c/ probabilidades dos
nos...\n",
    nome_ficheiro);
  prn_string(s);
  f=fopen(nome_ficheiro,"r");
  if (f!=NULL)
  {
    do{
      c=procura_prox_linha(f, 'N', &linha);
      if (c!=EOF)
      {
    sprintf(s, "linha %u (no ", linha);
    prn_string(s);
    if (isdigit((c=fgetc(f))))
    {
      id_no=le_ulint(f, c-'0');
      sprintf(s, "%u)\n", id_no);
      prn_string(s);
      if (fgetc(f)==',')
      {
        le_double(f, &p_intr); /* p_introd (ignorado) */
          if (fgetc(f)==',')
          {
           le_double(f, &p_posic);
             if ((c=fgetc(f))=='\n')
             {
               linha++;
               u=insere_uid(h, u, id_no, p_posic, 0);
               if (h==NULL) h=u;
             }
             else {sprintf(s, "erro: caracter '%c' invalido\n",c);
                    prn_string(s); break;}
          }
          else {prn_string("erro: caracter ',' nao encontrado entre
p_introd e p_posic\n");break;}
      }
      else {prn_string("erro: caracter ',' nao encontrado entre id_no e
p_introd\n");break;}
    }
    else { prn_string("??)\n erro: leitura do id_no\n"); break; }
      }
      else { sprintf(s, " Fim do ficheiro na linha %u\n", linha);
             prn_string(s); }
    } while (c!=EOF);
    fclose(f);
  }
  else { sprintf(s, " erro: abertura do ficheiro %s\n", nome_ficheiro);
       prn_string(s); }
  if (c!=EOF) h=limpa_l_uid(h);
  return h;
}
```

```
int som_prob_nos_unit(nos_tar)
/*
- verifica se o somatorio das probabilidades dos nos das tarefas e 1.0
  para as cargas, descargas e posicionamentos
- se for retorna 1 senao retorna 0
*/
  ap_no_l_uid nos_tar;
{
  double s_pos=0.0;

  prn_string("\n\nSGM: verifica somatorio das probab. dos nos de
tarefas...\n");
  while (nos_tar!=NULL && s_pos<(1.0+TOLER_IGUAL_DOUBLES) )
  {
    if (nos_tar->d>=0.0 && nos_tar->d<=1.0) s_pos+=nos_tar->d;
    else
    {
      sprintf(s, " erro: no %u tem uma probab. de posicionamento
invalida =%.4f\n",
    nos_tar->i,nos_tar->d);
    prn_string(s);
      s_pos=2.0;
    }

    nos_tar=nos_tar->seguinte;
  }
  prn_string("Verificacao concluida.\n");
  if (!diguald2(s_pos,1.0,TOLER_IGUAL_DOUBLES))
    prn_string("Erro nas probabilidades de op. de posicionamento\n");
  if (diguald2(s_pos,1.0,TOLER_IGUAL_DOUBLES))
    return 1; else return 0;
}


/* definicao de funcoes para manipulacao das listas de AGV's */

int insere_AGV(agv, p, u)
/*
- insere um novo elemento no fim da lista ligada
- actualiza os apontadores *p (primeiro) e *u (ultimo)
- retorna 1 quando bem sucedida ou 0 se ocorrer algum erro
*/
  ap_AGV agv;
  ap_AGV *p, *u;
{
  ap_AGV novo;

  novo= (ap_AGV) malloc(sizeof(AGV));
  if (novo!=NULL)
  {
    novo->id_agv=agv->id_agv;
    novo->estado_movim=agv->estado_movim;
    novo->estado_alocacao=agv->estado_alocacao;
    novo->estado_carga=agv->estado_carga;
    novo->tarefa_agv=agv->tarefa_agv;
    novo->t_enable=agv->t_enable;
    novo->t_disable=agv->t_disable;
```

```
    novo->t_mud_est_aloc=agv->t_mud_est_aloc;
    novo->seguinte=NULL;
    novo->anterior=*u;
    if (*u!=NULL) (*u)->seguinte=novo;
    *u=novo;
    if (*p==NULL) *p=novo;
    return 1;
  }
  else prn_string("\"insere_AGV()\": erro de alocacao de memoria\n");
  return 0;
}

int remove_AGV(agv, p, u)
/*
- remove um elemento da lista de AGV's
- actualiza *p e *u quando isso for necessario
- retorna 1 quando e bem sucedida ou 0 no caso contrario
*/
  ap_AGV agv;
  ap_AGV *p, *u;
{
  if (agv!=NULL)
  {
    if (agv->anterior!=NULL) agv->anterior->seguinte=agv->seguinte;
    else *p=agv->seguinte;
    if (agv->seguinte!=NULL) agv->seguinte->anterior=agv->anterior;
    else *u=agv->anterior;
    free(agv);
    return 1;
  }
  else return 0;

/*  ap_AGV e=*p, ant=NULL, aux;
  int r=0;

  while (e!=NULL)
  {
    if (e->id_agv==id)
    {
      if (ant!=NULL) ant->seguinte=e->seguinte;
    else *p=e->seguinte;
      if (e->seguinte==NULL) *u=ant;
      aux=e->seguinte;
      free(e);
      e=aux;
      r=1;
    }
    else
    {
      ant=e;
      e=e->seguinte;
    }
  }
  return r;*/
}

ap_AGV obtem_AGV(id, p)
/*
```

```
- retorna o 1. elemento da lista com id_agv==id, percorrendo-a no
sentido
cabeca->cauda
- retorna NULL se nao for encontrado nenhum elemento
- p e cabeca da lista ligada
*/
 uint id;
 ap_AGV p;
{
 while (p!=NULL && p->id_agv!=id) p=p->seguinte;
 return p;
}

void limpa_l_AGVs(p)
/*
- destroi uma lista, comecando pela cabeca
*/
  ap_AGV p;
{
  ap_AGV prox;

  while (p!=NULL)
  {
     prox=p->seguinte;
     free((void *) p);
     p=prox;
  }
}

void prn_agv(agv)
/*
- imprime a informacao relativa a um AGV
*/
  ap_AGV agv;
{
  char s1[15];

  if (agv!=NULL)
  {
     sprintf(s, "AGV %u\tid_tarefa=", agv->id_agv);
     prn_string(s);

     if (agv->tarefa_agv!=NULL) sprintf(s, "%u", agv->tarefa_agv-
>id_tar);
     else sprintf(s, "---");
     prn_string(s);

     escr_str_est_movim_agv(s1, agv->estado_movim);
     sprintf(s, "\n estado=(%s,",s1);
     prn_string(s);

     escr_str_est_aloc_agv(s1, agv->estado_movim, agv->estado_alocacao);
     sprintf(s, "%s,",s1);
     prn_string(s);

     escr_str_est_carga_agv(s1, agv->estado_movim, agv->estado_carga);
     sprintf(s, "%s)\n t_enable=%.2f, t_disable=%.2f,
t_mud_est_aloc=%.2f\n",
     s1, agv->t_enable, agv->t_disable, agv->t_mud_est_aloc);
```

```
    prn_string(s);
    sprintf(s, "ult_destino=%u\n", agv->ult_destino);
    prn_string(s);
  }
  else prn_string("(NULL)\n");
}

void prn_l_AGVs(p)
/*
- imprime o conteudo de uma lista de AGV's, percorrendo-a no sentido
cabeca->cauda
*/
  ap_AGV p;
{
  while (p!=NULL)
  {
    prn_agv(p);
    p=p->seguinte;
  }
}


/* definicao de funcoes para manipulacao das listas de Tarefas */

int insere_Tarefa(tarefa, p, u)
/*
- insere um novo elemento no fim da lista ligada
- actualiza os apontadores *p (primeiro) e *u (ultimo)
- retorna 1 quando bem sucedida ou 0 se ocorrer algum erro
*/
  ap_Tarefa tarefa;
  ap_Tarefa *p, *u;
{
  ap_Tarefa novo;

  novo= (ap_Tarefa) malloc(sizeof(Tarefa));
  if (novo!=NULL)
  {
    novo->id_tar=tarefa->id_tar;
    novo->tipo_tar=tarefa->tipo_tar;
    novo->prior=tarefa->prior;
    novo->no1=tarefa->no1;
    novo->no2=tarefa->no2;
    novo->estado_tar=tarefa->estado_tar;
    novo->agv_tarefa=tarefa->agv_tarefa;
    novo->seguinte=NULL;
    novo->anterior=*u;
    if (*u!=NULL) (*u)->seguinte=novo;
    *u=novo;
    if (*p==NULL) *p=novo;
    return 1;
  }
  else prn_string("\"insere_Tarefa()\": erro de alocacao de memoria\n");
  return 0;
}

int remove_Tarefa(tarefa, p, u)
/*
- remove um elemento da lista de Tarefas
```

```
- actualiza *p e *u quando isso for necessario
- retorna 1 quando e bem sucedida ou 0 no caso contrario
*/
  ap_Tarefa tarefa;
  ap_Tarefa *p, *u;
{
  if (tarefa!=NULL)
  {
     if (tarefa->anterior!=NULL) tarefa->anterior->seguinte=tarefa->seguinte;
     else *p=tarefa->seguinte;
     if (tarefa->seguinte!=NULL) tarefa->seguinte->anterior=tarefa->anterior;
     else *u=tarefa->anterior;
     free(tarefa);
     return 1;
  }
  else return 0;

/*   ap_Tarefa e=*p, ant=NULL, aux;
  int r=0;

  while (e!=NULL)
  {
     if (e->id_tar==id)
     {
       if (ant!=NULL) ant->seguinte=e->seguinte;
     else *p=e->seguinte;
       if (e->seguinte==NULL) *u=ant;
       aux=e->seguinte;
       free(e);
       e=aux;
       r=1;
     }
     else
     {
       ant=e;
       e=e->seguinte;
     }
  }
  return r;*/
}

ap_Tarefa obtem_Tarefa(id, p)
/*
- retorna o 1. elemento da lista de Tarefas com id_tar==id, percorrendo-a no sentido
cabeca->cauda
- retorna NULL se nao for encontrado nenhum elemento
- p e cabeca da lista ligada
*/
 ulint id;
 ap_Tarefa p;
{
 while (p!=NULL && p->id_tar!=id) p=p->seguinte;
 return p;
}

void limpa_l_Tarefas(p)
```

```
/*
- destroi uma lista, comecando pela cabeca
*/
  ap_Tarefa p;
{
  ap_Tarefa prox;

  while (p!=NULL)
  {
    prox=p->seguinte;
    free((void *) p);
    p=prox;
  }
}

void prn_Tarefa(tar)
/*
- imprime a informacao relativa a uma Tarefa
*/
  ap_Tarefa tar;
{
  char s1[15];

  if (tar!=NULL)
  {
    sprintf(s, "Tarefa %u\tid_agv=", tar->id_tar);
    prn_string(s);

    if (tar->agv_tarefa!=NULL) sprintf(s, "%u", tar->agv_tarefa-
>id_agv);
     else sprintf(s, "-");
    prn_string(s);
    escr_str_tipo_tarefa(s1, tar->tipo_tar);
    sprintf(s, "\n tipo_tar=%s, prior=%u, ", s1, tar->prior);
    prn_string(s);

    switch (tar->tipo_tar)
    {
      case NORMAL: case NORMAL_AGV:
    sprintf(s, "origem=%u, destino=%u", tar->no1, tar->no2); break;
      case POSICION: case T_CARGA: case CARGA_AGV: case POSIC_ESPEC:
    sprintf(s, "ponto=%u", tar->no1); break;
      case T_DESCARGA: sprintf(s, "ponto=%u", tar->no2);
    }
    prn_string(s);

    escr_str_est_tarefa(s1, tar->estado_tar);
    sprintf(s, "\n estado_tar=%s\n", s1);
    prn_string(s);
  }
  else prn_string("(NULL)\n");
}

void prn_l_Tarefas(p)
/*
- imprime o conteudo de uma lista de Tarefas, percorrendo-a nc sentido
cabeca->cauda
*/
  ap_Tarefa p;
```

```
{
  while (p!=NULL)
  {{
     prn_Tarefa(p);
     p=p->seguinte;
  }}
}


/* definicao de outras funcoes utilizadas na RP do SGM */

/*void proc_estado_agv(t, m, p_agvs)*/
/*
- processa uma mensagem do tipo ESTADO_AGV, alterando o estado do
respectivo
AGV
*/
/*   double t;
  mensag_SGM_SGAGV *m;
  ap_AGV p_agvs;
{
  p_agvs=obtem_AGV(m->p1, p_agvs);
  if (p_agvs!=NULL)
  {{
     p_agvs->estado_movim=m->p2;
     p_agvs->estado_alocacao=m->p3;
     p_agvs->estado_carga=m->p4;
     sprintf(s, "\nSGM: alteracao do estado do AGV %u com sucesso apos a
recepcao de:\n", m->p1);
  }}
  else sprintf(s, "\nSGM: AGV %u inexistente apos a recepcao de:\n", m-
>p1);
  prn_string(s);
  prn_m_SGM_SGAGV(t, *m);
}*//

void proc_estado_tarefa(t, m, tar, p_agvs)
/*
- processa uma mensagem do tipo ESTADO_TAREFA, alterando o estado da
respectiva
tarefa e o AGV afectado a mesma
*/
  double t;
  mensag_SGM_SGAGV *m;
  ap_Tarefa tar;
  ap_AGV p_agvs;
{
  char s1[12];

  tar->estado_tar=m->p2;

  if (tar->agv_tarefa!=NULL && tar->agv_tarefa->id_agv!=m->p3 &&
     tar->agv_tarefa->tarefa_agv==tar)
     tar->agv_tarefa->tarefa_agv=NULL;

  if (tar->agv_tarefa==NULL || tar->agv_tarefa->id_agv!=m->p3)
  {
     tar->agv_tarefa=obtem_AGV(m->p3, p_agvs);
     if (tar->agv_tarefa!=NULL) tar->agv_tarefa->tarefa_agv=tar;
```

```
  }
  escr_str_est_tarefa(s1, tar->estado_tar);
  sprintf(s, "\nSGM: Tarefa %u passa ao estado %s, apos a recepcao
de:\n", m->p1, s1);
  prn_string(s);
  prn_m_SGM_SGAGV(t, *m);
}

ap_Tarefa proc_t_posic_espec(t, m, p_tars, u_tars, p_agvs)
/*
- processa uma mensagem do tipo T_POSIC_ESPEC, inserindo a nova tarefa
na fila
*/
  double t;
  mensag_SGM_SGAGV *m;
  ap_Tarefa *p_tars;
  ap_Tarefa *u_tars;
  ap_AGV p_agvs;
{
  Tarefa tar;

  tar.id_tar=m->p1;
  tar.tipo_tar=POSIC_ESPEC;
  tar.prior=0;
  tar.no1=m->p3;
  tar.estado_tar=PASSO1;
  tar.agv_tarefa=obtem_AGV(m->p2, p_agvs);

  if (insere_Tarefa(&tar, p_tars, u_tars))
  {
    if (tar.agv_tarefa!=NULL) tar.agv_tarefa->tarefa_agv=*u_tars;
    sprintf(s, "\nSGM: insercao da tarefa %u na fila apos a recepcao
de:\n", m->p1);
    prn_string(s);
    prn_m_SGM_SGAGV(t, *m);
    return (*u_tars);
  }
  else
  {
    sprintf(s, "\nSGM: erro na insercao da tarefa %u na fila apos a
recepcao de:\n", m->p1);
    prn_string(s);
    prn_m_SGM_SGAGV(t, *m);
    return NULL;
  }
}

double calc_carga(n_tars, n_agvs_activos)
/*
- determina o valor da carga do sistema
- a carga do sistema e definida como sendo o quociente entre o numero de
tarefas
pendentes e o numero de AGV's num estado diferente de {DISABLED,
ENABLED},
admitindo um denominador diferente de zero
- se o denominador for igual a zero, e retornado o valor MAX_DOUBLE
(valor muito elevado)
*/
  ulint n_tars;
```

```
  ulint n_agvs_activos;
{
  if (n_agvs_activos>0) return( ((double) n_tars)/((double)
n_agvs_activos) );
    else return(MAX_DOUBLE);
}

int existe_AGV_nova_tarefa(p_agvs, p_esc_sgm)
/*
- retorna 1 se houver 1 AGV passivel de ser afectado a uma nova tarefa,
ou 0
se for decidido que o escalonamento nao deve ser feito ao nivel do SGM
ou se nao
for encontrado nenhum AGV disponivel (PARADO/ALOCADO ou
PARADO/CARREGADO)
*/
  ap_AGV p_agvs;
  double p_esc_sgm;
{
  if (rnd_binomial(p_esc_sgm))
  {
    while (p_agvs!=NULL && (p_agvs->estado_movim!=PARADO ||
    p_agvs->tarefa_agv!=NULL ||
    ((p_agvs->estado_carga!=CARREGADO && p_agvs-
>estado_alocacao!=ALOCADO)))
          p_agvs=p_agvs->seguinte;

    if (p_agvs!=NULL) return 1;
  }
  return 0;
}

ap_AGV deter_AGV_nova_tarefa(p_agvs)
/*
- retorna o AGV a afectar a uma nova tarefa
*/
  ap_AGV p_agvs;
{
  ap_AGV p_agv_aloc=NULL; /* apontador para um AGV PARADO/ALOCADO */

  while (p_agvs!=NULL &&
    ((p_agvs->estado_movim!=PARADO || p_agvs->tarefa_agv!=NULL ||
          p_agvs->estado_carga!=CARREGADO))
  {
    if (p_agvs->estado_movim==PARADO && p_agvs->tarefa_agv==NULL &&
    p_agvs->estado_alocacao==ALOCADO)
    p_agv_aloc=p_agvs;
    p_agvs=p_agvs->seguinte;
  }
  if (p_agvs!=NULL) return p_agvs;
    else return p_agv_aloc;
}

ulint deter_tipo_tarefa(agv_tarefa, p_tar_carga, p_posicion,
p_tar_carga_agv)
/*
- determina o tipo de uma nova tarefa, tomando como base se a tarefa e
escalonada
pelo SGM ou nao, e as probabilidades definidas para o efeito
```

```
*/
  ap_AGV agv_tarefa;
  double p_tar_carga;
  double p_posicion;
  double p_tar_carga_agv;
{
  if (agv_tarefa==NULL)
  {
    if (rnd_binomial(p_tar_carga)) return T_CARGA;
    else return NORMAL;
  }
  else
  {
    if (rnd_binomial(p_posicion)) return POSICION;
    else if (agv_tarefa->estado_carga==CARREGADO) return T_DESCARGA;
     else if (rnd_binomial(p_tar_carga_agv)) return CARGA_AGV;
           else return NORMAL_AGV;
  }
}

ulint deter_prior_tar(p_tar_prior, prior_tar_max)
/*
- determina a prioridade de uma nova tarefa do tipo NORMAL ou T_CARGA
- se a tarefa for considerada prioritaria (com base na probabilidade
p_tar_prior)
a prioridade valera entre 1 e prior_tar_max, senao valera 0
*/
  double p_tar_prior;
  ulint prior_tar_max;
{
  if (rnd_binomial(p_tar_prior))
    return ( ( (ulint) (random()*((double) prior_tar_max)) )+1);
  else return 0;
}

ulint decide_no_tarefa(nos_tarefas)
/*
- determina um no do layout associado a uma tarefa de Posicionamento com
base na probabilidade associada a cada no para onde possa ocorrer o mov.
*/
  ap_no_l_uid nos_tarefas;
{
  double som_p_ant=0.0, p=random(), p_no;
  ap_no_l_uid anterior=NULL;

  while (nos_tarefas!=NULL)
  {
    p_no=nos_tarefas->d;
    if ((som_p_ant+p_no)>=p && p_no>TOLER_IGUAL_DOUBLES)
    return nos_tarefas->i;
    som_p_ant+=p_no;
    if (p_no>TOLER_IGUAL_DOUBLES) anterior=nos_tarefas;
    nos_tarefas=nos_tarefas->seguinte;
  }

  if (anterior!=NULL) return anterior->i;
    else return 0;
}
```

```
ulint decide_tipo_cancel_tar(p_1_tar, p_todas_exec)
/*
- decide o tipo de cancelamento de tarefas a efectuar com base nas
probabilidades
correspondentes a UMA_TAREFA, TODAS_TAR_EXEC e TODAS_TAR
*/
  double p_1_tar;
  double p_todas_exec;
{
  double p=random();

  if (p<p_1_tar) return UMA_TAREFA;
    else if (p<(p_1_tar+p_todas_exec)) return TODAS_TAR_EXEC;
      else return TODAS_TAR;
}

ulint decide_tar_a_cancelar(p_tars, n_tars)
/*
- escolhe aleatoriamente uma tarefa a cancelar dentre as primeiras
n_tars
- retorna 0 se a tarefa escolhida nao for encontrada
*/
  apo_Tarefa p_tars;
  ullint n_tars;
{
  ullint i=( (ulint) random()*((double) n_tars) )+1;
  ullint j=1;

  while (p_tars!=NULL && j++<i) p_tars=p_tars->seguinte;

  iff (p_tars!=NULL) return p_tars->id_tar; else return 0;
}
```

A.2.13 sgagv.c

```
/* sgagv.c */

#include <stdio.h>
#include <string.h>
#include "mens_log.h"
#include "listas.h"
#include "layout.h"
#include "fun_aux.h"
#include "sgm_sga.h"
#include "sgagv.h"
#include "sistagvs.h"


char s[255]; /* buffer utilizado para chamar 'prn_string()' */


/* definicao de funcoes para manipulacao de listas de AGV's */


int insere_AGV_DISABLED(id_agv, h, t)
/*
- insere um novo AGV DISABLED na cauda da lista, actualizando *h e *t
- retorna 1 em caso de sucesso ou 0 em caso de erro
```

```
*/
  ulint id_agv;
  ap_AGV *h;
  ap_AGV *t;
{
  ap_AGV novo;

  novo=(ap_AGV) malloc(sizeof(AGV));
  if (novo!=NULL)
  {
    novo->id_agv=id_agv;
    novo->posicao=NULL;
    novo->ult_posicao=NULL;
    novo->orient=0;
    novo->estado_movim=DISABLED;
    novo->est_mov_ant=DISABLED;
    novo->tarefa_agv=NULL;
    novo->num_mov_avan=0;
    novo->t_mud_estado=0.0;
    novo->dt_espera=0.0;
    novo->dt_movim_descarr=0.0;
    novo->dt_movim_carr=0.0;
    novo->dt_bloqueado=0.0;
    novo->dt_carga=0.0;
    novo->dt_descarga=0.0;
    novo->dt_erro=0.0;
    novo->dt_enabled=0.0;
    novo->dt_disabled=0.0;
    novo->taxa_util=100.0;
    novo->n_arr_carr=0;
    novo->n_arr_descarr=0;
    novo->n_mov_carr=0;
    novo->n_mov_descarr=0;
    novo->n_cargas=0;
    novo->n_descargas=0;

    novo->seguinte=NULL;
    if (*h==NULL) *h=novo;
    if (*t!=NULL) (*t)->seguinte=novo;
    *t=novo;
    return 1;
  }
  else
  {
    prn_string("\"insere_AGV_DISABLED()\": erro de alocacao de
memoria\n");
    return 0;
  }
}

void prn_el_l_AGVs(agv)
/*
- imprime no stdout a informacao relativa a um AGV
*/
  ap_AGV agv;
{
  char s1[15];

  if (agv!=NULL)
```

```
  {
    sprintf(s, "AGV %u, estado=(", agv->id_agv);
    prn_string(s);
    escr_str_est_movim_agv(s1, agv->estado_movim);
    sprintf(s, "%s,", s1);
    prn_string(s);
    escr_str_est_aloc_agv(s1, agv->estado_movim, agv->estado_alocacao);
    sprintf(s, "%s,", s1);
    prn_string(s);
    escr_str_est_carga_agv(s1, agv->estado_movim, agv->estado_carga);
    sprintf(s, "%s), est_mov_ant=", s1);
    prn_string(s);
    escr_str_est_movim_agv(s1, agv->est_mov_ant);
    sprintf(s, "%s\n posicao=", s1);
    prn_string(s);
    if (agv->posicao!=NULL) sprintf(s, "%u", agv->posicao->id_no);
     else sprintf(s, " -");
    prn_string(s);
    prn_string(", ult_posicao=");
    if (agv->ult_posicao!=NULL) sprintf(s, "%u", agv->ult_posicao-
>id_no);
     else sprintf(s, " -");
    prn_string(s);
    sprintf(s, "\n orient=%d, tarefa_agv=", agv->orient);
    prn_string(s);
    if (agv->tarefa_agv!=NULL) sprintf(s, "%u", agv->tarefa_agv-
>id_tar);
     else sprintf(s, " -");
    prn_string(s);
    sprintf(s, ", num_mov_avan=%u, t_mud_estado=%.3f, taxa_util=%.3f\n
",
     agv->num_mov_avan, agv->t_mud_estado, agv->taxa_util);
    prn_string(s);

prn_string("(dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_c
arr,dt_descarga)=\n");
    sprintf(s, " (%.3f, %.3f, %.3f, %.3f, %.3f, %.3f)\n ", agv-
>dt_espera, agv->dt_bloqueado,
     agv->dt_movim_descarr, agv->dt_carga, agv->dt_movim_carr, agv-
>dt_descarga);
    prn_string(s);

prn_string("(dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=\n");
    sprintf(s, " (%.3f, %.3f, %.3f, %u, %u)\n ", agv->dt_disabled, agv-
>dt_enabled,
     agv->dt_erro, agv->n_cargas, agv->n_descargas);
    prn_string(s);
    prn_string("
(n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=\n");
    sprintf(s, " (%u, %u, %u, %u)\n", agv->n_arr_carr, agv-
>n_arr_descarr,
     agv->n_mov_carr, agv->n_mov_descarr);
    prn_string(s);
  }
  else prn_string("(NULL)\n");
}

void prn_AGVs(h)
/*
```

```
- imprime no stdout os elementos de uma lista de AGV's
*/
  ap_AGV h;
{
  prn_string("\n");
  while (h!=NULL)
  {
    prn_el_l_AGVs(h);
    h=h->seguinte;
  }
}

void limpa_lista_AGVs(h)
/*
- limpa toda a lista, colocando *h=NULL
*/
  ap_AGV *h;
{
  ap_AGV p;

  while (*h!=NULL)
  {
    p=(*h)->seguinte;
    free(*h);
    *h=p;
  }
}

ap_AGV obtem_ap_AGV(id_agv, h)
/*
- retorna um apontador para um AGV da lista de AGV's cujo identif e
id_agv
*/
  ulint id_agv;
  ap_AGV h;
{
  while (h!=NULL)
  {
    if (h->id_agv==id_agv) return h;
    h=h->seguinte;
  }
  return h;
}



/* definicao de funcoes para manipulacao de listas de Bloqueios de No´s
do Layout */

int insere_bloqueio(no, agv, h, t)
/*
- insere um novo bloqueio na cauda da lista, actualizando *h e *t
- nao e admitido mais do que um bloqueio por no
- retorna 1 em caso de sucesso ou 0 em caso de erro
*/
  ap_no_l_nos no;
  ap_AGV agv;
  ap_bloqueio *h;
  ap_bloqueio *t;
```

```
{
  ap_bloqueio novo, bloq;

  if ((bloq=obtem_bloqueio(no->id_no, *h))==NULL)
  {
    novo=(ap_bloqueio) malloc(sizeof(bloqueio));
    if (novo!=NULL)
    {
      novo->no=no;
      novo->agv=agv;

      novo->seguinte=NULL;
      if (*h==NULL) *h=novo;
      if (*t!=NULL) (*t)->seguinte=novo;
      *t=novo;
      return 1;
    }
    else
    {
      prn_string("\"insere_bloqueio()\": erro de alocacao de
memoria\n");
      return 0;
    }
  }
  else
  {
    if (agv!=bloq->agv)
    {
      sprintf(s, "\"insere_bloqueio()\": bloqueio (id_no,id_agv)=(%u,%u)
nao inserido\n porque ja existe o bloqueio (id_no,id_agv)=(%u,%u)\n",
      no->id_no, agv->id_agv, bloq->no->id_no, bloq->agv->id_agv);
      prn_string(s);
      return 1;
    }
    return 0;
  }
}

void insere_bloqueios(no, agv, h, t, layout)
/*
- insere os bloqueios necessarios para que o no fique reservado para o
AGV,,
sendo bloqueado o proprio no e os nos que constem na sua lista de
ocupacao
de nos
- os bloqueios sao inseridos na cauda da lista, sendo actualizados
os apontadores *h e *t
*/
  ap_no_l_nos no;
  ap_AGV agv;
  ap_bloqueio *h;
  ap_bloqueio *t;
  ap_no_l_nos layout;
{
  ap_no_l_ulint no_oc;

  insere_bloqueio(no, agv, h, t);
  no_oc=no->ocupacao_nos;
```

```
  while (no_oc!=NULL)
  {
    if (no_oc->i!=no->id_no) insere_bloqueio(get_no(layout, no_oc->i),
agv, h, t);
    no_oc=no_oc->seguinte;
  }
}

void insere_bloqs_movim(agv, no_origem, id_no_destino, layout, h, t)
/*
- reserva todos os nos necessarios a execucao de um movimento
*/
  ap_AGV agv;
  ap_no_l_nos no_origem;
  ulint id_no_destino;
  ap_no_l_nos layout;
  ap_bloqueio *h;

  ap_bloqueio *t;
{
  ap_no_l_ramos ramo;
  ap_no_l_ulint bloq_ramo;
  ap_no_l_nos no_bloq;

  if (no_origem!=NULL)
  {
    ramo=get_ramo(no_origem, id_no_destino);
    if (ramo!=NULL)
    {
      bloq_ramo=ramo->bloqueios;
      while (bloq_ramo!=NULL)
      {
    no_bloq=get_no(layout, bloq_ramo->i);
    insere_bloqueio(no_bloq , agv, h, t);
    bloq_ramo=bloq_ramo->seguinte;
      }
    }
  }
}

ap_bloqueio obtem_bloqueio(id_no, h)
/*
- retorna um apontador para um bloqueio a partir do no bloqueado
*/
  ulint id_no;
  ap_bloqueio h;
{
  while (h!=NULL)
  {
    if (h->no->id_no==id_no) return h;
    h=h->seguinte;
  }
  return h;
}

ap_bloqueio bloq_no_indisponivel(no, agv, h)
/*
- retorna NULL se o no estiver bloqueado pelo agv ou se nao estiver
bloqueado por
```

```
qualquer AGV
- ou retorna o 1. bloqueio que impeca que o no esteja disponivel para o
AGV
- e verificado o proprio no, bem como todos os nos que facam parte da
sua lista
ocupacao de nos
*/
  ap_no_l_nos no;
  ap_AGV agv;
  ap_bloqueio h;
{
  ap_bloqueio bloq;
  ap_no_l_ulint no_oc;

  bloq=obtem_bloqueio(no->id_no, h);
  if (bloq!=NULL && bloq->agv!=agv) return bloq;
  else
  {
    no_oc=no->ocupacao_nos;
    while (no_oc!=NULL)
    {
      if (no_oc->i!=no->id_no)
      {
    bloq=obtem_bloqueio(no_oc->i, h);
    if (bloq!=NULL && bloq->agv!=agv) return bloq;
      }
      no_oc=no_oc->seguinte;
    }
    return NULL;
  }
}

ap_bloqueio bloq_movim_inviavel(agv, no_origem, id_no_destino, est_lay)
/*
- retorna o 1. bloqueio que torna inviavel a execucao do proximo
movimento do AGV, isto e, que bloqueie um no que faca parte da lista
de bloqueios do ramo associado ao movimento
- retorna NULL se nao for encontrado nenhum
*/
  ap_AGV agv;
  ap_no_l_nos no_origem;
  ullint id_no_destino;
  ap_bloqueio est_lay;
{
  ap_no_l_ramos ramo;
  ap_no_l_ulint bloq_ramo;
  ap_bloqueio bloq;

  ramo=get_ramo(no_origem, id_no_destino);
  if (ramo!=NULL)
  {
    bloq_ramo=ramo->bloqueios;
    while (bloq_ramo!=NULL)
    {
    if ((bloq=obtem_bloqueio(bloq_ramo->i, est_lay))!=NULL &&
        bloq->agv!=agv) return bloq;
    bloq_ramo=bloq_ramo->seguinte;
    }
  }
```

```
  return NULL;
}

/*void remove_bloqueios_agv(agv, todos, h, t)

- se todos valer VERDADEIRO, remove todos os bloqueios referentes a um
AGV
- se todos valer FALSO remove todos os bloqueios referentes a um AGV,
excepto aqueles
que digam respeito a nos que pertencam a lista de ocupacao de nos do no
correspondente
a sua posicao actual
- actualiza *h e *t, caso isso seja necessario

  ap_AGV agv;
  char todos;
  ap_bloqueio *h;
  ap_bloqueio *t;
{
  ap_bloqueio p=*h, a=NULL, aux;

  while (p!=NULL)
  {
    if (p->agv==agv &&
    (todos==VERDADEIRO || p->agv->posicao==NULL ||
     (p->agv->posicao!=NULL && p->no!=p->agv->posicao &&
      procura_ulint(p->no->id_no, p->agv->posicao->ocupacao_nos)==NULL)
    )
       )
    {
      if (a!=NULL) a->seguinte=p->seguinte;
    else *h=p->seguinte;
      if (p==*t) *t=a;
      aux=p->seguinte;
      free(p);
      p=aux;
    }
    else { a=p; p=p->seguinte; }
  }
}*/

void remove_bloqueios_agv(agv, h, t)
/*
- remove todos os bloqueios de um AGV que ja nao sejam necessarios
- actualiza *h e *t
*/
  ap_AGV agv;
  ap_bloqueio *h;
  ap_bloqueio *t;
{
  ap_no_l_ramos ramo;
  ap_no_l_ulint p_bloqs_necess=NULL, u_bloqs_necess=NULL, no_oc,
bloq_ramo;
  ap_no_l_apno no_dest;
  ap_no_l_nos no_orig;
  ulint i=0;
  ap_bloqueio p=*h, a=NULL, aux;

  if (agv->estado_movim!=DISABLED && agv->estado_movim!=ENABLED)
```

```
  {
    insere_ulint(agv->posicao->id_no, &p_bloqs_necess, &u_bloqs_necess,
0);
    no_oc=agv->posicao->ocupacao_nos;
    while (no_oc!=NULL)
    {
      if (no_oc->i!=agv->posicao->id_no)
    insere_ulint(no_oc->i, &p_bloqs_necess, &u_bloqs_necess, 0);
      no_oc=no_oc->seguinte;
    }

    if (agv->estado_movim==EX_MOVIM) /* pressupoe-se que agv-
>numm_mov_avan > 0 */
    {
      if (agv->tarefa_agv->estado_tar==PASSO1) no_dest=agv->tarefa_agv-
>traaj1;
    else no_dest=agv->tarefa_agv->traj2;
      while (i++<agv->num_mov_avan)
      {
    if (no_dest->anterior!=NULL) no_orig=no_dest->anterior->no;
          else no_orig=agv->posicao;
    ramo=get_ramo(no_orig, no_dest->no->id_no);
    bloq_ramo=ramo->bloqueios;
    while (bloq_ramo!=NULL)
    {
      insere_ulint(bloq_ramo->i, &p_bloqs_necess, &u_bloqs_necess, 0);
      bloq_ramo=bloq_ramo->seguinte;
    }
    no_dest=no_dest->seguinte;
      }
    }
  }

  wrhile (p!=NULL)
  {
    if (p->agv==agv &&
    procura_ulint(p->no->id_no, p_bloqs_necess)==NULL
       )
    {
      if (a!=NULL) a->seguinte=p->seguinte;
    else *h=p->seguinte;
      if (p==*t) *t=a;
      aux=p->seguinte;
      free(p);
      p=aux;
    }
    else { a=p; p=p->seguinte; }
  }

  liimpa_l_ulint(p_bloqs_necess);
}

voicd limpa_l_bloqueios(h)
/*
- liimpa toda a lista, colocando *h=NULL
*/
  ap_bloqueio *h;
{
  ap_bloqueio p;
```

```
  while (*h!=NULL)
  {
    p=(*h)->seguinte;
    free(*h);
    *h=p;
  }
}

void prn_l_bloqueios(h)
/*
- imprime no stdout o conteudo da lista de bloqueios
*/
  ap_bloqueio h;
{
  ap_bloqueio p=h;

  prn_string("(id_no,id_agv)={");
  while (p!=NULL)
  {
    if (p!=h) prn_string(",");
    sprintf(s, "(%u,%u)", p->no->id_no, p->agv->id_agv);
    prn_string(s);
    p=p->seguinte;
  }
  prn_string("}\n");
}



/* definicao de funcoes para manipulacao de listas de Tarefas */

void inicial_tarefa(tar, id_tar, tipo_tar, prior, no1, no2, t)
/*
- inicializa os campos de uma nova Tarefa
*/
  ap_Tarefa tar;
  ulint id_tar;
  ulint tipo_tar;
  ulint prior;
  ap_no_l_nos no1;
  ap_no_l_nos no2;
  double t;
{
  tar->id_tar=id_tar;
  tar->tipo_tar=tipo_tar;
  tar->prior=prior;
  if (tipo_tar!=T_DESCARGA) tar->no1=no1; else tar->no1=NULL;
  if (tipo_tar==NORMAL || tipo_tar==NORMAL_AGV || tipo_tar==T_DESCARGA)
    tar->no2=no2;
   else tar->no2=NULL;
  tar->estado_tar=FILA;
  tar->est_antes_desp=FILA;
  tar->agv_tarefa=NULL;
  tar->ctraj1=MAXINT;
  tar->traj1=NULL;
  tar->ctraj2=MAXINT;
  tar->traj2=NULL;
  tar->t_mud_estado=t;
```

BIBLIOTECA F.E. PORTO

```
  tar->t_lancam=t;
  tar->dt_fila=0.0;
  tar->dt_p1=0.0;
  tar->dt_p2=0.0;
  tar->dt_p3=0.0;
  tar->dt_p4=0.0;
  tar->num_af=0;
}

int insere_tarefa(tar, h, t)
/*
- insere uma nova tarefa na cauda da lista, actualizando *h e *t
- retorna 1 em caso de sucesso ou 0 em caso de erro
*/
  ap_Tarefa tar;
  ap_Tarefa *h;
  ap_Tarefa *t;
{
  ap_Tarefa novo;

  novo=(ap_Tarefa) malloc(sizeof(Tarefa));
  if (novo!=NULL)
  {
    novo->id_tar=tar->id_tar;
    novo->tipo_tar=tar->tipo_tar;
    novo->prior=tar->prior;
    novo->no1=tar->no1;
    novo->no2=tar->no2;
    novo->estado_tar=tar->estado_tar;
    novo->est_antes_desp=tar->est_antes_desp;
    novo->agv_tarefa=tar->agv_tarefa;
    novo->ctraj1=tar->ctraj1;
    novo->traj1=tar->traj1;
    novo->ctraj2=tar->ctraj2;
    novo->traj2=tar->traj2;
    novo->t_mud_estado=tar->t_mud_estado;
    novo->t_lancam=tar->t_lancam;
    novo->dt_fila=tar->dt_fila;
    novo->dt_p1=tar->dt_p1;
    novo->dt_p2=tar->dt_p2;
    novo->dt_p3=tar->dt_p3;
    novo->dt_p4=tar->dt_p4;
    novo->num_af=tar->num_af;

    novo->seguinte=NULL;
    if (*h==NULL) *h=novo;
    if (*t!=NULL) (*t)->seguinte=novo;
    *t=novo;
    return 1;
  }
  else
  {
    prn_string("\"insere_tarefa()\": erro de alocacao de memoria\n");
    return 0;
  }
}

void prn_el_l_tarefas(tar)
/*
```

```
- imprime no stdout a informacao relativa a uma Tarefa
*/
  ap_Tarefa tar;
{
  char s1[15];

  if (tar!=NULL)
  {
    escr_str_tipo_tarefa(s1, tar->tipo_tar);
    sprintf(s, "Tarefa %u, tipo=%s, prior=%u, estado=", tar->id_tar, s1,
tar->prior);
    prn_string(s);
    escr_str_est_tarefa(s1, tar->estado_tar);
    sprintf(s, "%s, est_antes_desp=", s1);
    prn_string(s);
    escr_str_est_tarefa(s1, tar->est_antes_desp);
    sprintf(s, "%s\n", s1);
    prn_string(s);
    switch (tar->tipo_tar)
    {
      case POSICION: case T_CARGA: case CARGA_AGV: case POSIC_ESPEC:
    sprintf(s, " ponto=%u", tar->no1->id_no); break;
      case T_DESCARGA:
    sprintf(s, " ponto=%u", tar->no2->id_no); break;
      default: sprintf(s, " origem=%u, destino=%u", tar->no1->id_no,
tar->no2->id_no);
    }
    prn_string(s);
    if (tar->agv_tarefa!=NULL) sprintf(s, ", AGV=%u", tar->agv_tarefa-
>id_agv);
    else sprintf(s, ", AGV=-");
    prn_string(s);
    sprintf(s, "\n ctraj1=%u, traj1=", tar->ctraj1);
    prn_string(s);
    prn_l_apno(tar->traj1);
    sprintf(s, " ctraj2=%u, traj2=", tar->ctraj2);
    prn_string(s);
    prn_l_apno(tar->traj2);
    prn_string(" (t_lancam, t_mud_estado, dt_fila, dt_p1, dt_p2, dt_p3,
dt_p4, num_af)=");
    sprintf(s, "\n  (%.3f, %.3f, %.3f, %.3f, %.3f, %.3f, %.3f, %u)\n",
tar->t_lancam, tar->t_mud_estado,
    tar->dt_fila, tar->dt_p1, tar->dt_p2, tar->dt_p3, tar->dt_p4, tar-
>num_af);
    prn_string(s);
  }
  else prn_string("(NULL)\n");
}

void prn_tarefas(h)
/*
- imprime no stdout os elementos de uma lista de Tarefas
*/
  ap_Tarefa h;
{
  prn_string("\n");
  while (h!=NULL)
  {
    prn_el_l_tarefas(h);
```

```
    h=h->seguinte;
  }
}

void limpa_lista_tarefas(h)
/*
- limpa toda a lista, colocando *h=NULL
*/
  ap_Tarefa *h;
{
  ap_Tarefa p;

  while (*h!=NULL)
  {
    p=(*h)->seguinte;
    free(*h);
    *h=p;
  }
}

ap_Tarefa obtem_ap_tarefa(id_tar, h)
/*
- retorna um apontador para uma Tarefa da lista de Tarefas cujo identif
e id_tar
*/
  unsigned int id_tar;
  ap_Tarefa h;
{
  while (h!=NULL)
  {
    if (h->id_tar==id_tar) return h;
    h=h->seguinte;
  }
  return h;
}

void remove_tarefa(id_tar, h, t)
/*
- remove todas as tarefas com identif. = id_tar
- actualiza *h e *t, caso isso seja necessario
*/
  unsigned int id_tar;
  ap_Tarefa *h;
  ap_Tarefa *t;
{
  ap_Tarefa p=*h, a=NULL, aux;

  while (p!=NULL)
  {
    if (p->id_tar==id_tar)
    {
      limpa_l_apno(&p->traj1);
      limpa_l_apno(&p->traj2);

      if (a!=NULL) a->seguinte=p->seguinte;
    else *h=p->seguinte;
      if (p==*t) *t=a;
      aux=p->seguinte;
      free(p);
```

```
      p=aux;
    }
    else { a=p; p=p->seguinte; }
  }
}



/* definicao de funcoes para manipulacao de listas representando AGV's
bloqueados */

int insere_AGV_bloq(agv, agv_bloa, ramo, p, u)
/*
- insere um AGV bloqueado na cauda da lista, actualizando *p e *u
- retorna 1 em caso de sucesso ou 0 em caso de erro
*/
  ap_AGV agv;
  ap_AGV agv_bloq;
  ap_no_l_ramos ramo;
  ap_AGV_bloqueado *p;
  ap_AGV_bloqueado *u;
{
  ap_AGV_bloqueado novo;

  novo=(ap_AGV_bloqueado) malloc(sizeof(AGV_bloqueado));
  if (novo!=NULL)
  {
    novo->agv=agv;
    novo->agv_bloq=agv_bloq;
    novo->ramo=ramo;
    novo->custo_adicional=MAXINT;

    novo->seguinte=NULL;
    if (*p==NULL) *p=novo;
    if (*u!=NULL) (*u)->seguinte=novo;
    *u=novo;
    return 1;
  }
  else
  {
    prn_string("\"insere_AGV_bloq()\": erro de alocacao de memoria\n");
    return 0;
  }
}

ap_AGV_bloqueado obtem_AGV_bloq(agv, p)
/*
- retorna um apontador para um AGV bloqueado da lista, a partir do campo
'agv'
*/
  ap_AGV agv;
  ap_AGV_bloqueado p;
{
  while (p!=NULL)
  {
    if (p->agv==agv) return p;
    p=p->seguinte;
  }
```

```
  return p;
}

void limpa_l_AGVs_bloq(p)
/*
- limpa toda a lista, colocando *p=NULL
*/
  ap_AGV_bloqueado *p;
{
  ap_AGV_bloqueado a;

  while (*p!=NULL)
  {
    a=(*p)->seguinte;
    free(*p);
    *p=a;
  }
}

void prn_l_AGVs_bloq(p)
/*
- imprime no stdout os elementos de uma lista de Tarefas
*/
  ap_AGV_bloqueado p;
{
  ap_AGV_bloqueado a=p;

  while (a!=NULL)
  {
    if (a!=p) prn_string("->");
    sprintf(s, "(%u,%u)", a->agv->id_agv, a->agv_bloq->id_agv);
    prn_string(s);
    a=a->seguinte;
  }
  prn_string("\n");
}


/* definicao de funcoes para manipulacao de listas de elem. tipo
EstadoFila */

ap_EstadoFila procura_EstadoFila(id, primeiro)
/*
- retorna o 1. elemento encontrado que satisfaca id, ou NULL
se nao for encontrado nenhum
- primeiro e cabeca da lista ligada
*/
 ulint id;
 ap_EstadoFila primeiro;
{
 while (primeiro!=NULL && primeiro->id!=id)
    primeiro=primeiro->seguinte;
 return primeiro;
}

int insere_EstadoFila(id, Cf, primeiro, ultimo)
/*
- insere o elemento (id,Cf,0,0,NULL,0.0) no fim da lista
```

```
- primeiro e a cabeca da lista
- ultimo e o ultimo elemento inserido na lista (ignorado se valer NULL)
- retorna 1 em caso de sucesso ou 0 em caso de insucesso
*/
  ulint id;
  ulint Cf;
  ap_EstadoFila *primeiro;
  ap_EstadoFila *ultimo;
{
  ap_EstadoFila novo;

  novo= (ap_EstadoFila) malloc(sizeof(EstadoFila));
  if (novo!=NULL)
  {
    novo->id=id;
    novo->Cf=Cf;
    novo->Lf=0;
    novo->Rf=0;
    novo->tar_mais_ant=NULL;
    novo->t_MFCFS=0.0;
    novo->seguinte=NULL;
    if (*primeiro==NULL) *primeiro=novo;
    if (*ultimo!=NULL) (*ultimo)->seguinte=novo;
    *ultimo=novo;
    return 1;
  }
  else
  {
   prn_string("\"insere_EstadoFila()\": erro de alocacao de memoria\n");
   return 0;
  }
}

void limpa_l_EstadoFila(h)
/*
- elimina os elementos de uma lista
- *h e a cabeca da lista
- *h fica a valer NULL a saida da funcao
*/
 ap_EstadoFila *h;
{
 ap_EstadoFila proximo;

 while (*h!=NULL)
 {
  proximo=(*h)->seguinte;
  free((void *)*h);
  *h=proximo;
 }
}

void prn_EstadoFila(p)
/*
- mostra no stdout a informacao armazenada numa estrutura do tipo
EstadoFila
*/
  ap_EstadoFila p;
{
  sprintf(s, "Fila %2u: Cf=%2u, Lf=%2u, Rf=%2u, tar_mais_ant=",
```

```
    p->id, p->Cf, p->Lf, p->Rf);
  prn_string(s);
  if (p->tar_mais_ant==NULL) sprintf(s, "(NULL)");
    else sprintf(s, "%u", p->tar_mais_ant->id_tar);
  prn_string(s);
  sprintf(s, ", t_MFCMS=%.3f\n", p->t_MFCFS);
  prn_string(s);
}

void prn_l_EstadoFila(h)
/*
- mostra no stdout o conteudo da lista ligada cuja cabeca e h
*/
  ap_EstadoFila h;
{
  if (h==NULL) prn_string("(NULL)\n");

  while (h!=NULL)
  {
    prn_EstadoFila(h);
    h=h->seguinte;
  }
}



/* definicao de outras funcoes utilizadas na RP do SGAGV */

void act_estat_temporais_agv(agv, t)
/*
- actualiza as estatisticas temporais do AGV, tomando como o referencia
o instante actual t
*/
  ap_AGV agv;
  double t;
{
  double soma1, soma2;

  switch (agv->estado_movim)
  {
    case DISABLED: agv->dt_disabled+=t-agv->t_mud_estado; break;
    case ENABLED: agv->dt_enabled+=t-agv->t_mud_estado; break;
    case PARADO: agv->dt_espera+=t-agv->t_mud_estado; break;
    case EX_MOVIM:
    if (agv->estado_carga==DESCARREGADO)
          agv->dt_movim_descarr+=t-agv->t_mud_estado;
       else agv->dt_movim_carr+=t-agv->t_mud_estado;
    break;
    case CARGA_AGV: agv->dt_carga+=t-agv->t_mud_estado; break;
    case DESCARGA_AGV: agv->dt_descarga+=t-agv->t_mud_estado; break;
    case BLOQUEADO: agv->dt_bloqueado+=t-agv->t_mud_estado; break;
    case ERRO: agv->dt_erro+=t-agv->t_mud_estado;
  }
  agv->t_mud_estado=t;

  soma1=agv->dt_espera+agv->dt_bloqueado+agv->dt_erro;
  soma2=agv->dt_movim_descarr+agv->dt_carga+agv->dt_movim_carr+agv-
>dt_descarga;
  if (soma1+soma2>0.0)
```

```
    agv->taxa_util=soma2/(soma1+soma2)*100.0;
  else agv->taxa_util=100.0;
}

void act_estat_temporais_tarefa(tar, t)
/*
- actualiza as estatisticas temporais da Tarefa, tomando como o
referencia o instante actual t
*/
  ap_Tarefa tar;
  double t;
{
  switch(tar->estado_tar)
  {
    case FILA: tar->dt_fila+=t-tar->t_mud_estado; break;
    case PASSO1: tar->dt_p1+=t-tar->t_mud_estado; break;
    case PASSO2: tar->dt_p2+=t-tar->t_mud_estado; break;
    case PASSO3: tar->dt_p3+=t-tar->t_mud_estado; break;
    case PASSO4: tar->dt_p4+=t-tar->t_mud_estado;
  }
  tar->t_mud_estado=t;
}

ulint deter_num_tar_cancel(h1, tipo_cancel, id_tar, h2)
/*
- determina o numero de Tarefas cancelaveis, mediante o tipo de
cancelamento pretendido, criando uma lista de apontadores para as
tarefas a
cancelar
- retorna em *h2 o 1. elemento da lista de apontadores para tarefas a
cancelar
*/
  ap_Tarefa h1;
  ulint tipo_cancel;
  ulint id_tar;
  ap_no_l_ap_Tarefas *h2;
{
  ap_no_l_ap_Tarefas u=NULL;
  ulint n=0;

  *h2=NULL;

  while (h1!=NULL)
  {
    if (tipo_cancel==TODAS_TAR ||
    (tipo_cancel==TODAS_TAR_EXEC && h1->estado_tar!=FILA) ||
    (tipo_cancel==UMA_TAREFA && h1->id_tar==id_tar)
       )
    {
    insere_no_l_ap_Tarefas(h1, h2, &u);
    n++;
    if (tipo_cancel==UMA_TAREFA) h1=NULL;
          else h1=h1->seguinte;
    }
    else h1=h1->seguinte;
  }

  return n;
}
```

```
int ha_1_AGV_disponivel(h)
/*
- retorna 1 se houver na lista pelo menos um AGV com
estado=(PARADO, NAO_ALOCADO, DESCARREGADO), ou 0 no caso contrario
*/
  ap_AGV h;
{
  /*prn_debug("ha_1_AGV_disponivel");*/
  while (h!=NULL)
  {
    if (h->estado_movim==PARADO && h->estado_alocacao==NAO_ALOCADO &&
    h->estado_carga==DESCARREGADO) return 1;
    h=h->seguinte;
  }
  return 0;
}

int ha_1_Tar_n_escalon(h)
/*
- retorna 1 se houver na lista pelo menos uma Tarefa do tipo NORMAL ou
T_CARGA e no estado FILA, ou 0 no caso contrario
*/
  ap_Tarefa h;
{
  /*prn_debug("ha_1_Tar_n_escalon");*/
  while (h!=NULL)
  {
    if ((h->tipo_tar==NORMAL || h->tipo_tar==T_CARGA) &&
    h->estado_tar==FILA) return 1;
    h=h->seguinte;
  }
  return 0;
}

ap_AGV agv_PARADO_BLOQUEADO(h)
/*
- retorna um AGV no estado PARADO com condicoes para passar ao estado
BLOQUEADO (afectado a uma tarefa)
- retorna NULL se nao for encontrado nenhum
*/
  ap_AGV h;
{
  /*prn_debug("agv_PARADO_BLOQUEADO");*/
  while (h!=NULL)
  {
    if (h->estado_movim==PARADO && h->tarefa_agv!=NULL)
    return h;
    h=h->seguinte;
  }
  return NULL;
}

void preenche_mens_ESTADO_AGV(m, agv)
/*
- preenche uma mensagem do tipo ESTADO_AGV
*/
  mensag_SGM_SGAGV *m;
  ap_AGV agv;
```

```
{
  m->id_mens=0;
  m->tipo_mens=ESTADO_AGV;
  m->p1=agv->id_agv;
  m->p2=agv->estado_movim;
  m->p3=agv->estado_alocacao;
  m->p4=agv->estado_carga;
}

void preenche_mens_ESTADO_TAREFA(m, tar)
/*
- preenche uma mensagem do tipo ESTADO_TAREFA
*/
  mensag_SGM_SGAGV *m;
  ap_Tarefa tar;
{
  m->id_mens=0;
  m->tipo_mens=ESTADO_TAREFA;
  m->p1=tar->id_tar;
  m->p2=tar->estado_tar;
  if (tar->agv_tarefa!=NULL) m->p3=tar->agv_tarefa->id_agv;
    else m->p3=0;
}

ap_AGV agv_BLOQUEADO_MOVIMENTO(h, est_lay)
/*
- retorna um AGV no estado BLOQUEADO com condicoes para passar ao estado
EX_MOVIM: afectado a uma Tarefa no estado PASSO1 ou PASSO3, com pelo
menos
1 movimento por executar e com todos os nos necessarios disponiveis
- retorna NULL se nao for encontrado nenhum
*/
  ap_AGV h;
  ap_bloqueio est_lay;
{
  /*prn_debug("agv_BLOQUEADO_MOVIMENTO");*/
  while (h!=NULL)
  {
    if (h->estado_movim==BLOQUEADO && h->tarefa_agv!=NULL)
    if ((h->tarefa_agv->estado_tar==PASSO1 && h->tarefa_agv->traj1!=NULL
&&
         bloq_movim_inviavel(h, h->posicao,
         h->tarefa_agv->traj1->no->id_no, est_lay)==NULL) ||
        (h->tarefa_agv->estado_tar==PASSO3 && h->tarefa_agv->traj2!=NULL
&&
         bloq_movim_inviavel(h, h->posicao,
         h->tarefa_agv->traj2->no->id_no, est_lay)==NULL)
       ) return h;
    h=h->seguinte;
  }
  return h;
}

ap_AGV agv_BLOQUEADO_CD(h)
/*
- retorna um AGV no estado BLOQUEADO com condicoes para passar ao estado
CARGA_AGV ou DESCARGA_AGV, ou seja, afectado a uma Tarefa no estado
PASSO2 ou
PASSO4, respectivamente
```

```
- retorna NULL se nao for encontrado nenhum
*/
  ap_AGV h;
{
  /*prn_debug("agv_BLOQUEADO_CD");*/
  while (h!=NULL)
  {
    if (h->estado_movim==BLOQUEADO && h->tarefa_agv!=NULL &&
     (h->tarefa_agv->estado_tar==PASSO2 || h->tarefa_agv-
>estado_tar==PASSO4)
       ) return h;
    h=h->seguinte;
  }
  return h;
}

ap_no_l_apno obtem_no_traj_mov_avan(agv)
/*
- retorna o no do trajecto calculado para um AGV, correspondente ao
proximo movimento em avanco a enviar
- retorna NULL se nao existir
*/
  ap_AGV agv;
{
  uint i=0;
  ap_no_l_apno no_traj=NULL;

  if (agv->tarefa_agv->estado_tar==PASSO1) no_traj=agv->tarefa_agv-
>traj1;
   else if (agv->tarefa_agv->estado_tar==PASSO3) no_traj=agv-
>tarefa_agv->traj2;

  while (no_traj!=NULL && i++<agv->num_mov_avan) no_traj=no_traj-
>seguinte;
  return no_traj;
}

ap_AGV agv_enviar_mov_avan(h, est_lay, num_max_mov_avan)
/*
- retorna um AGV no estado EX_MOVIM com condicoes para receber
movimentos
em avanco
- retorna NULL se nao for encontrado nenhum
*/
  ap_AGV h;
  ap_bloqueio est_lay;
  uint num_max_mov_avan;
{
  ap_no_l_apno no_de_pr_mov;
  ap_no_l_nos no_or_pr_mov;

  /*prn_debug("agv_enviar_mov_avan");*/
  while (h!=NULL)
  {
    if (h->estado_movim==EX_MOVIM && h->tarefa_agv!=NULL &&
    h->num_mov_avan<num_max_mov_avan)
    {
    no_de_pr_mov=obtem_no_traj_mov_avan(h);
    if (no_de_pr_mov!=NULL)
```

```
      {
        if (no_de_pr_mov->anterior!=NULL) no_or_pr_mov=no_de_pr_mov-
>anterior->no;
             else no_or_pr_mov=h->posicao;
        if (bloq_movim_inviavel(h, no_or_pr_mov,
             no_de_pr_mov->no->id_no, est_lay)==NULL) return h;
      }
      }
      h=h->seguinte;
    }
    return h;
}

void proc_tarefa_concluida(tar, t)
/*
- actualiza a informacao relativa a uma Tarefa concluida
*/
    ap_Tarefa tar;
    double t;
{
    char s1[15];
    double t_oper;

    act_estat_temporais_tarefa(tar, t);
    tar->estado_tar=CONCLUIDA;

    escr_str_tipo_tarefa(s1, tar->tipo_tar);
    sprintf(s, "\nSGAGV: Tarefa %u do tipo %s foi concluida com exito pelo
AGV %u.\n",
      tar->id_tar, s1, tar->agv_tarefa->id_agv);
    prn_string(s);
    sprintf(s, "\n Estatisiticas:\n
(dt_fila,dt_p1,dt_p2,dt_p3,dt_p4)=(%.3f,%.3f,%.3f,%.3f,%.3f)\n",
      tar->dt_fila, tar->dt_p1, tar->dt_p2, tar->dt_p3, tar->dt_p4);
    prn_string(s);
    t_oper=tar->dt_p1+tar->dt_p2+tar->dt_p3+tar->dt_p4;
    sprintf(s, "  num_af=%u, Tempo Operacao=%.3f, Tempo Total
Execucao=%.3f\n\n",
      tar->num_af, t_oper, t_oper+tar->dt_fila);
    prn_string(s);

    tar->agv_tarefa->tarefa_agv=NULL;
    tar->agv_tarefa->est_mov_ant=tar->agv_tarefa->estado_movim;
    tar->agv_tarefa->estado_movim=PARADO;
    tar->agv_tarefa=NULL;
}

ap_Tarefa tarefa_traj_nulo(h)
/*
- retorna uma Tarefa no estado PASSO1 ou PASSO3, afectada a um AGV no
estado
BLOQUEADO e com um trajecto nulo para ser executado
- retorna NULL se nao for encontrada nenhuma
*/
    ap_AGV h;
{
    while (h!=NULL)
    {
      if (h->estado_movim==BLOQUEADO &&
```

```
    h->tarefa_agv!=NULL &&
    ((h->tarefa_agv->estado_tar==PASSO1 && h->tarefa_agv->traj1==NULL)
||
     (h->tarefa_agv->estado_tar==PASSO3 && h->tarefa_agv->traj2==NULL)
    )
d      ) return h->tarefa_agv;
    h=h->seguinte;
  }
  return NULL;
}

ap_AGV agv_BLOQUEADO_PARADO(h)
/*
- retorna um AGV no estado BLOQUEADO com condicoes para passar ao estado
PARADO (nao afectado a uma tarefa)
- retorna NULL se nao for encontrado nenhum
*/
  ap_AGV h;
{
  while (h!=NULL)
  {
    if (h->estado_movim==BLOQUEADO && h->tarefa_agv==NULL)
    return h;
    h=h->seguinte;
  }
  return NULL;
}

int ha_agv_bloq_por_agv_PARADO(h, layout, est_lay, num_max_mov_avan)
/*
- verifica se existe um AGV que obedeca as seguintes condicoes:
    -estado_movim==BLOQUEADO ou EX_MOVIM
    -AGV bloqueante no estado PARADO
    -AGV bloqueante sem tarefa
- retorna 1 se existir ou 0 no caso contrario
*/
  ap_AGV h;
  ap_no_l_nos layout;
  ap_bloqueio est_lay;
  ulint num_max_mov_avan;
{
  ap_no_l_nos no_orig, no_dest;
  ap_no_l_apno prox_no_traj;
  ap_bloqueio bloq;
  ap_no_l_ramos ramo, r;
  ap_no_l_ulint no_oc;
  char no_viavel;

  while (h!=NULL)
  {
    if (h->tarefa_agv!=NULL &&
     (h->estado_movim==BLOQUEADO ||
      (h->estado_movim==EX_MOVIM && h->num_mov_avan<num_max_mov_avan)
     )
        )
     {
      if (h->estado_movim==BLOQUEADO)
      {
     no_orig=h->posicao;
```

```
    if (h->tarefa_agv->estado_tar==PASSO1)
           no_dest=h->tarefa_agv->traj1->no;
    else if (h->tarefa_agv->estado_tar==PASSO3)
           no_dest=h->tarefa_agv->traj2->no;
       else no_dest=NULL;
      }
      else
      {
    prox_no_traj=obtem_no_traj_mov_avan(h);
    if (prox_no_traj!=NULL)
    {
      no_dest=prox_no_traj->no;
      if (prox_no_traj->anterior!=NULL) no_orig=prox_no_traj->anterior-
>no;
           else no_orig=h->posicao;
    }
    else no_dest=NULL;
      }

      if (no_dest!=NULL) /* se o AGV tiver algum movimento para executar
*/
      {
    bloq=bloq_movim_inviavel(h, no_orig, no_dest->id_no, est_lay);
    if (bloq!=NULL && bloq->agv->estado_movim==PARADO && bloq->agv-
>tarefa_agv==NULL)
    { /* foi encontrado um AGV bloqueante */
/*    ramo=get_ramo(no_orig, no_dest->id_no);
      r=bloq->agv->posicao->ramos;
      while (r!=NULL)
      {
        no_oc=get_no(layout, r->destino)->ocupacao_nos;
        no_viavel=1;
        while (no_oc!=NULL && no_viavel)
        {
           if (procura_ulint(no_oc->i, ramo->bloqueios)!=NULL)
                 no_viavel=0;
           no_oc=no_oc->seguinte;
        }*/

        /* se o no_escape nao inclui na sua lista de ocupacao de nos
nenhum
           dos nos necessarios para desbloquear o AGV
        */
        /*if (no_viavel && bloq_movim_inviavel(bloq->agv, bloq->agv-
>posicao,
                 r->destino, est_lay)==NULL) return 1;
        r=r->seguinte;
      }*/
      return 1;
        }
      }
    }
    h=h->seguinte;
  }
  return 0;
}

ap_AGV agv_bloq_por_agv_PARADO(h, layout, est_lay,
    num_max_mov_avan, agv_bloqueante, r_no_escape)
```

```
/*
- verifica se existe um AGV que obedeca as seguintes condicoes:
     -estado_movim==BLOQUEADO ou EX_MOVIM
     -AGV bloqueante no estado PARADO
     -AGV bloqueante sem tarefa
- retorna o apontador para o AGV se existir, ou NULL se nao existir
- se existir, retorna atraves de 'agv_bloqueante' e de 'r_no_escape', o
AGV
bloqueante e o ramo associado ao no destino para onde deve ser movido
*/
  ap_AGV h;
  ap_no_l_nos layout;
  ap_bloqueio est_lay;
  ulint num_max_mov_avan;
  ap_AGV *agv_bloqueante;
  ap_no_l_ramos *r_no_escape;
{
  ap_no_l_nos no_orig, no_dest, no_escape;
  ap_no_l_apno prox_no_traj;
  ap_bloqueio bloq;
  ap_no_l_ramos ramo, r;
  ap_no_l_ulint no_oc;
  char no_viavel;
  ap_no_l_ramos r_cmin;

  *agv_bloqueante=NULL;
  *r_no_escape=NULL;

  while (h!=NULL)
  {
    if (h->tarefa_agv!=NULL &&
     (h->estado_movim==BLOQUEADO ||
      (h->estado_movim==EX_MOVIM && h->num_mov_avan<num_max_mov_avan)
     )
        )
    {
      if (h->estado_movim==BLOQUEADO)
      {
    no_orig=h->posicao;
    if (h->tarefa_agv->estado_tar==PASSO1)
            no_dest=h->tarefa_agv->traj1->no;
    else if (h->tarefa_agv->estado_tar==PASSO3)
            no_dest=h->tarefa_agv->traj2->no;
       else no_dest=NULL;
      }
      else
      {
    prox_no_traj=obtem_no_traj_mov_avan(h);
    if (prox_no_traj!=NULL)
    {
      no_dest=prox_no_traj->no;
      if (prox_no_traj->anterior!=NULL) no_orig=prox_no_traj->anterior-
>no;
            else no_orig=h->posicao;
    }
    else no_dest=NULL;
      }
```

```
      if (no_dest!=NULL) /* se o AGV tiver algum movimento para executar
*/
      {
    bloq=bloq_movim_inviavel(h, no_orig, no_dest->id_no, est_lay);
    if (bloq!=NULL && bloq->agv->estado_movim==PARADO && bloq->agv-
>tarefa_agv==NULL)
    { /* foi encontrado um AGV bloqueante */
      *agv_bloqueante=bloq->agv;
      ramo=get_ramo(no_orig, no_dest->id_no);
      r=bloq->agv->posicao->ramos;
      *r_no_escape=NULL; r_cmin=NULL;
      while (r!=NULL)
      {
        if (*r_no_escape==NULL || r->custo<(*r_no_escape)->custo)
        {
          no_oc=get_no(layout, r->destino)->ocupacao_nos;
/*        no_viavel=1;
          while (no_oc!=NULL && no_viavel)
          {
           if (procura_ulint(no_oc->i, ramo->bloqueios)!=NULL)
                 no_viavel=0;
           no_oc=no_oc->seguinte;
          }*/

          /* se o no_escape nao inclui na sua lista de ocupacao de nos
nenhum
           dos nos necessarios para desbloquear o AGV
          */
          /*if (no_viavel && bloq_movim_inviavel(bloq->agv, bloq->agv-
>posicao,
                 r->destino, est_lay)==NULL)*/
          if (inters_igual_vazio(no_oc, ramo->bloqueios) &&
             inters_igual_vazio(no_orig->ocupacao_nos, r->bloqueios))
           *r_no_escape=r;
        }
        if (r_cmin==NULL || r->custo<r_cmin->custo) r_cmin=r;
        r=r->seguinte;
      }

      if (*r_no_egcape==NULL) *r_no_escape=r_cmin;

      if (*r_no_escape!=NULL) return h;
    }
      }
    }
    h=h->seguinte;
  }
  return h;
}

ap_no_l_ulint forma_l_nos_oc_AGVs_ERRO(est_lay)
/*
- retorna um apontador para a cabeca da lista de nos ocupados por AGV's
no estado ERRO
*/
  ap_bloqueio est_lay;
{
  ap_no_l_ulint p_nos_oc_agvs_erro=NULL, u_nos_oc_agvs_erro=NULL;
```

```
  while (est_lay!=NULL)
  {
    if (est_lay->agv->estado_movim==ERRO)
    insere_ulint(est_lay->no->id_no,
     &p_nos_oc_agvs_erro, &u_nos_oc_agvs_erro, 0);
    est_lay=est_lay->seguinte;
  }
/*  prn_string("\nLista de nos ocupados por AGV's no estado ERRO: ");
  prn_l_ulint(p_nos_oc_agvs_erro);*/
  return p_nos_oc_agvs_erro;
}

ap_AGV_bloqueado obtem_deadlock_circular(h, layout, est_lay,
    poss_acesso, p_nos_oc_agvs_erro)
/*
- procura detectar um Deadlock Circular envolvendo 2 ou mais AGV's
- se existir e se for resoluvel, retorna uma lista de AGV's bloqueados,
senao retorna NULL
*/
  ap_AGV h;
  ap_no_l_nos layout;
  ap_bloqueio est_lay;
  int poss_acesso;
  ap_no_l_ulint *p_nos_oc_agvs_erro;
{
  ap_AGV_bloqueado p_agvs_bloq=NULL, u=NULL, p_agvs_dead, p, p2;
  ap_bloqueio bloq;
  int acesso, dead_nencontr;
  ap_AGV prox_agv_dead;

  *p_nos_oc_agvs_erro=forma_l_nos_oc_AGVs_ERRO(est_lay);

  /* cria a lista dos AGV's bloqueados */
  while (h!=NULL)
  {
    if (h->estado_movim==BLOQUEADO && h->tarefa_agv!=NULL &&
      (h->tarefa_agv->estado_tar==PASSO1 || h->tarefa_agv-
>estado_tar==PASSO3)
     )
    {
    if (h->tarefa_agv->estado_tar==PASSO1 && h->tarefa_agv->traj1!=NULL)
    {
      bloq=bloq_movim_inviavel(h, h->posicao,
           h->tarefa_agv->traj1->no->id_no, est_lay);
      if (bloq!=NULL) insere_AGV_bloq(h, bloq->agv,
          get_ramo(h->posicao, h->tarefa_agv->traj1->no->id_no),
          &p_agvs_bloq, &u);
    }
    else if (h->tarefa_agv->traj2!=NULL)
    {
      bloq=bloq_movim_inviavel(h, h->posicao,
           h->tarefa_agv->traj2->no->id_no, est_lay);
      if (bloq!=NULL) insere_AGV_bloq(h, bloq->agv,
          get_ramo(h->posicao, h->tarefa_agv->traj2->no->id_no),
          &p_agvs_bloq, &u);
    }
    }
    h=h->seguinte;
  }
```

```
  /* procura um Deadlock Circular */
  p=p_agvs_bloq; p_agvs_dead=NULL; u=NULL; dead_nencontr=1;
  while (p!=NULL && dead_nencontr)
  {
    insere_AGV_bloq(p->agv, p->agv_bloq, p->ramo, &p_agvs_dead, &u);
    prox_agv_dead=p->agv_bloq;
    do{
      if ((p2=obtem_AGV_bloq(prox_agv_dead, p_agvs_dead))!=NULL)
        if (p2==p_agvs_dead) dead_nencontr=0;
        else prox_agv_dead=NULL; /* o 1. AGV da lista nao intervem no
Deadlock Circular */
      else
        if ((p2=obtem_AGV_bloq(prox_agv_dead, p_agvs_bloq))!=NULL)
        {
           insere_AGV_bloq(p2->agv, p2->agv_bloq, p2->ramo,
&p_agvs_dead, &u);
           prox_agv_dead=p2->agv_bloq;
        }
        else prox_agv_dead=NULL;
    } while (prox_agv_dead!=NULL && dead_nencontr);

    if (!dead_nencontr)
    {
      p2=p_agvs_dead; dead_nencontr=1;
      while (p2!=NULL)
      {
        if (p2->agv->tarefa_agv->tipo_tar!=POSICION &&
                p2->agv->tarefa_agv->tipo_tar!=POSIC_ESPEC)
           if (p2->agv->tarefa_agv->estado_tar==PASSO1)
                if (p2->ramo->destino==p2->agv->tarefa_agv->no1-
>id_no)
                        acesso=poss_acesso;
                else acesso=0;
           else
           if (p2->ramo->destino==p2->agv->tarefa_agv->no2->id_no)
                acesso=poss_acesso;
           else acesso=0;
        else acesso=0;

        if (p2->agv->tarefa_agv->estado_tar==PASSO1)
        {
           p2->custo_adicional=custo_traj_optimo(layout, p2->agv-
>posicao,
                p2->agv->tarefa_agv->no1, *p_nos_oc_agvs_erro, p2-
>ramo, acesso)-
            p2->agv->tarefa_agv->ctraj1;
           if (p2->custo_adicional+p2->agv->tarefa_agv->ctraj1<MAXINT)
                dead_nencontr=0;
        }
        else
        {
           p2->custo_adicional=custo_traj_optimo(layout, p2->agv-
>posicao,
                p2->agv->tarefa_agv->no2, *p_nos_oc_agvs_erro, p2-
>ramo, acesso)-
            p2->agv->tarefa_agv->ctraj2;
           if (p2->custo_adicional+p2->agv->tarefa_agv->ctraj2<MAXINT)
                dead_nencontr=0;
```

```
        }

        p2=p2->seguinte;
      }
    }

    if (dead_nehcontr) { limpa_l_AGVs_bloq(&p_agvs_dead); u=NULL; }

    p=p->seguinte;
  }

  limpa_l_AGVs_bloq(&p_agvs_bloq);
  return (p_agvs_dead);
}

int existe_deadlock_circular(h, layout, est_lay, poss_acesso)
/*
- procura detectar um Deadlock Circular envolvendo 2 ou mais AGV's
- se existir e se for resoluvel, retorna 1, senao retorna NULL
*/
  ap_AGV h;
  ap_no_l_nos layout;
  ap_bloqueio est_lay;
  int poss_acesso;
{
  ap_no_l_ulint p_nos_oc_agvs_erro;
  ap_AGV_bloqueado p_agvs_dead;

  if ((p_agvs_dead=
   (obtem_deadlock_circular(h, layout, est_lay,
     poss_acesso, &p_nos_oc_agvs_erro))!=NULL)
  {
    limpa_l_ulint(p_nos_oc_agvs_erro);
    limpa_l_AGVs_bloq(&p_agvs_dead);
    return 1;
  }
  else
  {
    limpa_l_ulint(p_nos_oc_agvs_erro);
    return 0;
  }
}

void resolve_deadlock_circular(agvs_dead, layout, poss_acesso,
p_nos_oc_agvs_erro)
/*
- resolve um Deadlock Circular (resoluvel) representado pela lista,
cuja cabeca e 'agvs_dead'
- calcula um trajecto alternativo para o AGV cujo custo adicional
seja minimo
*/
  ap_AGV_bloqueado agvs_dead;
  ap_no_l_nos layout;
  int poss_acesso;
  ap_no_l_ulint p_nos_oc_agvs_erro;
{
  ap_AGV_bloqueado agv_cmin=NULL, p=agvs_dead;
  int acesso;
  ap_no_l_apno traj_altern;
```

```
  ap_no_l_nos no_dest;

  prn_string("\nSGAGV: resolve o Deadlock Circular, representado pela
lista: \n");
  prn_l_AGVs_bloq(agvs_dead);

  while (p!=NULL)
  {
    if (agv_cmin==NULL || p->custo_adicional<agv_cmin->custo_adicional)
    agv_cmin=p;
    p=p->seguinte;
  }

  if (agv_cmin!=NULL)
  {
    sprintf(s, " vai ser calculado um trajecto alternativo para o AGV
%u\n",
    agv_cmin->agv->id_agv);
    prn_string(s);

    if (agv_cmin->agv->tarefa_agv->tipo_tar!=POSICION &&
    agv_cmin->agv->tarefa_agv->tipo_tar!=POSIC_ESPEC)
    if (agv_cmin->agv->tarefa_agv->estado_tar==PASSO1)
          if (agv_cmin->ramo->destino==agv_cmin->agv->tarefa_agv->no1-
>id_no)
                acesso=poss_acesso;
          else acesso=0;
    else
    if (agv_cmin->ramo->destino==agv_cmin->agv->tarefa_agv->no2->id_no)
          acesso=poss_acesso;
    else acesso=0;
    else acesso=0;

    if (agv_cmin->agv->tarefa_agv->estado_tar==PASSO1)
    no_dest=agv_cmin->agv->tarefa_agv->no1;
    else no_dest=agv_cmin->agv->tarefa_agv->no2;

    traj_optimo(layout, agv_cmin->agv->posicao,
    no_dest, p_nos_oc_agvs_erro, agv_cmin->ramo, acesso, &traj_altern);

    if (traj_altern!=NULL)
    {
/*  if (agv_cmin->agv->tarefa_agv->estado_tar==PASSO1)
          traj_antigo=agv_cmin->agv->tarefa_agv->traj1;
    else traj_antigo=agv_cmin->agv->tarefa_agv->traj2;

    u_traj_altern=traj_altern;
    while (u_traj_altern->no->id_no!=agv_cmin->ramo->destino)
          u_traj_altern=u_traj_altern->seguinte;

    u_traj_altern->seguinte=traj_antigo->seguinte;

    if (traj_antigo->seguinte!=NULL) traj_antigo->seguinte-
>anterior=u_traj_altern;
    free(traj_antigo);*/

    if (agv_cmin->agv->tarefa_agv->estado_tar==PASSO1)
    {
          limpa_l_apno(&agv_cmin->agv->tarefa_agv->traj1);
```

```
                agv_cmin->agv->tarefa_agv->traj1=traj_altern;
                agv_cmin->agv->tarefa_agv->ctraj1+=agv_cmin->custo_adicional;
        }
        else
        {
                limpa_l_apno(&agv_cmin->agv->tarefa_agv->traj2);
                agv_cmin->agv->tarefa_agv->traj2=traj_altern;
                agv_cmin->agv->tarefa_agv->ctraj2+=agv_cmin->custo_adicional;
        }

        sprintf(s, " trajecto alternativo com um custo adicional de %u:\n ",
                agv_cmin->custo_adicional);
        prn_string(s);
        prn_l_apno(traj_altern);
        }
    }
    limpa_l_ulint(p_nos_oc_agvs_erro);
    limpa_l_AGVs_bloq(&agvs_dead);
}

int cam_bloq_nos_ocup(pos, traj, nos_oc)
/*
- verifica se ha algum ramo de um trajecto que necessite de bloquear
os nos pertencsntes a nos_oc
-/
    ap_no_l_nos pos;
    ap_no_l_apno traj;
    ap_no_l_ulint nos_oc;
{
    while (traj!=NULL)
    {
        if (!inters_igual_vazio(get_ramo(pos, traj->no->id_no)->bloqueios,
                nos_oc)) return 1;
        pos=traj->no;
        traj=traj->seguinte;
    }
    return 0;
}

int ha_agvs_bloq_por_agvs_ERRO(h, est_lay, layout, poss_acesso)
/*
- forma uma lista de nos ocupados por AGV's no estado ERRO e verifica se
existe pelo menos 1 AGV que obedeca as seguintes condicoes:
 - no estado EX_MOVIM ou BLOQUEADO com Tarefa no estado PASSO1 ou
PASSO3;
 - com necessidade de bloquear nos que constem na lista de nos ocupados
   pelos AGV's no estado ERRO;
 - para o qual exista um caminho alternativo que nao implique o bloqueio
   do AGV por causa dos AGV's no estado ERRO.
*/
    ap_AGV h;
    ap_bloqueio est_lay;
    ap_no_l_nos layout;
    int poss_acesso;
{
    ap_no_l_ulint p_nos_oc_agvs_erro;
    ap_no_l_nos or, de;
    ap_no_l_apno *traj;
      ulint c;
```

```
  p_nos_oc_agvs_erro=forma_l_nos_oc_AGVs_ERRO(est_lay);

/*  prn_string("\n Nos ocupados por AGV's no estado ERRO:");
  prn_l_ulint(p_nos_oc_agvs_erro);*/

  if (p_nos_oc_agvs_erro!=NULL)
  {
  while (h!=NULL)
  {
    if ((h->estado_movim==EX_MOVIM || h->estado_movim==BLOQUEADO) &&
          h->tarefa_agv!=NULL)
    {
    or=h->posicao;
      if (h->tarefa_agv->estado_tar==PASSO1)
    {
      de=h->tarefa_agv->no1;
      traj=&h->tarefa_agv->traj1;
    }
    else if (h->tarefa_agv->estado_tar==PASSO3)
    {
      de=h->tarefa_agv->no2;
      traj=&h->tarefa_agv->traj2;
    }
    else traj=NULL;

    if (traj!=NULL && cam_bloq_nos_ocup(or, *traj, p_nos_oc_agvs_erro))
      if ((c=custo_traj_optimo(layout, or, de, p_nos_oc_%gvs_erro,
                  NULL, poss_acesso))<MAXINT)
      {
/*          prn_string("\nTrajecto actual:");
            prn_l_apno(*traj);
            sprintf(s, "  \nAGV %u pode ser redireccionado com custo
%u\n", h->id_agv, c);
            prn_string(s);*/
            limpa_l_ulint(p_nos_oc_agvs_erro);
            return 1;
      }
    }
    h=h->seguinte;
  }
  limpa_l_ulint(p_nos_oc_agvs_erro);
  }

  return 0;
}

void resolve_bloq_por_agvs_ERRO(h, est_lay, layout, poss_acesso,
agvs_redir)
/*
- forma uma lista de nos ocupados por AGV's no estado ERRO e procura
AGV's que obedecam as seguintes condicoes:
 - no estado EX_MOVIM ou BLOQUEADO com Tarefa no estado PASSO1 ou
PASSO3;
 - com necessidade de bloquear nos que constem na lista de nos ocupados
   pelos AGV's no estado ERRO;
 - para o qual exista um caminho alternativo que nao implique o bloqueio
   do AGV por causa dos AGV's no estado ERRO.
- para cada um desses AGV's actualiza o seu trajecto
```

```
*/
  ap_AGV h;
  ap_bloqueio est_lay;
  ap_no_l_nos layout;
  int poss_acesso;
  ap_no_l_ap_AGVs *agvs_redir;
{
  ap_no_l_ulint p_nos_oc_agvs_erro;
  ap_no_l_nos or, de;
  ap_no_l_apno *traj, novo_traj;
  ulint *ctraj, c_novo_traj;
  ap_no_l_ap_AGVs u=NULL;

  prn_string("\nSGAGV: resolucao de conflitos provocados por AGV's no
estado ERRO\n");
  *agvs_redir=NULL;

  p_nos_oc_agvs_erro=forma_l_nos_oc_AGVs_ERRO(est_lay);

  if (p_nos_oc_agvs_erro!=NULL)
  while (h!=NULL)
  {
    if ((h->estado_movim==EX_MOVIM || h->estado_movim==BLOQUEADO) &&
          h->tarefa_agv!=NULL)
    {
    or=h->posicao;
      if (h->tarefa_agv->estado_tar==PASSO1)
    {
      de=h->tarefa_agv->no1;
      traj=&h->tarefa_agv->traj1;
      ctraj=&h->tarefa_agv->ctraj1;
    }
    else if (h->tarefa_agv->estado_tar==PASSO3)
    {
      de=h->tarefa_agv->no2;
      traj=&h->tarefa_agv->traj2;
      ctraj=&h->tarefa_agv->ctraj2;
    }
    else traj=NULL;

    if (traj!=NULL && cam_bloq_nos_ocup(or, *traj, p_nos_oc_agvs_erro))
      if ((c_novo_traj=traj_optimo(layout, or, de, p_nos_oc_agvs_erro,
            NULL, poss_acesso, &novo_t^aj))<MAXINT)
      {
          sprintf(s, " foi calculado um novo caminho para o AGV %u com
um custo adicional de %u\n",
                h->id_agv, c_novo_traj-*ctraj);
          prn_string(s);
          *ctraj=c_novo_traj;
          limpa_l_apno(traj);
          *traj=novo_traj;
          insere_no_l_ap_AGVs(h, agvs_redir, &u);
      }
    }
    h=h->seguinte;
  }

  limpa_l_ulint(p_nos_oc_agvs_erro);
}
```

```
void act_dados_MFCFS_fila(fila, h)
/*
- actualiza tar_mais_ant e t_MFCFS da fila, apos a afectacao de uma
tarefa
com origem nessa fila
*/
  ap_EstadoFila fila;
  ap_Tarefa h;
{
  fila->tar_mais_ant=NULL;
  fila->t_MFCFS=1.0E99;

  while (h!=NULL)
  {
    if (h->no1->id_no==fila->id &&
     (h->tipo_tar==NORMAL || h->tipo_tar==T_CARGA) &&
     h->estado_tar==FILA &&
         h->t_lancam<fila->t_MFCFS)
    {
      fila->tar_mais_ant=h;
      fila->t_MFCFS=h->t_lancam;
    }
    h=h->seguinte;
  }

  if (fila->tar_mais_ant==NULL) fila->t_MFCFS=0.0;
}
```

A.2.14 spagv.c

```
/* spagv.c */


#include <stdio.h>
#include <ctype.h> /* isdigit() */
#include <string.h>
#include "listas.h"
#include "mens_log.h"
#include "layout.h"
#include "spagv.h"
#include "lefich.h"
#include "fun_aux.h"
#include "rand_fun.h"     /* random()  */
#include "sga_spa.h"
#include "sistagvs.h"


char s[255]; /* buffer utilizado para chamar 'prn_string()' */


/* definicao das funcoes para leitura e validacao
dos ficheiros de configuracao do SPAGV */

void limpa_l_ramos_SP(h)
/*
- limpa uma lista de ramos do SPAGV
*/
```

```
  ap_Ramo_SP h;
{
  ap_Ramo_SP p;

  while (h!=NULL)
  {
    p=h->seguinte;
    free(h);
    h=p;
  }
}


void limpa_l_nos_SP(h)
/*
- limpa uma lista de nos do SPAGV
*/
  ap_No_SP h;
{
  ap_No_SP p;

  while (h!=NULL)
  {
    limpa_l_ramos_SP(h->ramos);
    p=h->seguinte;
    free(h);
    h=p;
  }
}


ap_No_SP get_no_lay_SP(id, h)
/*
- retorna um apontador para o no da lista com id_no==id
- se nao existir retorna NULL
*/
  ulint id;
  ap_No_SP h;
{
  while (h!=NULL && h->id_no!=id) h=h->seguinte;
  return h;
}


ap_No_SP le_layout_SP(n_fich_lay, n_fich_mesas, vre, vcr, vma)
/*
- le o ficheiro de layout, utilizando a funcao le_layout() do SGAGV e
filtra essa
informacao para a formatar segundo a estrutura definida para o SPAGV
- le tambem o ficheiro de mesas para determinar as suas posicoes
relativas
- retorna NULL se ocorrer algum erro
*/
  char *n_fich_lay;
  char *n_fich_mesas;
  ulint vre, vcr, vma;
{
  ap_no_l_nos lay_SG, no_SG;
  ap_No_SP no_SP, h_lay_SP=NULL, u_lay_SP=NULL;
```

```
  ap_no_l_ramos r_SG;
  ap_Ramo_SP u, r_SP;
  char noerrors=1;

  prn_string("\n\nSPAGV");
  lay_SG=le_layout(n_fich_lay, vre, vcr, vma);
  le_f_pos_rel_pcd(n_fich_mesas, lay_SG);

  no_SG=lay_SG;

  while (no_SG!=NULL && noerrors)
  {
    no_SP=(ap_No_SP) malloc(sizeof(No_SP));
    if (no_SP!=NULL)
    {
      no_SP->id_no=no_SG->id_no;
      no_SP->tipo_no=no_SG->tipo_no;
      no_SP->pos_rel_mesa=no_SG->pos_rel_mesa;
      no_SP->ramos=NULL;
      no_SP->seguinte=NULL;
      if (h_lay_SP==NULL) h_lay_SP=no_SP;
      if (u_lay_SP!=NULL) u_lay_SP->seguinte=no_SP;
      u_lay_SP=no_SP;
    }
    else
    {
      prn_string("\"le_layout_SP()\": erro na alocacao de um novo
no\n");
      noerrors=0;
    }
    no_SG=no_SG->seguinte;
  }

  no_SP=h_lay_SP; no_SG=lay_SG;
  while (no_SP!=NULL && noerrors)
  {
    while (no_SG->id_no!=no_SP->id_no) no_SG=no_SG->seguinte;
    r_SG=no_SG->ramos; u=NULL;
    while (r_SG!=NULL && noerrors)
    {
      r_SP=(ap_Ramo_SP) malloc(sizeof(Ramo_SP));
      if (r_SP!=NULL)
      {
    r_SP->cabeca=get_no_lay_SP(r_SG->destino, h_lay_SP);
    if (r_SP->cabeca!=NULL)
    {
      r_SP->tempo=((double) r_SG->custo-r_SG->pen_aux*10)/10.0;
      r_SP->seguinte=NULL;
      if (no_SP->ramos==NULL) no_SP->ramos=r_SP;
      if (u!=NULL) u->seguinte=r_SP;
      u=r_SP;
    }
    else
    {
      free(r_SP);
      sprintf(s, "\"le_layout_SP()\": no %u invalido no destino de um
ramo do no %u\n",
           r_SG->destino, no_SG->id_no);
      prn_string(s);
```

```
        noerrors=0;
      }
        }
        else
        {
      prn_string("\"le_layout_SP()\": erro na alocacao de memoria de um
novo ramo\n");
      noerrors=0;
        }
        r_SG=r_SG->seguinte;
      }
     no_SP=no_SP->seguinte;
   }

   if (!noerrors) { limpa_l_nos_SP(h_lay_SP); h_lay_SP=NULL; }

   limpa_layout(lay_SG);
   return h_lay_SP;
}


void prn_ap_No_SP(ap_no)
/*
- imprime no stdout informacao de um no, a partir de um apontador
*/
   ap_No_SP ap_no;
{
   ap_Ramo_SP r;

   if (ap_no==NULL) prn_string("(NULL)\n");
   else
   {
     sprintf(s, "%u,", ap_no->id_no);
     prn_string(s);
     switch (ap_no->tipo_no)
     {
     case TRANSITO: prn_string("TRANSITO"); break;
     case CARGA_DESCARGA: prn_string("CARGA_DESCARGA"); break;
     case CARGA: prn_string("CARGA"); break;
     case DESCARGA: prn_string("DESCARGA"); break;
     default: prn_string("???");
     }
     prn_string(",Ramos={");

     r=ap_no->ramos;
     while (r!=NULL)
     {
       if (r!=ap_no->ramos) prn_string(",");
       sprintf(s, "(%u,%.1f)", r->cabeca->id_no, r->tempo);
     prn_string(s);
       r=r->seguinte;
     }
     prn_string("}\n");
   }
}


void prn_lay_SP(h)
/*
```

```
- mostra o layout do SPAGV
*/
  ap_No_SP h;
{

  prn_string("Layout do SPAGV:\n");
  while (h!=NULL)
  {
    prn_ap_No_SP(h);
    h=h->seguinte;
  }
}


ap_no_l_ui4d le_f_tempos_op_cd(nome_ficheiro)
/*
- le um ficheiro de configuracao de pontos de carga/descarga com os
tempos associados
  as operacoes de carga e de descarga
- retorna uma lista em que cada elemento contem o valor medio e o desvio
padrao
  associados ao tempo de uma operacao de carga e de uma operacao de
descarga no
  no respectivo
- retorna NULL se ocorrer algum erro
- significado dos campos da estrutura:
  i  : identificador do no/mesa
  d1 : valor medio da duracao de uma operacao de carga
  d2 : desvio padrao da duracao de uma operacao de carga
  d3 : valor medio da duracao de uma operacao de descarga
  d4 : desvio padrao da duracao de uma operacao de descarga
*/
  char *nome_ficheiro;
{
  FILE *f;
  int c;
  ulint id_no, linha=1;
  double tm_op_car, dp_op_car, tm_op_des, dp_op_des;
  ap_no_l_ui4d h=NULL, u=NULL;

  sprintf(s, "\n\nSPAGV: leitura do ficheiro '%s' c/ tempos das op. de
carga/descarga...\n",
    nome_ficheiro);
  prn_string(s);
  f=fopen(nome_ficheiro,"r");
  if (f!=NULL)
  {
    do{
      c=procura_prox_linha(f, 'M', &linha);
      if (c!=EOF)
      {
    sprintf(s, "linha %u (mesa ", linha);
    prn_string(s);
    if (isdigit((c=fgetc(f))))
    {
      id_no=le_ulint(f, c-'0');
      sprintf(s, "%u)\n", id_no);
      prn_string(s);
      if (fgetc(f)==',')
```

```
      {
        if ((c=fgetc(f))=='-') c=fgetc(f); /* le pos_rel_mesa (ignorado)
*/
        if (isdigit(c))
        {
          le_ulint(f, c-'0');
          if (fgetc(f)==',')
          {
           le_double(f, &tm_op_car);
             if (fgetc(f)==',')
             {
               le_double(f, &dp_op_car);
                 if (fgetc(f)==',')
                 {
                  le_double(f, &tm_op_des);
                    if (fgetc(f)==',')
                    {
                      le_double(f, &dp_op_des);
                        if ((c=fgetc(f))=='\n')
                        {
                         linha++;
                         u=insere_ui4d(h, u, id_no, tm_op_car,
dp_op_car,
                                 tm_op_des, dp_op_des, 0);
                         if (h==NULL) h=u;
                        }
                        else {sprintf(s, "erro: caracter '%c'
invalido\n",c);
                              prn_string(s); break;}
                    }
                    else {prn_string("erro: caracter ',' nao encontrado
entre tm_descarga e dp_descarga\n");break;}
                 }
                      else {prn_string("erro: caracter ',' nao
encontrado entre dp_carga e tm_descarga\n");break;}
             }
             else {prn_string("erro: caracter ',' nao encontrado entre
tm_carga e dp_carga\n");break;}
          }
          else {prn_string("erro: caracter ',' nao encontrado entre
pos_rel_mesa e tm_carga\n");break;}
        }
        else { prn_string(" erro: leitura da posicao\n"); break; }
      }
      else { prn_string(" erro: caracter ',' nao encontrado entre id_no
e pos_rel\n"); break; }
    }
    else { prn_string("??)\n erro: leitura do id_no\n"); break; }
     }
     else { sprintf(s, " Fim do ficheiro na linha %u\n", linha);
prn_string(s); }
    } while (c!=EOF);
    fclose(f);
  }
  else { sprintf(s, " erro: abertura do ficheiro %s\n", nome_ficheiro);
prn_string(s); }
  if (c!=EOF) h=limpa_l_ui4d(h);
  return h;
}
```

```
int t_op_mesas_conhec(nos, t_op_cd)
/*
- verifica se e conhecido o valor medio e o desvio padrao do tempo
associado
  as operacoes de carga/descarga de todos os nos do layout de tipo
diferente de
  TRANSITO (mesas)
- retorna 1 em caso afirmativo ou 0 em caso negativo
*/
ap_No_SP nos;
ap_no_l_ui4d t_op_cd;
{
  int t_conh=1;

  prn_string("\n\nSPAGV: verifica se sao conhecidos os tempos para todas
as mesas...\n");
  while (nos!=NULL)
  {
    if (nos->tipo_no!=TRANSITO)
      if (get_ui4d_ui(t_op_cd, nos->id_no)==NULL)
      {
    t_conh=0;
    sprintf(s, "erro: nao sao conhecidos os tempos para a mesa/no %u\n",
nos->id_no);
    prn_string(s);
      }
    nos=nos->seguinte;
  }
  prn_string("Verificacao concluida.\n\n");
  return t_conh;
}


ap_no_l_uid le_f_nos_introd(nome_ficheiro)
/*
- le um ficheiro de configuracao com os nos de introducao e as
respectivas
  probabilidades de ocorrencia de introducoes
- retorna uma lista ligada com essa informacao
- significado dos campos da estrutura:
  i : identificador do no/mesa
  d : probabilidade
*/
  char *nome_ficheiro;
{
  FILE *f;
  int c;
  ulint id_no, linha=1;
  double p;
  ap_no_l_uid h=NULL, u=NULL;

  sprintf(s, "\n\nSPAGV: leitura do ficheiro '%s' c/ nos e probab. de
introducao de AGV's...\n",
    nome_ficheiro);
  prn_string(s);
  f=fopen(nome_ficheiro,"r");
  if (f!=NULL)
```

```
{
  do{
    c=procura_prox_linha(f, 'N', &linha);
    if (c!=EOF)
    {
   sprintf(s, "linha %u (no ", linha);
   prn_string(s);
   if (isdigit((c=fgetc(f))))
   {
     id_no=le_ulint(f, c-'0');
     sprintf(s, "%u)\n", id_no);
     prn_string(s);
     if (fgetc(f)==',')
     {
       le_double(f, &p);
         linha++;
         u=insere_uid(h, u, id_no, p, 0);
         if (h==NULL) h=u;
         procura_caracter(f, '\n', &linha);
     }
     else { prn_string(" erro: caracter ',' nao encontrado entre id_no
e probabilidade\n"); break; }
   }
   else { prn_string("??)\n erro: leitura do id_no\n"); break; }
     }
     else { sprintf(s, " Fim do ficheiro na linha %u\n", linha);
prn_string(s); }
   } while (c!=EOF);
   fclose(f);
 }
 else { sprintf(s, " erro: abertura do ficheiro %s\n", nome_ficheiro);
prn_string(s); }
 if (c!=EOF) h=limpa_l_uid(h);
 return h;
}


int som_prob_unit(nos_intr)
/*
- verifica se o somatorio das probabilidades dos nos de introducao e 1.0
- se for retorna 1 senao retorna 0
*/
 ap_no_l_uid nos_intr;
{
 double soma=0.0;

 prn_string("\n\nSPAGV: verifica somatorio das probab. dos nos de
introducao...\n");
 while (nos_intr!=NULL && soma<(1.0+TOLER_IGUAL_DOUBLES))
  {
    if (nos_intr->d>=0.0 && nos_intr->d<=1.0) soma+=nos_intr->d;
    else
    {
      sprintf(s, " erro: no %u tem uma probabilidade invalida =%.4f\n",
    nos_intr->i,nos_intr->d);
    prn_string(s);
      soma=2.0;
    }
    nos_intr=nos_intr->seguinte;
```

```
  }
  sprintf(s, "Verificacao concluida (soma=%.3f).\n\n", soma);
  prn_string(s);
  if (diguald2(soma,1.0,TOLER_IGUAL_DOUBLES)) return 1; else return 0;
}



/* definicao de outras funcoes utilizadas na RP do SPAGV */

ulint decide_no_intr(nos_intr)
/*
- retorna o identificador do no onde deve ser introduzido um AGV
- baseia esta decisao nos nos e respectivas probabilidades que constem
na lista passada como parametro
- retorna 0 se a lista for vazia
*/
  ap_no_l_uid nos_intr;
{
  double som_p_ant=0.0, p;
  ap_no_l_uid anterior=NULL;

  p=random();

  while (nos_intr!=NULL)
  {
    if ((som_p_ant+nos_intr->d)>=p && nos_intr->d>TOLER_IGUAL_DOUBLES)
    return nos_intr->i;
    som_p_ant+=nos_intr->d;
    if (nos_intr->d>TOLER_IGUAL_DOUBLES) anterior=nos_intr;
    nos_intr=nos_intr->seguinte;
  }

  if (anterior!=NULL) return anterior->i;
    else return 0;
}

double processa_mov_transl(mens, posicao)
/*
- processa uma mensagem relativa a uma ordem de movimento de translaccao
- retorna uma estimativa do tempo necessario para executar o movimento
*/
  mensag_SGAGV_SPAGV mens;
  ap_No_SP *posicao;
{
  ap_Ramo_SP r;

  sprintf(s, "\nSPAGV: inicio de um movimento de translaccao do AGV %u
(%u->%u)\n",
    mens.id_agv, (*posicao)->id_no, mens.p2);
  prn_string(s);
  if (*posicao!=NULL)
  {
    r=(*posicao)->ramos;
    while (r!=NULL && r->cabeca->id_no!=mens.p2)
    r=r->seguinte;
    if (r!=NULL)
    {
    *posicao=r->cabeca;
```

```
      sprintf(s, "\tVai demorar %.1f segundos\n", r->tempo);
      prn_string(s);
      return(r->tempo);
      }
    }
    return 0.0;
}

void processa_mov_car_des(mens, posicao)
/*
- processa uma mensagem relativa a uma ordem de movimento de carga ou de
descarga
*/
  mensag_SGAGV_SPAGV mens;
  ap_No_SP posicao;
{
  switch (mens.p1)
  {
    case MOV_CARGA: sprintf(s, "\nSPAGV: inicio de uma operacao de carga
com o AGV %u,\n no no %u, para o lado ",
                          mens.id_agv, posicao->id_no);
            break;
    case MOV_DESCARGA: sprintf(s, "\nSPAGV: inicio de uma operacao de
descarga com o AGV %u,\n no no %u, para o lado ",
                          mens.id_agv, posicao->id_no);
            break;
  }
  prn_string(s);

  switch (mens.p2)
  {
    case ESQUERDA: sprintf(s, "Esquerdo\n"); break;
    default:       sprintf(s, "Direito\n");
  }
  prn_string(s);
}
```

Anexo B – Communication units

Este anexo apresenta as *communication units* utilizadas para definir os tipos dos *tokens* utilizados na comunicação entre os objectos do modelo do SAGVs, descrito nos capítulos 8 e 9. O nome atribuído às CMN Units coincide com o tipo dos *tokens* por elas definido. Por exemplo, a CMN Unit define o tipo ULINT.

B.1 ULINT

B.1.1 Ficheiros de cabeçalho incluídos

```
#include "ulint.h"
```

B.1.2 Definição

```
ulint i;
```

B.2 BOOLEANO

```
int b;
```

B.3 MENS_SGM_SGAGV

B.3.1 Ficheiros de cabeçalho incluídos

```
#include "ulint.h"
#include "sgm_sga.h"
```

B.3.2 Definição

```
mensag_SGM_SGAGV mens;
```

B.4 MENS_SGAGV_SPAGV

B.4.1 Ficheiros de cabeçalho incluídos

```
#include "ulint.h"
#include "sga_spa.h"
```

B.4.2 Definição

```
mensag_SGAGV_SPAGV mens;
```

B.5 TEMPORIZACAO

B.5.1 Ficheiros de cabeçalho incluídos

```
#include "ulint.h"
```

B.5.2 Definição

```
ulint id;
double t;
```

B.5.3 Descrição dos campos

- id – identificador da temporização;
- t – tempo associado.

B.6 ORDEM

```
unsigned long int origem;
unsigned long int destino;
```

B.7 CAPAC_FILA

B.7.1 Definição

```
unsigned long int fila;
unsigned long int cap;
```

B.7.2 Descrição dos campos

- fila – identificador da fila de espera;
- cap – capacidade da fila de espera.

B.8 PALETE

B.8.1 Definição

```
unsigned long int fila;
unsigned long int destino;
unsigned long int recurso;
double dt_prox_oper;
double t_mud_est;
```

B.8.2 Descrição dos campos mais importantes

- fila – identificador da fila de espera;

- destino – ponto para onde a palete deve ser encaminhada.

Anexo C – Classes Artifex das 3 camadas de gestão do SAGVs

C.1 Classe TEMPORIZADOR

C.1.1 CMN units incluídas

- TEMPORIZACAO;
- ULINT.

C.1.2 Posições de entrada

- TEMPO:TEMPORIZACAO;
- ID_TEMPOR_INTERR:ULINT.

C.1.3 Posições de saída

- ID_FIM_TEMPOR:ULINT.

C.1.4 Ficheiros de cabeçalho incluídos

```
#include "listas.h"
```

C.1.5 Funções C definidas internamente

```
static void prn_AP_NO_L_UID(ap)
  AP_NO_L_UID *ap;
{
 if (ap->p!=NULL)
    printf("\n(ulint) i=%u, (double) d=%f\n", ap->p->i, ap->p->d);
}

static void prn_ULINT(ap)
  ULINT *ap;
{ printf("(ulint) i=%u\n", ap->i); }

static void prn_TEMPORIZACAO(ap)
  TEMPORIZACAO *ap;
{ printf("(ulint) id=%u, (double) t=%f\n", ap->id, ap->t); }
```

C.1.4 Acções iniciais

```
XX->h_lista_tempor=NULL;
XX->u_lista_tempor=NULL;
```

```
XW_AP_NO_L_UID_print=prn_AP_NO_L_UID;
XW_ULINT_print=prn_ULINT;
XW_TEMPORIZACAO_print=prn_TEMPORIZACAO;
xx_setstate(0);
```

C.1.5 Acções finais

```
limpa_l_uid(XX->h_lista_tempor);
```

C.1.6 Tipos de tokens

C.1.6.1 AP_NO_L_UID

```
ap_no_l_uid p;
```

C.1.7 Posições

C.1.7.1 TEMPOR_ACTIVA:AP_NO_L_UID

C.1.7.1.1 Inicialização do token da marcação inicial

```
TEMPOR_ACTIVA->p=NULL;
```

C.1.8 Transições

C.1.8.1 INICIA_TEMPORIZACAO

C.1.8.1.1 Propriedades básicas

- prioridade 2.

C.1.8.1.2 Acção

```
ap_no_l_uid u=insere_uid(XX->h_lista_tempor,
    XX->u_lista_tempor,
    TEMPO->id, xx_gettime()+TEMPO->t, 0);
if (XX->u_lista_tempor!=u)
{
  XX->u_lista_tempor=u;
  xx_setstate(xx_getstate()+1);

  if (XX->h_lista_tempor==NULL)
   XX->h_lista_tempor=XX->u_lista_tempor;

  TEMPOR_ACTIVA->p=obtem_uid_min_d(XX->h_lista_tempor);

  if (TEMPOR_ACTIVA->p!=NULL)
    xx_fdelay(XT_TEMPORIZA,
    TEMPOR_ACTIVA->p->d-xx_gettime());
 }
```

C.1.8.2 INTERR_TEMPOR_AGORA_INICIADA

C.1.8.2.1 Propriedades básicas

- Prioridade 3.

C.1.8.2.2 Predicado

```
(TEMPO->id==ID_TEMPOR_INTERR->i)
```

C.1.8.3 INTERR_TEMPOR_A_DECORRER

C.1.8.3.1 Propriedades básicas

- Prioridade 1.

C.1.8.3.2 Acção

```
if (remove_uid(ID_TEMPOR_INTERR->i,
    &XX->h_lista_tempor, &XX->u_lista_tempor))
{
  xx_setstate(xx_getstate()-1);
  TEMPOR_ACTIVA->p=obtem_uid_min_d(XX->h_lista_tempor);
  if (TEMPOR_ACTIVA->p!=NULL)
     xx_fdelay(XT_TEMPORIZA,
    TEMPOR_ACTIVA->p->d-xx_gettime());
}
```

C.1.8.4 TEMPORIZA

C.1.8.4.1 Propriedades básicas

- Prioridade 0;
- Tempo de disparo = 0.

C.1.8.4.2 Predicado

```
(TEMPOR_ACTIVA->p!=NULL)
```

C.1.8.4.3 Acção

```
ID_FIM_TEMPOR->i=TEMPOR_ACTIVA->p->i;

remove_uid(TEMPOR_ACTIVA->p->i,
    &XX->h_lista_tempor, &XX->u_lista_tempor);
xx_setstate(xx_getstate()-1);

TEMPOR_ACTIVA->p=obtem_uid_min_d(XX->h_lista_tempor);

if (TEMPOR_ACTIVA->p!=NULL)
  xx_fdelay(XT_TEMPORIZA,
```

```
TEMPOR_ACTIVA->p->d-xx_gettime());
```

C.2 Classe SGM

C.2.1 CMN units incluídas

- MENS_SGM_SGAGV;
- BOOLEANO;
- TEMPORIZACAO;
- ULINT;
- ORDEM.

C.2.2 Posições de entrada

- MENS_RECEB_SGAGV:MENS_SGM_SGAGV;
- NOVA_TAREFA_NORMAL:ORDEM.

C.2.3 Posições de saída

- MENS_ENV_SGAGV:MENS_SGM_SGAGV;
- PONTO_CARGA_DISPONIVEL:ULINT.

C.2.4 Parâmetros

- numero_agvs: unsigned long int – número máximo de AGVs no sistema;
- nome_fich_nos_tarefas: xx_string – nome do ficheiro de configuração de nós de introdução e de nós candidatos a destino nas tarefas de posicionamento;
- t_med_dis_en:double:double – valor médio do tempo que um AGV permanece no estado DISABLED, antes de passar ao estado ENABLED; admite-se que este tempo segue uma FDP exponencial;
- t_med_en_dis:double:double – valor médio do tempo que decorre desde a passagem de um AGV ao estado ENABLED, até voltar ao estado DISABLED; admite-se que este tempo segue uma FDP exponencial;
- t_med_mud_est_aloc:double – valor médio do tempo que medeia entre duas alterações do estado de alocação de um AGV; admite-se que este tempo segue uma FDP exponencial;
- p_alocado:double – probabilidade de um AGV ficar no estado de alocação ALOCADO, quando transita do estado DISABLED para o estado ENABLED; a probabilidade de ficar no estado NAO_ALOCADO é complementar;

- vm_carga:double – valor médio da carga do SAGVs, definida como o quociente entre o número de tarefas pendentes e o número de AGVs num estado de movimentação não pertencente ao conjunto {DISABLED, ENABLED, ERRO}; admite-se que este tempo segue uma FDP normal (vm_carga, dp_carga^2);
- dp_carga:double – desvio padrão da carga do SAGVs;
- p_esc_sgm:double – probabilidade da afectação de uma nova tarefa a um AGV ser decidida pelo SGM;
- p_tar_prior:double – probabilidade do SGM criar uma nova tarefa com prioridade superior a 0, quando o escalonamento da mesma for remetido para o SGAGV;
- priorid_tar_max:unsigned long int – valor máximo da prioridade de uma tarefa escalonada pelo SGAGV;
- t_med_cancel_tar:double – valor méedio do tempo que decorre entre o envio ao SGAGV de duas mensagens de cancelamento de tarefas; admite-se que este tempo segue uma FDP exponencial;
- p_cancel_1_tar:double – probabilidade de uma mensagem do tipo CANCELA_TAREFA ter o parâmetro 1 igual a UMA_TAREFA;
- p_cancel_tar_exec:double – probabilidade de uma mensagem do tipo CANCELA_TAREFA ter o parâmetro 1 igual a TODAS_TAR_EXEC;

C.2.5 Ficheiros de cabeçalho incluídos

```
#include "listas.h"
#include "mens_log.h"
#include "rand_fun.h"
#include "sgm.h"
#include "sistagvs.h"
```

C.2.6 Funções definidas internamente

```
/* funcao para permitir a visualizacao das probabilidades de
ocorrencia de movimentos em cada no, a partir de uma posicao que
contenha um token do tipo AP_NOS_TAREFAS (utilizando o comando
"examine Queue") */
void prn_AP_NOS_TAREFAS(a)
 AP_NOS_TAREFAS *a;
{
 prn_string("Probabilidades de ocorrencia de movimentos em cada no\n");
 prn_string("(<id_no>,<p_posic>)\n");
 prn_l_uid(a->ap);
 prn_string("\n");
}

/* funcao para permitir a visualizacao do valor da variavel
booleana contida num token do tipo BOOLEANO
(utilizando o comando "examine Queue") */
void prn_BOOLEANO_SGM(bool)
 BOOLEANO *bool;
{if (bool->b==FALSO) prn_string("FALSO\n"); else
prn_string("VERDADE\n");}

/* funcao para permitir a visualizacao de um AGV a partir de uma
posicao que contenha um token do tipo AP_AGV (utilizando o comando
"examine Queue") */
void prn_AP_AGV(a)
```

```
 AP_AGV *a;
{ prn_agv(a->agv); }

/* funcao para permitir a visualizacao do conteudo de uma
mensagem SGM <-> SGAGV, a partir de uma posicao que contenha
um token do tipo MENS_SGM_SGAGV
(utilizando o comando "examine Queue") */
void prn_MENS_SGM_SGAGV(a)
  MEnS_SGM_SGAGV *a;
{ prn_m_SGM_SGAGV(a->mens); }

/* funcao para permitir a visualizacao de uma Tarefa a partir de
uma posicao que contenha um token do tipo AP_TAREFA (utilizando o
comando "examine Queue") */
void prn_AP_TAREFA(a)
 AP_TAREFA *a;
{ prn_Tarefa(a->tarefa); }

/* funcao para permitir a visualizacao de um token do tipo
DOUBLE (utilizando o comando "examine Queue") */
void prn_DOUBLE(a)
 DOUBLE *a;
{
  char s[500];

  sprintf(s, "(double) d=%f\n", a->d);
  prn_string(s);
}

/* funcao para permitir a visualizacao de um token do tipo
ULINT (utilizando o comando "examine Queue") */
void prn_ULINT_SGM(a)
 ULINT *a;
{
  char s[255];

  sprintf(s, "(ulint) i=%u\n", a->i);
  prn_string(s);
}
```

C.2.7 Acções iniciais

```
/* Examine Queue em posicoes de tipos nao geridos automatica/ */
 XW_AP_NOS_TAREFAS_print = prn_AP_NOS_TAREFAS;
 XW_BOOLEANO_print = prn_BOOLEANO_SGM;
 XW_AP_AGV_print = prn_AP_AGV;
 XW_MENS_SGM_SGAGV_print = prn_MENS_SGM_SGAGV;
 XW_AP_TAREFA_print = prn_AP_TAREFA;
 XW_DOUBLE_print = prn_DOUBLE;
 XW_ULINT_print = prn_ULINT_SGM;

/* inicializa o estado do objecto */
 xx_setstate(0);

/* inicializa a lista ligada de AGV's */
XX->p_agvs=NULL;
XX->u_agvs=NULL;

/* inicializa a lista ligada de AGV's */
XX->p_tarefas=NULL;
XX->u_tarefas=NULL;

/* inicializa o contador de mensagens */
```

```
XX->id_prox_mens=0;

/* inicializa os contadores de tarefas e de AGV's disponiveis */
XX->n_tars=0;
XX->n_agvs_activos=0;
```

C.2.8 Acções finais

```
/* destroi a lista de Nos de Tarefas */
limpa_l_uid(XX->nos_tarefas);

/* destroi a lista de AGV's */
limpa_l_AGVs(XX->p_agvs);

/* destroi a lista de Tarefas */
limpa_l_Tarefas(XX->p_tarefas);
```

C.2.9 Tipos de tokens

C.2.9.1 AP_NOS_TAREFAS

```
ap_no_l_uid ap;
```

C.2.9.2 AP_AGV

```
ap_AGV agv;
```

C.2.9.3 AP_TAREFA

```
ap_Tarefa tarefa;
```

C.2.9.4 DOUBLE

```
double d;
```

C.2.10 Variáveis locais

- nos_tarefas:ap_no_l_uid – apontador para a cabeça da lista, em que cada elemento contém um identificador do nó do *layout* (i) e a probabilidade de ocorrência de um posicionamento (d);
- p_agvs:ap_AGV – apontador para o primeiro elemento da lista ligada de AGVs;
- u_agvs:ap_AGV – apontador para o último elemento da lista ligada de AGVs;
- p_tarefas:ap_Tarefa – apontador para o primeiro elemento da lista ligada de tarefas;
- u_tarefas:ap_Tarefa – apontador para o último elemento da lista ligada de tarefas;
- id_prox_mens:unsigned long int – guarda o identificador da próxima mensagem que o SGM enviará ao SGAGV;
- n_tars:unsigned long int – número de tarefas de movimentação simples;

- n_agvs_activos:unsigned long int – número de AGVs activos (num estado diferente de DISABLED e de ENABLED);
- buffer:xx:string – *buffer* para chamar a função prn_string();

C.2.111 Posições partilhadas

- FIM_CONFIG_SGM:NUL;
- AGV_DISABLED:AP_AGV;
- TEMPO_DIS_EN:TEMPORIZACAO;
- TAREFA_FILA:AP_TAREFA;
- FIM_TEMP_DIS_EN:ULINT;
- TEMPO_MUD_EST_ALOC:TEMPORIZACAO;
- INTERR_TEMP_MUD_EST_ALOC:ULINT;
- FIM_TEMP_MUD_EST_ALOC:ULINT;
- TEMPO_EN_DIS:TEMPORIZACAO;
- FIM_TEMP_EN_DIS:ULINT;
- CARGA_ACTUAL:DOUBLE;
- AGV_ENABLED:AP_AGV;
- AGV_ACTIVO:AP_AGV;
- TAREFA_PASSO1:AP_TAREFA;
- TAREFA_PASSO2:AP_TAREFA;
- TAREFA_PASSO3:AP_TAREFA;
- TAREFA_PASSO4_AP_TAREFA;
- TAREFA_CANCELADA:AP_TAREFA.

C.2.112 Entidades da RP da página principal da classe SGM

C.2.112.1 Objectos

- TEMP_ACT_DIS:TEMPORIZADOR – temporizador utilizado para gerir as decisões relativas à colocação de AGVs activos no estado DISABLED;
- TEMP_DIS_EN_AGVS:TEMPORIZADOR – temporizador utilizado para gerir as decisões relativas à transição para o estado ENABLED dos AGVs no estado DISABLED;
- TEMP_MUD_EST_ALOC:TEMPORIZADOR – temporizador utilizado para gerir as decisões relativas à alteração do estado de alocação dos AGVs, que estejam num estado de movimentação diferente de DISABLED.

C.2.12.2 Acção da transição 'LIMPA_FIM_TEMP_MUD_EST_ALOC'

```
xx_setstate(35);
```

C.2.13 Entidades da RP da página 'gera_AGVs_DISABLED'

C.2.13.1 Posições

- ID_AGV:ULINT.

C.2.13.2 Transições

C.2.13.2.1 GERA_AGV_DISABLED

C.2.13.2.1.1 Predicado

```
(ID_AGV->i<=XX->numero_agvs)
```

C.2.13.2.1.2 Acção

```
AGV agv;

 agv.id_agv=ID_AGV->i;
 agv.estado_movim=DISABLED;
 agv.ult_destino=0;
 agv.tarefa_agv=NULL;
 agv.t_enable=rnd_exp_neg(XX->t_med_dis_en);
 agv.t_disable=0.0;
 agv.t_mud_est_aloc=0.0;

 if (insere_AGV(&agv, &XX->p_agvs, &XX->u_agvs))
 {
   sprintf(XX->buffer, "SGM: AGV %u no estado DISABLED criado\n",
agv.id_agv);
   prn_string(XX->buffer);
 }
 (ID_AGV->i)++;

 AGV_DISABLED->agv=XX->u_agvs;
 TEMPO_DIS_EN->id=agv.id_agv;
 TEMPO_DIS_EN->t=agv.t_enable;

 xx_setstate(5);
```

C.2.13.2.2 Acção de 'INIC_ID_AGV'

```
ID_AGV->i=1;

xx_setstate(4);
```

C.2.13.2.3 Acção de 'LIMPA_ID_AGV'

```
xx_setstate(6);
```

C.2.14 Transições da RP da página 'gere_AGVs_ACTIVOS'

C.2.14.1 DISABLE_AGV_ACTIVO

C.2.14.1.1 Predicado

```
(AGV_ACTIVO->agv->id_agv==FIM_TEMP_EN_DIS->i)
```

C.2.14.1.2 Acção

```
double dt=rnd_exp_neg(XX->t_med_dis_en);
double t=xx_gettime();

AGV_ACTIVO->agv->estado_movim=DISABLED;
AGV_ACTIVO->agv->t_enable=t+dt;

MENS_ENV_SGAGV->mens.id_mens=XX->id_prox_mens;
(XX->id_prox_mens)++;
MENS_ENV_SGAGV->mens.tipo_mens=DISABLE_AGV;
MENS_ENV_SGAGV->mens.p1=AGV_ACTIVO->agv->id_agv;

INTERR_TEMP_MUD_EST_ALOC->i=AGV_ACTIVO->agv->id_agv;

TEMPO_DIS_EN->id=AGV_ACTIVO->agv->id_agv;
TEMPO_DIS_EN->t=dt;

(XX->n_agvs_activos)--;
CARGA_ACTUAL->d=calc_carga(XX->n_tars, XX->n_agvs_activos);

sprintf(XX->buffer,
"\nSGM: AGV %u ENABLED passa ao estado DISABLED apos o envio de:\n",
AGV_ACTIVO->agv->id_agv);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(t, MENS_ENV_SGAGV->mens);

xx_setstate(16);
```

C.2.14.2 MUDA_EST_ALOC_AGV_ACT

C.2.14.2.1 Predicado

```
(AGV_ACTIVO->agv->id_agv==FIM_TEMP_MUD_EST_ALOC->i)
```

C.2.14.2.2 Acção

```
double dt=rnd_exp_neg(XX->t_med_mud_est_aloc);
double t=xx_gettime();

if (AGV_ACTIVO->agv->estado_alocacao==ALOCADO)
   AGV_ACTIVO->agv->estado_alocacao=NAO_ALOCADO;
 else AGV_ACTIVO->agv->estado_alocacao=ALOCADO;
AGV_ACTIVO->agv->t_mud_est_aloc=t+dt;

TEMPO_MUD_EST_ALOC->id=AGV_ACTIVO->agv->id_agv;
TEMPO_MUD_EST_ALOC->t=dt;
```

```
MENS_ENV_SGAGV->mens.id_mens=XX->id_prox_mens;
(XX->id_prox_mens)++;
MENS_ENV_SGAGV->mens.tipo_mens=ENABLE_AGV;
MENS_ENV_SGAGV->mens.p1=AGV_ACTIVO->agv->id_agv;
if (AGV_ACTIVO->agv->estado_alocacao==ALOCADO)
   MENS_ENV_SGAGV->mens.p2=ALOCA_AGV;
 else MENS_ENV_SGAGV->mens.p2=NAO_ALOCA_AGV;

sprintf(XX->buffer,
"\nSGM: mudanca do estado de alocacao do AGV %u, atraves do envio
de:\n",
AGV_ACTIVO->agv->id_agv);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(t, MENS_ENV_SGAGV->mens);

xx_setstate(15);
```

C.2.14.3 REC_MUD_EST_AGV

C.2.14.3.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==0 &&
  MENS_RECEB_SGAGV->mens.tipo_mens==ESTADO_AGV &&
  MENS_RECEB_SGAGV->mens.p1==AGV_ACTIVO->agv->id_agv &&
  MENS_RECEB_SGAGV->mens.p2!=ENABLED
 )
```

C.2.14.3.2 Acção

```
sprintf(XX->buffer,
"\nSGM: notificacao da alteracao do estado do AGV %u,\n atraves da
recepcao da mensagem:\n",
AGV_ACTIVO->agv->id_agv);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

AGV_ACTIVO->agv->estado_movim=MENS_RECEB_SGAGV->mens.p2;
AGV_ACTIVO->agv->estado_alocacao=MENS_RECEB_SGAGV->mens.p3;
AGV_ACTIVO->agv->estado_carga=MENS_RECEB_SGAGV->mens.p4;

xx_setstate(13);
C.2.13.4 REC_PASSAG_AGV_MANUAL
C.2.13.4.1 Predicado
(MENS_RECEB_SGAGV->mens.id_mens==0 &&
  MENS_RECEB_SGAGV->mens.tipo_mens==ESTADO_AGV &&
  MENS_RECEB_SGAGV->mens.p1==AGV_ACTIVO->agv->id_agv &&
  MENS_RECEB_SGAGV->mens.p2==ENABLED
 )
C.2.13.4.2 Acção
sprintf(XX->buffer,
"\nSGM: notificacao da passagem do AGV %u ao modo manual,\n atraves da
recepcao da mensagem:\n",
AGV_ENABLED->agv->id_agv);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

(XX->n_agvs_activos)--;
```

```
CARGA_ACTUAL->d=calc_carga(XX->n_tars, XX->n_agvs_activos);

AGV_ACTIVO->agv->estado_movim=MENS_RECEB_SGAGV->mens.p2;
AGV_ACTIVO->agv->estado_alocacao=MENS_RECEB_SGAGV->mens.p3;

xx_setstate(14);
```

C.2.15 Entidades da RP da página 'gere_AGVs_DISABLED'

C.2.15.1 Posições

- AGV_ESP_CONF_EN:AP_AGV;
- MENS_ENABLE_AGV.

C.2.15.2 Transições

C.2.15.2.1 ENVIA_ENABLE_AGV

C.2.15.2.1.1 Predicado

```
(AGV_DISABLED->agv->id_agv==FIM_TEMP_DIS_EN->i)
```

C.2.15.2.1.2 Acção

```
MENS_ENV_SGAGV->mens.id_mens=XX->id_prox_mens;
(XX->id_prox_mens)++;
MENS_ENV_SGAGV->mens.tipo_mens=ENABLE_AGV;
MENS_ENV_SGAGV->mens.p1=AGV_DISABLED->agv->id_agv;
if (rnd_binomial(XX->p_alocado))
   MENS_ENV_SGAGV->mens.p2=ALOCA_AGV;
 else MENS_ENV_SGAGV->mens.p2=NAO_ALOCA_AGV;

sprintf(XX->buffer,
"\nSGM: notifica o SGAGV da colocacao no estado ENABLED do AGV %u,
enviando:\n",
AGV_DISABLED->agv->id_agv);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGAGV->mens);

MENS_ENABLE_AGV->mens.id_mens=MENS_ENV_SGAGV->mens.id_mens;
MENS_ENABLE_AGV->mens.tipo_mens=MENS_ENV_SGAGV->mens.tipo_mens;
MENS_ENABLE_AGV->mens.p1=MENS_ENV_SGAGV->mens.p1;
MENS_ENABLE_AGV->mens.p2=MENS_ENV_SGAGV->mens.p2;

xx_setstate(7);
```

C.2.15.2.2 REC_CONF_INSUC_EN

C.2.15.2.2.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==MENS_ENABLE_AGV->mens.id_mens &&
  MENS_RECEB_SGAGV->mens.tipo_mens==CONFIRMACAO &&
  MENS_ENABLE_AGV->mens.p1==AGV_ESP_CONF_EN->agv->id_agv &&
  MENS_RECEB_SGAGV->mens.p1==INSUCESSO
```

```
)
```

C.2.15.2.2.2 Acção

```
double dt=rnd_exp_neg(XX->t_med_dis_en);
double t=xx_gettime();

AGV_DISABLED->agv->t_enable=t+dt;

TEMPO_DIS_EN->id=AGV_ESP_CONF_EN->agv->id_agv;
TEMPO_DIS_EN->t=dt;

sprintf(XX->buffer,
"\nSGM: AGV %u permanece no estado DISABLED apos a recepcao de:\n",
AGV_ESP_CONF_EN->agv->id_agv);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(t, MENS_RECEB_SGAGV->mens);

xx_setstate(9);
```

C.2.15.2.3 REC_CONF_SUC_EN

C.2.15.2.3.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==MENS_ENABLE_AGV->mens.id_mens &&
 MENS_RECEB_SGAGV->mens.tipo_mens==CONFIRMACAO &&
 MENS_ENABLE_AGV->mens.p1==AGV_ESP_CONF_EN->agv->id_agv &&
 MENS_RECEB_SGAGV->mens.p1==SUCESSO
)
```

C.2.15.2.3.2 Acção

```
double dt_en_dis=rnd_exp_neg(XX->t_med_en_dis);
double dt_mud_est_aloc=rnd_exp_neg(XX->t_med_mud_est_aloc);
double t=xx_gettime();

AGV_ESP_CONF_EN->agv->estado_movim=ENABLED;
if (MENS_ENABLE_AGV->mens.p2==ALOCA_AGV)
   AGV_ESP_CONF_EN->agv->estado_alocacao=ALOCADO;
  else AGV_ESP_CONF_EN->agv->estado_alocacao=NAO_ALOCADO;
AGV_ESP_CONF_EN->agv->t_disable=t+dt_en_dis;
AGV_ESP_CONF_EN->agv->t_mud_est_aloc=t+dt_mud_est_aloc;

TEMPO_EN_DIS->id=AGV_ESP_CONF_EN->agv->id_agv;
TEMPO_EN_DIS->t=dt_en_dis;

TEMPO_MUD_EST_ALOC->id=AGV_ESP_CONF_EN->agv->id_agv;
TEMPO_MUD_EST_ALOC->t=dt_mud_est_aloc;

sprintf(XX->buffer,
"\nSGM: AGV %u passa ao estado ENABLED apos a recepcao de:\n",
AGV_ESP_CONF_EN->agv->id_agv);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(t, MENS_RECEB_SGAGV->mens);

xx_setstate(8);
```

C.2.16 Transições da RP da página 'gere_AGVs_ENABLED'

C.2.16.1 DISABLE_AGV_ENABLED

C.2.16.1.1 Predicado

```
(AGV_ENABLED->agv->id_agv==FIM_TEMP_EN_DIS->i)
```

C.2.16.2 Acção

```
double dt=rnd_exp_neg(XX->t_med_dis_en);
double t=xx_gettime();

AGV_ENABLED->agv->estado_movim=DISABLED;
AGV_ENABLED->agv->t_enable=t+dt;

MENS_ENV_SGAGV->mens.id_mens=XX->id_prox_mens;
(XX->id_prox_mens)++;
MENS_ENV_SGAGV->mens.tipo_mens=DISABLE_AGV;
MENS_ENV_SGAGV->mens.p1=AGV_ENABLED->agv->id_agv;

INTERR_TEMP_MUD_EST_ALOC->i=AGV_ENABLED->agv->id_agv;

TEMPO_DIS_EN->id=AGV_ENABLED->agv->id_agv;
TEMPO_DIS_EN->t=dt;

sprintf(XX->buffer,
"\nSGM: AGV %u ENABLED passa ao estado DISABLED apos o envio de:\n",
AGV_ENABLED->agv->id_agv);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(t, MENS_ENV_SGAGV->mens);

xx_setstate(12);
```

C.2.16.2 MUDA_EST_ALOC_AGV_EN

C.2.16.2.1 Predicado

```
(AGV_ENABLED->agv->id_agv==FIM_TEMP_MUD_EST_ALOC->i)
```

C.2.16.2.2 Acção

```
double dt=rnd_exp_neg(XX->t_med_mud_est_aloc);
double t=xx_gettime();

if (AGV_ENABLED->agv->estado_alocacao==ALOCADO)
   AGV_ENABLED->agv->estado_alocacao=NAO_ALOCADO;
 else AGV_ENABLED->agv->estado_alocacao=ALOCADO;
AGV_ENABLED->agv->t_mud_est_aloc=t+dt;

TEMPO_MUD_EST_ALOC->id=AGV_ENABLED->agv->id_agv;
TEMPO_MUD_EST_ALOC->t=dt;

MENS_ENV_SGAGV->mens.id_mens=XX->id_prox_mens;
(XX->id_prox_mens)++;
```

```
MENS_ENV_SGAGV->mens.tipo_mens=ENABLE_AGV;
MENS_ENV_SGAGV->mens.p1=AGV_ENABLED->agv->id_agv;
if (AGV_ENABLED->agv->estado_alocacao==ALOCADO)
   MENS_ENV_SGAGV->mens.p2=ALOCA_AGV;
 else MENS_ENV_SGAGV->mens.p2=NAO_ALOCA_AGV;

sprintf(XX->buffer,
"\nSGM: mudanca do estado de alocacao do AGV %u, atraves do envio
de:\n",
AGV_ENABLED->agv->id_agv);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(t, MENS_ENV_SGAGV->mens);

xx_setstate(11);
```

C.2.16.3 REC_PASSAG_AGV_AUTOM

C.2.16.3.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==0 &&
  MENS_RECEB_SGAGV->mens.tipo_mens==ESTADO_AGV &&
  MENS_RECEB_SGAGV->mens.p1==AGV_ENABLED->agv->id_agv &&
  MENS_RECEB_SGAGV->mens.p2==PARADO
)
```

C.2.16.3.2 Acção

```
sprintf(XX->buffer,
"\nSGM: notificacao da passagem do AGV %u ao modo automatico,\n atraves
da recepcao da mensagem:\n",
AGV_ENABLED->agv->id_agv);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

(XX->n_agvs_activos)++;
CARGA_ACTUAL->d=calc_carga(XX->n_tars, XX->n_agvs_activos);

AGV_ENABLED->agv->estado_movim=MENS_RECEB_SGAGV->mens.p2;
AGV_ENABLED->agv->estado_alocacao=MENS_RECEB_SGAGV->mens.p3;
AGV_ENABLED->agv->estado_carga=MENS_RECEB_SGAGV->mens.p4;

xx_setstate(10);
```

C.2.17 Transições na RP da página 'gere_Tarefas'

C.2.17.1 PROC_T_POSIC_ESPEC

C.2.17.1.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==0 &&
 MENS_RECEB_SGAGV->mens.tipo_mens==T_POSIC_ESPEC
)
```

C.2.17.1.2 Acção

```
XX->id_prox_mens=MENS_RECEB_SGAGV->mens.p1+1;

TAREFA_PASSO1->tarefa=proc_t_posic_espec(
    xx_gettime(),
    &MENS_RECEB_SGAGV->mens,
    &XX->p_tarefas, &XX->u_tarefas,
    XX->p_agvs);

(XX->n_tars)++;
CARGA_ACTUAL->d=calc_carga(XX->n_tars, XX->n_agvs_activos);

xx_setstate(50);
```

C.2.18 Transições na RP da página 'gere_Tarefas_FILA'

C.2.18.1 REC_CANCEL_TAR_FILA

C.2.18.1.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==TAREFA_FILA->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.tipo_mens==TAREFA_CANCEL
)
```

C.2.18.1.2 Acção

```
if (TAREFA_FILA->tarefa->agv_tarefa!=NULL)
    TAREFA_FILA->tarefa->agv_tarefa->tarefa_agv=NULL;

sprintf(XX->buffer,
"\nSGM: notificacao do cancel. da Tarefa %u no estado FILA,\n atraves da
recepcao de:\n",
TAREFA_FILA->tarefa->id_tar);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

xx_setstate(25);
```

C.2.18.2 TAR_FILA_PASSO1

C.2.18.2.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==0 &&
 MENS_RECEB_SGAGV->mens.tipo_mens==ESTADO_TAREFA &&
 MENS_RECEB_SGAGV->mens.p1==TAREFA_FILA->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.p2==PASSO1
)
```

C.2.18.2.2 Acção

```
proc_estado_tarefa(xx_gettime(),
    &(MENS_RECEB_SGAGV->mens),
    TAREFA_FILA->tarefa, XX->p_agvs);
```

```
if (TAREFA_FILA->tarefa->tipo_tar==NORMAL)
    TAREFA_FILA->tarefa->agv_tarefa->ult_destino=
          TAREFA_FILA->tarefa->no2;

xx_setstate(23);
```

C.2.18.3 TAR_FILA_PASSO3

C.2.18.3.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==0 &&
 MENS_RECEB_SGAGV->mens.tipo_mens==ESTADO_TAREFA &&
 MENS_RECEB_SGAGV->mens.p1==TAREFA_FILA->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.p2==PASSO3
)
```

C.2.18.3.2 Acção

```
proc_estado_tarefa(xx_gettime(),
    &(MENS_RECEB_SGAGV->mens),
    TAREFA_FILA->tarefa, XX->p_agvs);

TAREFA_FILA->tarefa->agv_tarefa->ult_destino=
          TAREFA_FILA->tarefa->no2;

xx_setstate(24);
```

C.2.18.4 TAR_FILA_REJEITADA

C.2.18.4.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==TAREFA_FILA->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.tipo_mens==CONFIRMACAO &&
 MENS_RECEB_SGAGV->mens.p1==INSUCESSO)
```

C.2.17.4.2 Acção

```
if (TAREFA_FILA->tarefa->agv_tarefa!=NULL)
    TAREFA_FILA->tarefa->agv_tarefa->tarefa_agv=NULL;

 sprintf(XX->buffer,
"\nSGM: rejeicao da Tarefa %u apos a recepcao de:\n",
TAREFA_FILA->tarefa->id_tar);
 prn_string(XX->buffer);
 prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

 xx_setstate(26);
```

C.2.19 Transições da RP da página 'gere_Tarefas_PASSO1'

C.2.19.1 REC_CANCEL_TAR_PASSO1

C.2.19.1.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==TAREFA_PASSO1->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.tipo_mens==TAREFA_CANCEL
)
```

C.2.19.1.2 Acção

```
if (TAREFA_PASSO1->tarefa->agv_tarefa!=NULL)
    TAREFA_PASSO1->tarefa->agv_tarefa->tarefa_agv=NULL;

sprintf(XX->buffer,
"\nSGM: notificacao do cancel. da Tarefa %u no estado PASSO1,\n atraves
da recepcao de:\n",
TAREFA_PASSO1->tarefa->id_tar);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

xx_setstate(31);
```

C.2.19.2 REC_FIM_T_POSICION

C.2.19.2.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==TAREFA_PASSO1->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.tipo_mens==CONFIRMACAO &&
 MENS_RECEB_SGAGV->mens.p1==SUCESSO &&
 (TAREFA_PASSO1->tarefa->tipo_tar==POSICION ||
  TAREFA_PASSO1->tarefa->tipo_tar==POSIC_ESPEC)
)
```

C.2.19.2.2 Acção

```
if (TAREFA_PASSO1->tarefa->agv_tarefa!=NULL)
    TAREFA_PASSO1->tarefa->agv_tarefa->tarefa_agv=NULL;

sprintf(XX->buffer,
"\nSGM: notificacao do fim da Tarefa de Posicion. %u,\n atraves da
recepcao de:\n",
TAREFA_PASSO1->tarefa->id_tar);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

remove_Tarefa(TAREFA_PASSO1->tarefa,
    &XX->p_tarefas, &XX->u_tarefas);

(XX->n_tars)--;
CARGA_ACTUAL->d=calc_carga(XX->n_tars, XX->n_agvs_activos);

xx_setstate(30);
```

C.2.19.3 REC_REJEICAO_TAR_PASSO1

C.2.19.3.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==TAREFA_PASSO1->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.tipo_mens==CONFIRMACAO &&
 MENS_RECEB_SGAGV->mens.p1==INSUCESSO)
```

C.2.19.3.2 Acção

```
if (TAREFA_PASSO1->tarefa->agv_tarefa!=NULL)
    TAREFA_PASSO1->tarefa->agv_tarefa->tarefa_agv=NULL;

sprintf(XX->buffer,
"\nSGM: rejeicao da Tarefa %u apos a recepcao de:\n",
TAREFA_PASSO1->tarefa->id_tar);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

xx_setstate(32);
```

C.2.19.4 TAR_PASSO1_FILA

C.2.19.4.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==0 &&
 MENS_RECEB_SGAGV->mens.tipo_mens==ESTADO_TAREFA &&
 MENS_RECEB_SGAGV->mens.p1==TAREFA_PASSO1->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.p2==FILA
)
```

C.2.19.4.2 Acção

```
proc_estado_tarefa(xx_gettime(),
    &(MENS_RECEB_SGAGV->mens),
    TAREFA_PASSO1->tarefa, XX->p_agvs);

xx_setstate(28);
```

C.2.19.5 TAR_PASSO1_PASSO2

C.2.19.5.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==0 &&
 MENS_RECEB_SGAGV->mens.tipo_mens==ESTADO_TAREFA &&
 MENS_RECEB_SGAGV->mens.p1==TAREFA_PASSO1->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.p2==PASSO2
)
```

C.2.19.5.2 Acção

```
proc_estado_tarefa(xx_gettime(),
    &(MENS_RECEB_SGAGV->mens),
    TAREFA_PASSO1->tarefa, XX->p_agvs);
```

```
xx_setstate(27);
```

C.2.19.6 TAR_PASSO1_REAF

C.2.19.6.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==0 &&
 MENS_RECEB_SGAGV->mens.tipo_mens==ESTADO_TAREFA &&
 MENS_RECEB_SGAGV->mens.p1==TAREFA_PASSO1->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.p2==PASSO1 &&
 MENS_RECEB_SGAGV->mens.p3!=TAREFA_PASSO1->tarefa->agv_tarefa->id_agv
)
```

C.2.19.6.2 Acção

```
 sprintf(XX->buffer,
"\nSGM: notificacao da reafectacao da Tarefa %u, que estava afectada ao
AGV %u",
TAREFA_PASSO1->tarefa->id_tar,
TAREFA_PASSO1->tarefa->agv_tarefa->id_agv);
 prn_string(XX->buffer);

 proc_estado_tarefa(xx_gettime(),
     &MENS_RECEB_SGAGV->mens,
     TAREFA_PASSO1->tarefa, XX->p_agvs);

xx_setstate(29);
```

C.2.20 Transições da RP da página 'gere_Tarefas_PASSO2'

C.2.20.1 REC_CANCEL_TAR_PASSO2

C.2.20.1.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==TAREFA_PASSO2->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.tipo_mens==TAREFA_CANCEL
)
```

C.2.20.1.2 Acção

```
if (TAREFA_PASSO2->tarefa->agv_tarefa!=NULL)
    TAREFA_PASSO2->tarefa->agv_tarefa->tarefa_agv=NULL;

 sprintf(XX->buffer,
"\nSGM: notificacao do cancel. da Tarefa %u no estado PASSO2,\n atraves
da recepcao de:\n",
TAREFA_PASSO2->tarefa->id_tar);
 prn_string(XX->buffer);
 prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

 remove_Tarefa(TAREFA_PASSO2->tarefa,
    &XX->p_tarefas, &XX->u_tarefas);
```

```
(XX->n_tars)--;
CARGA_ACTUAL->d=calc_carga(XX->n_tars, XX->n_agvs_activos);

xx_setstate(35);
```

C.2.20.2 REC_FIM_T_CARGA

C.2.20.2.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==TAREFA_PASSO2->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.tipo_mens==CONFIRMACAO &&
 MENS_RECEB_SGAGV->mens.p1==SUCESSO &&
 (TAREFA_PASSO2->tarefa->tipo_tar==T_CARGA ||
  TAREFA_PASSO2->tarefa->tipo_tar==CARGA_AGV)
)
```

C.2.20.2.2 Acção

```
if (TAREFA_PASSO2->tarefa->agv_tarefa!=NULL)
    TAREFA_PASSO2->tarefa->agv_tarefa->tarefa_agv=NULL;

sprintf(XX->buffer,
"\nSGM: notificacao do fim da Tarefa de Carga %u,\n atraves da recepcao
de:\n",
TAREFA_PASSO2->tarefa->id_tar);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

remove_Tarefa(TAREFA_PASSO2->tarefa,
    &XX->p_tarefas, &XX->u_tarefas);

(XX->n_tars)--;
CARGA_ACTUAL->d=calc_carga(XX->n_tars, XX->n_agvs_activos);

xx_setstate(34);
```

C.2.20.3 TAR_PASSO2_PASSO3

C.2.20.3.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==0 &&
 MENS_RECEB_SGAGV->mens.tipo_mens==ESTADO_TAREFA &&
 MENS_RECEB_SGAGV->mens.p1==TAREFA_PASSO2->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.p2==PASSO3
)
```

C.2.20.3.2 Acção

```
proc_estado_tarefa(xx_gettime(),
    &(MENS_RECEB_SGAGV->mens),
    TAREFA_PASSO2->tarefa, XX->p_agvs);

xx_setstate(33);
```

C.2.21 Transições da RP da página 'gere_Tarefas_PASSO3'

C.2.21.1 REC_CANCEL_TAR_PASSO3

C.2.21.1.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==TAREFA_PASSO3->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.tipo_mens==TAREFA_CANCEL
)
```

C.2.21.1.2 Acção

```
if (TAREFA_PASSO3->tarefa->agv_tarefa!=NULL)
    TAREFA_PASSO3->tarefa->agv_tarefa->tarefa_agv=NULL;

sprintf(XX->buffer,
    "\nSGM: notificacao do cancel. da Tarefa %u no estado PASSO3,\n
atraves da recepcao de:\n",
    TAREFA_PASSO3->tarefa->id_tar);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

remowe_Tarefa(TAREFA_PASSO3->tarefa,
    &XX->p_tarefas, &XX->u_tarefas);

(XX->n_tars)--;
CARGA_ACTUAL->d=calc_carga(XX->n_tars, XX->n_agvs_activos);

xx_setstate(37);
```

C.2.21.2 TAR_PASSO3_PASSO4

C.2.21.2.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==0 &&
 MENS_RECEB_SGAGV->mens.tipo_mens==ESTADO_TAREFA &&
 MENS_RECEB_SGAGV->mens.p1==TAREFA_PASSO3->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.p2==PASSO4
)
```

C.2.21.2.2 Acção

```
proc_estado_tarefa(xx_gettime(),
    &(MENS_RECEB_SGAGV->mens),
    TAREFA_PASSO3->tarefa, XX->p_agvs);

xx_setstate(36);
```

C.2.22 Transições da RP da página 'gere_Tarefas_PASSO4'

C.2.22.1 REC_CANCEL_TAR_PASSO4

C.2.22.1.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==TAREFA_PASSO4->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.tipo_mens==TAREFA_CANCEL
)
```

C.2.22.1.2 Acção

```
if (TAREFA_PASSO4->tarefa->agv_tarefa!=NULL)
   TAREFA_PASSO4->tarefa->agv_tarefa->tarefa_agv=NULL;

dsprintf(XX->buffer,
"\nSGM: notificacao do cancel. da Tarefa %u no estado PASSO4,\n atraves
da recepcao de:\n",
TAREFA_PASSO4->tarefa->id_tar);
 prn_string(XX->buffer);
 prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

 remove_Tarefa(TAREFA_PASSO4->tarefa,
    &XX->p_tarefas, &XX->u_tarefas);

 (XX->n_tars)--;
 CARGA_ACTUAL->d=calc_carga(XX->n_tars, XX->n_agvs_activos);

 xx_setstate(39);
```

C.2.22.2 REC_FIM_T_NORMAL_OU_DESCARGA

C.2.22.2.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==TAREFA_PASSO4->tarefa->id_tar &&
 MENS_RECEB_SGAGV->mens.tipo_mens==CONFIRMACAO &&
 MENS_RECEB_SGAGV->mens.p1==SUCESSO
)
```

C.2.22.2.2 Acção

```
if (TAREFA_PASSO4->tarefa->agv_tarefa!=NULL)
   TAREFA_PASSO4->tarefa->agv_tarefa->tarefa_agv=NULL;

 sprintf(XX->buffer,
"\nSGM: notificacao do fim da Tarefa Normal ou de Descarga %u,\n atraves
da recepcao de:\n",
TAREFA_PASSO4->tarefa->id_tar);
 prn_string(XX->buffer);
 prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

 remove_Tarefa(TAREFA_PASSO4->tarefa,
    &XX->p_tarefas, &XX->u_tarefas);

 (XX->n_tars)--;
```

```
CARGA_ACTUAL->d=calc_carga(XX->n_tars, XX->n_agvs_activos);

xx_setstate(38);
```

C.2.23 Entidades da RP da página 'gere_cancelamento_Tarefas'

C.2.23.1 Posições

- ESPERA_CANCEL_TAREFAS:NUL;
- FIM_CANCEL_TAREFAS:NUL;
- ID_MENS_CANCEL_TAR:ULINT;
- TIPO_CANCEL:ULINT.

C.2.23.2 Transições

C.2.23.2.1 CANCELA_1_TAR

C.2.23.2.1.1 Predicado

```
(XX->n_tars>0 && TIPO_CANCEL->i==UMA_TAREFA)
```

C.2.23.2.1.2 Acção

```
MENS_ENV_SGAGV->mens.id_mens=XX->id_prox_mens;
ID_MENS_CANCEL_TAR->i=XX->id_prox_mens;
(XX->id_prox_mens)++;
MENS_ENV_SGAGV->mens.tipo_mens=CANCELA_TAREFA;
MENS_ENV_SGAGV->mens.p1=UMA_TAREFA;
MENS_ENV_SGAGV->mens.p2=decide_tar_a_cancelar(
   XX->p_tarefas, XX->n_tars);

sprintf(XX->buffer,
"\nSGM: inicio do cancelamento da tarefa %u atraves do envio de:\n",
MENS_ENV_SGAGV->mens.p2);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGAGV->mens);

xx_setstate(44);
```

C.2.23.2.2 CANCELA_TAR_EXEC

C.2.23.2.2.1 Predicado

```
(XX->n_tars>0 && TIPO_CANCEL->i==TODAS_TAR_EXEC)
```

C.2.23.2.2.2 Acção

```
MENS_ENV_SGAGV->mens.id_mens=XX->id_prox_mens;
ID_MENS_CANCEL_TAR->i=XX->id_prox_mens;
(XX->id_prox_mens)++;
MENS_ENV_SGAGV->mens.tipo_mens=CANCELA_TAREFA;
MENS_ENV_SGAGV->mens.p1=TODAS_TAR_EXEC;
```

```
prn_string("\nSGM: inicio do cancelamento das tarefas em execucao
atraves do envio de:\n");
prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGAGV->mens);

xx_setstate(45);
```

C.2.23.2.3 CANCELA_TODAS_TAR

C.2.23.2.3.1 Predicado

```
(XX->n_tars>0 && TIPO_CANCEL->i==TODAS_TAR)
```

C.2.23.2.3.2 Acção

```
MENS_ENV_SGAGV->mens.id_mens=XX->id_prox_mens;
ID_MENS_CANCEL_TAR->i=XX->id_prox_mens;
(XX->id_prox_mens)++;
MENS_ENV_SGAGV->mens.tipo_mens=CANCELA_TAREFA;
MENS_ENV_SGAGV->mens.p1=TODAS_TAR;

prn_string("\nSGM: inicio do cancelamento de todas as tarefas atraves
do envio de:\n");
prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGAGV->mens);

xx_setstate(46);
```

C.2.23.2.4 Acção de 'INICIA_CANCEL_TAREFAS'

```
TIPO_CANCEL->i=decide_tipo_cancel_tar(XX->p_cancel_1_tar,
    XX->p_cancel_tar_exec);

xx_setstate(43);
```

C.2.23.2.5 Acção de 'NAO_CANCELA_TAREFAS'

```
xx_setstate(47);
```

C.2.23.2.6 Acção de 'PLANEIA_NOVO_CANCEL_TAREFAS'

```
xx_fdelay(XT_INICIA_CANCEL_TAREFAS,
    rnd_exp_neg(XX->t_med_cancel_tar));

xx_setstate(49);
```

C.2.23.2.7 REC_CONF_CANCEL_TAR

C.2.23.2.7.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_mens==ID_MENS_CANCEL_TAR->i &&
 MENS_RECEB_SGAGV->mens.tipo_mens==CONFIRMACAO
)
```

C.2.23.2.7.2 Acção

```
prn_string("\nSGM: notificacao do resultado do cancelamento de
tarefas,\n atraves da recepcao de:\n");
```

```
prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

xx_setstate(48);
```

C.2.24 Entidades da RP da página 'lanca_Tarefas'

C.2.24.1 Posições

- AGV_TAREFA:AP_AGV;
- CARGA_PRETENDIDA:DOUBLE;
- CARGA_ULTIMA_DECISAO:DOUBLE.

C.2.24.2 Transições

C.2.24.2.1 AVALIA_CARGA_PRETENDIDA

C.2.24.2.1.1 Predicado

```
((CARGA_ULTIMA_DECISAO->d!=CARGA_ACTUAL->d ||
  CARGA_ACTUAL->d==0.0) &&
 existe_AGV_nova_tarefa(XX->p_agvs, XX->p_esc_sgm)
)
```

C.2.24.2.1.2 Acção

```
CARGA_PRETENDIDA->d=rnd_normal(XX->vm_carga, XX->dp_carga);

xx_setstate(17);
```

C.2.24.2.2 INICIA_LANCAM_TAREFA_ESC_SGM

C.2.24.2.2.1 Predicado

```
(CARGA_ACTUAL->d < CARGA_PRETENDIDA->d)
```

C.2.24.2.2.2 Acção

```
AGV_TAREFA->agv=deter_AGV_nova_tarefa(XX->p_agvs);

CARGA_ULTIMA_DECISAO->d=CARGA_ACTUAL->d;

xx_setstate(18);
```

C.2.24.2.3 Acção de 'LANCA_NOVA_TAREFA_NORMAL'

```
Tarefa tar;

MENS_ENV_SGAGV->mens.id_mens=XX->id_prox_mens;
tar.id_tar=XX->id_prox_mens;
(XX->id_prox_mens)++;

tar.tipo_tar=NORMAL;
```

```
MENS_ENV_SGAGV->mens.tipo_mens=TAREFA_NORMAL;
MENS_ENV_SGAGV->mens.p1=QUALQUER;
MENS_ENV_SGAGV->mens.p4=deter_prior_tar(XX->p_tar_prior,
    XX->priorid_tar_max);
tar.prior=MENS_ENV_SGAGV->mens.p4;
tar.agv_tarefa=NULL;

MENS_ENV_SGAGV->mens.p2=NOVA_TAREFA_NORMAL->origem;
tar.no1=MENS_ENV_SGAGV->mens.p2;
MENS_ENV_SGAGV->mens.p3=NOVA_TAREFA_NORMAL->destino;
tar.no2=MENS_ENV_SGAGV->mens.p3;

tar.estado_tar=FILA;

insere_Tarefa(&tar, &XX->p_tarefas, &XX->u_tarefas);

TAREFA_FILA->tarefa=XX->u_tarefas;

prn_string("\nSGM: lancamento de uma nova tarefa normal atraves do
envio de:\n");
prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGAGV->mens);

(XX->n_tars)++;
CARGA_ACTUAL->d=calc_carga(XX->n_tars, XX->n_agvs_activos);

xx_setstate(22);
```

C.2.24.2.4 Acção de 'LANCA_TAREFA_ESC_SGM'

```
Tarefa tar;

MENS_ENV_SGAGV->mens.id_mens=XX->id_prox_mens;
tar.id_tar=XX->id_prox_mens;
(XX->id_prox_mens)++;

if (AGV_TAREFA->agv->estado_carga==CARREGADO)
    tar.tipo_tar=T_DESCARGA;
    else tar.tipo_tar=POSICION;

switch (tar.tipo_tar)
{
 case POSICION:
  MENS_ENV_SGAGV->mens.tipo_mens=TAREFA_POSICION;
  MENS_ENV_SGAGV->mens.p2=decide_no_tarefa(XX->nos_tarefas);
  break;
 case T_DESCARGA:
  MENS_ENV_SGAGV->mens.tipo_mens=TAREFA_DESCARGA;
  MENS_ENV_SGAGV->mens.p2=AGV_TAREFA->agv->ult_destino;
}

MENS_ENV_SGAGV->mens.p1=AGV_TAREFA->agv->id_agv;

tar.prior=0;
tar.agv_tarefa=AGV_TAREFA->agv;

if (tar.tipo_tar!=T_DESCARGA)
    tar.no1=MENS_ENV_SGAGV->mens.p2;
else tar.no2=MENS_ENV_SGAGV->mens.p2;
```

```
tar.estado_tar=FILA;

insere_Tarefa(&tar, &XX->p_tarefas, &XX->u_tarefas);

TAREFA_FILA->tarefa=XX->u_tarefas;
AGV_TAREFA->agv->tarefa_agv=XX->u_tarefas;

prn_string("\nSGM: lancamento de uma nova tarefa atraves do envio
de:\n");
prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGAGV->mens);

(XX->n_tars)++;
CARGA_ACTUAL->d=calc_carga(XX->n_tars, XX->n_agvs_activos);

xx_setstate(20);
```

C.2.24.2.5 Acção de 'NAO_LANCA_TAREFA_ESC_SGM'

```
CARGA_ULTIMA_DECISAO->d=CARGA_ACTUAL->d;

xx_setstate(19);
```

C.2.25 Entidades da RP da página 'leitura_fich_nos_tar'

C.2.25.1 Posições

- FIM_LEITURA_FICH_NOS_TAREFAS:NUL;
- INICIO_CONFIG_SGM:NUL;
- NOS_TAREFAS:AP_NOS_TAREFAS;
- RES_VER_SOM_PROB_NOS_TAREFAS:BOOLEANO.

C.2.25.2 Transições

C.2.25.2.1 Acção de 'LE_FICH_NOS_TAREFAS'

```
XX->nos_tarefas=le_f_nos_tarefas(XX->nome_fich_nos_tarefas);
NOS_TAREFAS->ap=XX->nos_tarefas;

xx_setstate(1);
```

C.2.25.2.2 TERMINA_CONFIG_SGM

C.2.25.2.2.1 Predicado

```
(RES_VER_SOM_PROB_NOS_TAREFAS->b!=FALSO)
```

C.2.25.2.2.2 Acção

```
xx_setstate(3);
```

C.2.25.2.3 VERIF_SOMAT_PROB_NOS_TAREFAS

C.2.25.2.3.1 Predicado

```
(XX->nos_tarefas!=NULL)
```

C.2.25.2.3.2 Acção

```
RES_VER_SOM_PROB_NOS_TAREFAS->b=
    som_prob_nos_unit(XX->nos_tarefas);

xx_setstate(2);
```

C.2.26 Entidades da RP da página 'proc_tar_canc_FILA_ou_PASSO1'

C.2.26.1 Posição 'TAREFA_A_REMOVER:AP_TAREFA'

C.2.26.2 Transições

C.2.26.2.1 LIBERTA_PONTO_CARGA

C.2.26.2.1.1 Predicado

```
(TAREFA_CANCELADA->tarefa->tipo_tar==NORMAL)
```

C.2.26.2.1.2 Acção

```
PONTO_CARGA_DISPONIVEL->i=TAREFA_CANCELADA->tarefa->no1;

xx_setstate(40);
```

C.2.26.2.2 Acção de 'REMOVE_TAREFA'

```
remove_Tarefa(TAREFA_A_REMOVER->tarefa,
    &XX->p_tarefas, &XX->u_tarefas);

(XX->n_tars)--;
CARGA_ACTUAL->d=calc_carga(XX->n_tars, XX->n_agvs_activos);

xx_setstate(42);
```

C.2.26.2.3 Acção de TAREFA_CANCELADA_NAO_NORMAL

```
xx_setstate(41);
```

C.3 Classe SGAGV

C.3.1 CMN units incluídas

- BOOLEANO;

- MENS_SGM_SGAGV;
- MENS_SGAGV_SPAGV;
- ULINT;
- CAPAC_FILA.

C.3.2 Posições de entrada

- MENS_RECEB_SGM:MENS_SGM_SGAGV;
- MENS_RECEB_SPAGV:MENS_SGAGV_SPAGV;
- FILA_PALETE_INSERIDA:ULINT;
- CAPACIDADE_FILA:CAPAC_FILA.

C.3.3 Posições de saída

- MENS_ENV_SGM:MENS_SGM_SGAGV;
- MENS_ENV_SPAGV:MENS_SGAGV_SPAGV.

C.3.4 Parâmetros

- vel_frente:unsigned long int – velocidade média de deslocamento de um AGV em movimento rectilíneo para a frente (em cm/s);
- vel_curva:unsigned long int – velocidade média de deslocamento de um AGV em movimento curvilíneo para a frente (em cm/s);
- vel_tras:unsigned long int – velocidade média de deslocamento de um AGV em movimento rectilíneo para trás (em cm/s);
- nome_fich_mesas: xx_string – nome do ficheiro de configuração dos pontos de carga/descarga, com a posição relativa destes em relação aos nós do layout respectivos;
- numero_agvs:unsigned long int – número máximo de AGVs no sistema;
- poss_acesso:signed int – define as possibilidades de acesso aos pontos de carga/descarga por parte dos AGVs; vale 0 se puderem operar para ambos os lados, vale +1 se só puderem operar para o lado esquerdo, ou vale -1 se só puderem operar para o lado direito;
- num_max_af:unsigned long int – número máximo de afectações realizadas numa chamada ao algoritmo de escalonamento;
- num_max_mov_avan:unsigned long int – número máximo de movimentos em avanço que podem ser enviados a um AGV no estado de movimentação EX_MOVIM;
- reaf_dinam:signed int – vale 0 se o algoritmo de escalonamento puder implementar o mecanismo de reafectação dinâmica; este parâmetro só é relevante quando os parâmetros 'regra_ADCT' e 'regra_ADV' valerem ambos zero, ou seja, quando o algoritmo de escalonamento a utilizar for o algoritmo húngaro;

- regra_ADCT:signed int – define a regra de despacho a utilizar numa Afectação Desencadeada por um Centro de Trabalho (pode tomar o valor definido pelas constantes VMP e VMU, declaradas no ficheiro de cabeçalho 'sgagv.h' (ver página 278);
- regra_ADV:signed int – define a regra de despacho a utilizar numa Afectação Desencadeada por um Veículo (pode tomar o valor definido pelas constantes TMCV, MTFS, MELFS e PCPSM, declaradas no ficheiro de cabeçalho 'sgagv.h' (ver página 278);

C.3.5 Ficheiros de cabeçalho incluídos

```
#include "listas.h"
#include "mens_log.h"
#include "fun_aux.h"
#include "layout.h"
#include "sgagv.h"
#include "sistagvs.h"
```

C.3.6 Funções definidas internamente

```
/* funcao para permitir a visualizacao do layout, a partir de
uma posicao que contenha um token do tipo AP_LAYOUT
(utilizando o comando "examine Queue") */
void prn_AP_LAY(a)
 AP_LAYOUT *a;
{ prn_layout(a->ap); }

/* funcao para permitir a visualizacao do valor da variavel
booleana contida num token do tipo BOOLEANO
(utilizando o comando "examine Queue") */
void prn_BOOLEANO_SGA(bool)
 BOOLEANO *bool;
{if (bool->b==FALSO) prn_string("FALSO\n"); else
prn_string("VERDADE\n");}

/* funcao para permitir a visualizacao da informacao contida numa
lista de AGV's apontada por um token do tipo L_AGVS
(utilizando o comando "examine Queue") */
void prn_L_AGVS(a)
 L_AGVS *a;
{prn_AGVs(a->ap);}

/* funcao para permitir a visualizacao da informacao de um AGV
apontado por um token do tipo APO_AGV
(utilizando o comando "examine Queue") */
void prn_APO_AGV(a)
 APO_AGV *a;
{prn_el_l_AGVs(a->ap);}

/* funcao para permitir a visualizacao do conteudo de uma
mensagem SGM <-> SGAGV, a partir de uma posicao que contenha
um token do tipo MENS_SGM_SGAGV
(utilizando o comando "examine Queue") */
void prn_MENS_SGAGV_SGM(a)
  MENS_SGM_SGAGV *a;
{ prn_m_SGM_SGAGV(a->mens); }

/* funcao para permitir a visualizacao do conteudo de uma
mensagem SGAGV <-> SPAGV, a partir de uma posicao que contenha
um token do tipo MENS_SGAGV_SPAGV
(utilizando o comando "examine Queue") */
```

```
void prn_MENS_SGAGV_SPAGV(a)
  MENS_SGAGV_SPAGV *a;
{ prn_m_SGAGV_SPAGV(a->mens); }

/* funcao para permitir a visualizacao do conteudo de uma lista de
bloqueios, a partir de uma posicao que contenha um token do tipo
L_BLOQ (utilizando o comando "examine Queue") */
void prn_L_BLOQ(a)
  L_BLOQ *a;
{ prn_l_bloqueios(a->ap); }

/* funcao para permitir a visualizacao do conteudo de uma lista de
tarefas, a partir de uma posicao que contenha um token do tipo
L_TAREFAS (utilizando o comando "examine Queue") */
void prn_L_TAREFAS(a)
  L_TAREFAS *a;
{ prn_tarefas(a->ap); }

/* funcao para permitir a visualizacao da informacao de uma Tarefa
apontado por um token do tipo APO_TAREFA
(utilizando o comando "examine Queue") */
void prn_APO_TAREFA(a)
 APO_TAREFA *a;
{prn_el_l_tarefas(a->ap);}

/* funcao para permitir a visualizacao da informacao de uma Tarefa
continda num token do tipo TAREFA
(utilizando o comando "examine Queue") */
void prn_TAREFA(a)
 TAREFA *a;
{prn_el_l_tarefas(&a->tarefa);}

/* funcao para permitir a visualizacao do conteudo de uma lista de
apontadores para Tarefas, a partir de uma posicao que contenha um token
do tipo L_AP_TAREFAS (utilizando o comando "examine Queue") */
void prn_L_AP_TAREFAS(a)
  L_AP_TAREFAS *a;
{ prn_l_ap_Tarefas(a->ap); }

/* funcao para permitir a visualizacao de um token do tipo
ULINT (utilizando o comando "examine Queue") */
void prn_ULINT_SGA(a)
 ULINT *a;
{
  char s[255];

  sprintf(s, "(ulint) i=%u\n", a->i);
  prn_string(s);
}

/* funcao para permitir a visualizacao do conteudo de uma lista de
apontadores para AGV's, a partir de uma posicao que contenha um token
do tipo L_AP_AGVS (utilizando o comando "examine Queue") */
void prn_L_AP_AGVS(a)
  L_AP_AGVS *a;
{ prn_l_ap_AGVs(a->ap); }

/* funcao para permitir a visualizacao do conteudo de uma lista de
elementos que representam o estado de filas de carga, a partir de
uma posicao que contenha um token do tipo L_ESTADO_FILAS
(utilizando o comando "examine Queue") */
void prn_L_ESTADO_FILAS(a)
  L_ESTADO_FILAS *a;
{ prn_l_EstadoFila(a->ap); }
```

C.3.7 Acções iniciais

```
char s1[30], s2[15];
int i;
ap_no_l_int u;

/* Examine Queue em posicoes de tipos nao geridos automatica/ */
XW_AP_LAYOUT_print = prn_AP_LAY;
XW_BOOLEANO_print = prn_BOOLEANO_SGA;
XW_L_AGVS_print = prn_L_AGVS;
XW_APO_AGV_print = prn_APO_AGV;
XW_MENS_SGM_SGAGV_print = prn_MENS_SGAGV_SGM;
XW_MENS_SGAGV_SPAGV_print=prn_MENS_SGAGV_SPAGV;
XW_L_BLOQ_print = prn_L_BLOQ;
XW_L_TAREFAS_print = prn_L_TAREFAS;
XW_APO_TAREFA_print = prn_APO_TAREFA;
XW_TAREFA_print = prn_TAREFA;
XW_L_AP_TAREFAS_print = prn_L_AP_TAREFAS;
XW_ULINT_print = prn_ULINT_SGA;
XW_L_AP_AGVS_print = prn_L_AP_AGVS;
XW_L_ESTADO_FILAS_print = prn_L_ESTADO_FILAS;

/* inicializa o estado do objecto */
xx_setstate(0);

/* inicializa id_max_mens_SGM */
XX->id_prox_mens_SGM=0;

/* inicializa o apontador para a lista que representa o Layout */
XX->layout=NULL;

/* inicializa os apontadores da lista de AGV's */
XX->p_agvs=NULL;
XX->u_agvs=NULL;

/* inicializa os apontadores da lista de Bloqueios */
XX->p_bloqueios=NULL;
XX->u_bloqueios=NULL;

/* inicializa os apontadores da lista de Tarefas */
XX->p_tarefas=NULL;
XX->u_tarefas=NULL;

/* inicializa os apontadores da lista de elementos que
representam o estado das filas de carga */
XX->p_estado_filas=NULL;
XX->u_estado_filas=NULL;

/* declaracoes de medidas relacionadas com a execucao de
Tarefas do tipo Normal */
XX->ve_dt_fila=xx_initmeasure("dt_fila");
XX->ve_dt_p1=xx_initmeasure("dt_p1");
XX->ve_dt_p2=xx_initmeasure("dt_p2");
XX->ve_dt_p3=xx_initmeasure("dt_p3");
XX->ve_dt_p4=xx_initmeasure("dt_p4");
XX->ve_num_af_tar=xx_initmeasure("num_af");
XX->ve_t_oper_tar=xx_initmeasure("t_oper_tar");
XX->ve_t_exec_tar=xx_initmeasure("t_exec_tar");
```

```
/* inicializacao dos contadores de tarefas */
XX->num_tar_NORMAL_lan=0;
XX->num_tar_NORMAL_ex=0;
XX->num_tar_NORMAL_AGV_lan=0;
XX->num_tar_NORMAL_AGV_ex=0;
XX->num_tar_POSICION_lan=0;
XX->num_tar_POSICION_ex=0;
XX->num_tar_T_CARGA_lan=0;
XX->num_tar_T_CARGA_ex=0;
XX->num_tar_CARGA_AGV_lan=0;
XX->num_tar_CARGA_AGV_ex=0;
XX->num_tar_T_DESCARGA_lan=0;
XX->num_tar_T_DESCARGA_ex=0;
XX->num_tar_POSIC_ESPEC_lan=0;
XX->num_tar_POSIC_ESPEC_ex=0;
XX->num_tar_lan=0;
XX->num_tar_rej=0;
XX->num_tar_cancel=0;
XX->num_tar_ex=0;

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com a contagem de
Tarefas lancadas */
XX->l_ve_contag_tar_lan=NULL;
u=NULL;
for (i=0;i<7;i++)
/* tipos de Tarefas: 0..6 */
{
 escr_str_tipo_tarefa(s2, i);
 sprintf(s1, "num_tar_%s_lan", s2);
 insere_int(xx_initmeasure(s1),
    &XX->l_ve_contag_tar_lan, &u);
 xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com a contagem de
Tarefas executadas */
XX->l_ve_contag_tar_ex=NULL;
u=NULL;
for (i=0;i<7;i++)
/* tipos de Tarefas: 0..6 */
{
 escr_str_tipo_tarefa(s2, i);
 sprintf(s1, "num_tar_%s_ex", s2);
 insere_int(xx_initmeasure(s1),
    |Xx->l_ve_contag_tar_ex, &u);
 xx_writesample(u->i, 0.0);
}

/* declaracoes de medidas relacionadas com Totais de
contagens de Tarefas e sua inicializacao */
XX->ve_num_tar_lan=xx_initmeasure("num_tar_lan");
xx_writesample(XX->ve_num_tar_lan, 0.0);
XX->ve_num_tar_rej=xx_initmeasure("num_tar_rej");
xx_writesample(XX->ve_num_tar_rej, 0.0);
XX->ve_num_tar_cancel=xx_initmeasure("num_tar_cancel");
xx_writesample(XX->ve_num_tar_cancel, 0.0);
XX->ve_num_tar_ex=xx_initmeasure("num_tar_ex");
```

```
xx_writesample(XX->ve_num_tar_ex, 0.0);

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Estado dos AGV's */
XX->l_ve_estado_AGV=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "estado_AGV%d", i);
   insere_int(xx_initmeasure(s1), &XX->l_ve_estado_AGV, &u);
   xx_writesample(u->i, DISABLED);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Tempo de
Permanencia dos AGV's no estado DISABLED */
XX->l_ve_dt_disabled=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "dt_disabled_AGV%d", i);
   insere_int(xx_initmeasure(s1), &XX->l_ve_dt_disabled, &u);
   xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Tempo de
Permanencia dos AGV's no estado ENABLED */
XX->l_ve_dt_enabled=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "dt_enabled_AGV%d", i);
   insere_int(xx_initmeasure(s1), &XX->l_ve_dt_enabled, &u);
   xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Tempo de
Permanencia dos AGV's no estado PARADO */
XX->l_ve_dt_espera=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "dt_espera_AGV%d", i);
   insere_int(xx_initmeasure(s1), &XX->l_ve_dt_espera, &u);
   xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Tempo de
Permanencia dos AGV's no estado BLOQUEADO */
XX->l_ve_dt_bloqueado=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "dt_bloqueado_AGV%d", i);
   insere_int(xx_initmeasure(s1), &XX->l_ve_dt_bloqueado, &u);
   xx_writesample(u->i, 0.0);
```

```
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Tempo de
Permanencia dos AGV's no estado EX_MOVIM/DESCARREGADO */
XX->l_ve_dt_movim_descarr=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "dt_movim_descarr_AGV%d", i);
   insere_int(xx_initmeasure(s1), &XX->l_ve_dt_movim_descarr, &u);
   xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Tempo de
Permanencia dos AGV's no estado EX_MOVIM/CARREGADO */
XX->l_ve_dt_movim_carr=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "dt_movim_carr_AGV%d", i);
   insere_int(xx_initmeasure(s1), &XX->l_ve_dt_movim_carr, &u);
   xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Tempo de
Permanencia dos AGV's no estado CARGA_AGV */
XX->l_ve_dt_carga=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "dt_carga_AGV%d", i);
   insere_int(xx_initmeasure(s1), &XX->l_ve_dt_carga, &u);
   xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Tempo de
Permanencia dos AGV's no estado DESCARGA_AGV */
XX->l_ve_dt_descarga=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "dt_descarga_AGV%d", i);
   insere_int(xx_initmeasure(s1), &XX->l_ve_dt_descarga, &u);
   xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Tempo de
Permanencia dos AGV's no estado ERRO */
XX->l_ve_dt_erro=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "dt_erro_AGV%d", i);
   insere_int(xx_initmeasure(s1), &XX->l_ve_dt_erro, &u);
```

```
    xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Numero de
Arranques DESCARREGADO */
XX->l_ve_n_arr_descarr=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "n_arr_descarr_AGV%d", i);
   insere_int(xx_initmeasure(s1), &XX->l_ve_n_arr_descarr, &u);
   xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Numero de
Arranques CARREGADO */
XX->l_ve_n_arr_carr=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "n_arr_carr_AGV%d", i);
   insere_int(xx_initmeasure(s1), &XX->l_ve_n_arr_carr, &u);
   xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Numero de
Movimentos DESCARREGADO */
XX->l_ve_n_mov_descarr=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "n_mov_descarr_AGV%d", i);
   insere_int(xx_initmeasure(s1), &XX->l_ve_n_mov_descarr, &u);
   xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Numero de
Movimentos CARREGADO */
XX->l_ve_n_mov_carr=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "n_mov_carr_AGV%d", i);
   insere_int(xx_initmeasure(s1), &XX->l_ve_n_mov_carr, &u);
   xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Numero de
Operacoes de Carga */
XX->l_ve_n_cargai=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
   sprintf(s1, "n_cargas_AGV%d", i);
```

```
    insere_int(xx_initmeasure(s1), &XX->l_ve_n_cargas, &u);
    xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o Numero de
Operacoes de Descarga */
XX->l_ve_n_descargas=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
    sprintf(s1, "n_descargas_AGV%d", i);
    insere_int(xx_initmeasure(s1), &XX->l_ve_n_descargas, &u);
    xx_writesample(u->i, 0.0);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com a Taxa de Utilizacao
dos AGV's */
XX->l_ve_taxa_util=NULL;
u=NULL;
for (i=1;i<=XX->numero_agvs;i++)
  {
    sprintf(s1, "taxa_util_AGV%d", i);
    insere_int(xx_initmeasure(s1), &XX->l_ve_taxa_util, &u);
    xx_writesample(u->i, 100.0);
}
```

C.3.8 Acções finais

```
ap_AGV agv=XX->p_agvs;
double t=xx_gettime();

/* actualiza as estatisticas temporais dos AGV's */
while (agv!=NULL)
{
  act_estat_temporais_agv(agv, t);

  xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_disabled),
    agv->dt_disabled);
  xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_enabled),
    agv->dt_enabled);
  xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_espera),
    agv->dt_espera);
  xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_bloqueado),
    agv->dt_bloqueado);
  xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_movim_descarr),
    agv->dt_movim_descarr);
  xx_wrmtesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_movim_carr),
    agv->dt_movim_carr);
  xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_carga),
```

```
      agv->dt_carga);
    xx_writesample(
      obtem_int(agv->id_agv, XX->l_ve_dt_descarga),
      agv->dt_descarga);
    xx_writesample(
      obtem_int(agv->id_agv, XX->l_ve_dt_erro),
      agv->dt_erro);
    xx_writesample(
      obtem_int(agv->id_agv, XX->l_ve_taxa_util),
      agv->taxa_util);

    agv=agv->seguinte;
  }

  /* destroi a lista que representa o Layout */
  XX->layout=limpa_layout(XX->layout);

  /* destroi a lista de AGV's */
  limpa_lista_AGVs(&XX->p_agvs);

  /* destroi a lista de Bloqueios */
  limpa_l_bloqueios(&XX->p_bloqueios);

  /* destroi a lista de Tarefas */
  limpa_lista_tarefas(&XX->p_tarefas);

  /* destroi a lista de elementos que representam o
  estado das filas de carga */
  limpa_l_EstadoFila(&XX->p_estado_filas);

  /* destroi as listas de identificadores de var. estat. */
  limpa_l_int(&XX->l_ve_contag_tar_lan);
  limpa_l_int(&XX->l_ve_contag_tar_ex);
  limpa_l_int(&XX->l_ve_estado_AGV);
  limpa_l_int(&XX->l_ve_dt_bloqueado);
  limpa_l_int(&XX->l_ve_dt_carga);
  limpa_l_int(&XX->l_ve_dt_descarga);
  limpa_l_int(&XX->l_ve_dt_disabled);
  limpa_l_int(&XX->l_ve_dt_enabled);
  limpa_l_int(&XX->l_ve_dt_erro);
  limpa_l_int(&XX->l_ve_dt_espera);
  limpa_l_int(&XX->l_ve_dt_movim_carr);
  limpa_l_int(&XX->l_ve_dt_movim_descarr);
  limpa_l_int(&XX->l_ve_n_arr_carr);
  limpa_l_int(&XX->l_ve_n_arr_descarr);
  limpa_l_int(&XX->l_ve_n_cargas);
  limpa_l_int(&XX->l_ve_n_descargas);
  limpa_l_int(&XX->l_ve_n_mov_carr);
  limpa_l_int(&XX->l_ve_n_mov_descarr);
  limpa_l_int(&XX->l_ve_taxa_util);
```

C.3.9 Tipos de tokens

C.3.9.1 AP_LAYOUT

```
ap_no_l_nos ap;
```

C.3.9.2 L_AGVS

```
ap_AGV ap;
```

C.3.9.3 L_BLOQ

```
ap_bloqueio ap;
```

C.3.9.4 APO_AGV

```
ap_AGV ap;
```

C.3.9.5 APO_TAREFA

```
ap_Tarefa ap;
```

C.3.9.6 L_TAREFAS

```
ap_Tarefa ap;
```

C.3.9.7 TAREFA

```
Tarefa tarefa;
```

C.3.9.8 L_AP_TAREFAS

```
ap_no_l_ap_Tarefas ap;
```

C.3.9.9 L_AP_AGVS

```
ap_no_l_ap_AGVs ap;
```

C.3.9.10 L_ESTADO_FILAS

```
ap_EstadoFila ap;
```

C.3.10 Variáveis locais

- layout:ap_no_l_nos – apontador para a cabeça da lista de nós do *layout*;
- p_agvs:ap_AGV – apontador para o primeiro elemento da lista ligada de AGVs;
- u_agvs:ap_AGV – apontador para o último elemento da lista ligada de AGVs;
- p_bloqueios:ap_bloqueio – apontador para o primeiro elemento da lista ligada de bloqueios de nós do *layout*;
- u_bloqueios:ap_bloqueio – apontador para o último elemento da lista ligada de bloqueios;
- p_tarefas:ap_Tarefa – apontador para o primeiro elemento da lista ligada de tarefas;
- u_tarefas:ap_Tarefa – apontador para o último elemento da lista ligada de Tarefas;

- ve_dt_fila:int – identificador da variável estatística "tempo de permanência em fila de espera de uma tarefa do tipo NORMAL";
- ve_dt_p1:int – identificador da variável estatística "tempo de execução do passo 1 de uma tarefa do tipo NORMAL";
- ve_dt_p2:int – identificador da variável estatística "tempo de execução do passo 2 de uma tarefa do tipo NORMAL";
- ve_dt_p3:int – identificador da variável estatística "tempo de execução do passo 3 de uma tarefa do tipo NORMAL";
- ve_dt_p4:int – identificador da variável estatística "tempo de execução do passo 4 de uma tarefa do tipo NORMAL";
- ve_num_af_tar:int – identificador da variável estatística "número de afectações de AGVs a uma tarefa do tipo NORMAL";
- ve_t_oper_tar:int – identificador da variável estatística "tempo de operação de uma tarefa do tipo NORMAL" (tempo durante o qual uma tarefa está afectada a um AGV);
- ve_t_exec_tar:int – identificador da variável estatística "tempo de execução de uma tarefa do tipo NORMAL"; é igual ao tempo de operação da tarefa, adicionado ao tempo de permanência da mesma na fila de espera;
- num_tar_NORMAL_lan:int – contador do número de tarefas do tipo NORMAL lançadas;
- num_tar_NORMAL_AGV_lan:int – contador do número de tarefas do tipo NORMAL_AGV lançadas;
- num_tar_POSICION_lan:int – contador do número de tarefas do tipo POSICION lançadas;
- num_tar_T_CARGA_lan:int – contador do número de tarefas do tipo T_CARGA lançadas;
- num_tar_CARGA_AGV_lan:int – contador do número de tarefas do tipo CARGA_AGV lançadas;
- num_tar_T_DESCARGA_lan:int – contador do número de tarefas do tipo T_DESCARGA lançadas;
- num_tar_POSIC_ESPEC_lan:int – contador do número de tarefas do tipo POSIC_ESPEC lançadas;
- num_tar_NORMAL_ex:int – contador do número de tarefas do tipo NORMAL executadas;
- num_tar_NORMAL_AGV_ex:int – contador do número de tarefas do tipo NORMAL_AGV executadas;
- num_tar_POSICION_ex:int – contador do número de tarefas do tipo POSICION executadas;
- num_tar_T_CARGA_ex:int – contador do número de tarefas do tipo T_CARGA executadas;

- num_tar_CARGA_AGV_ex:int – contador do número de tarefas do tipo CARGA_AGV executadas;
- num_tar_T_DESCARGA_ex:int – contador do número de tarefas do tipo T_DESCARGA executadas;
- num_tar_POSIC_ESPEC_ex:int – contador do número de tarefas do tipo POSIC_ESPEC executadas;
- num_tar_lan:int – contador do número total de tarefas lançadas;
- num_tar_rej:int – contador do número total de tarefas rejeitadas;
- num_tar_cancel:int – contador do número total de tarefas canceladas;
- num_tar_ex:int – contador do número total de tarefas executadas;
- ve_num_tar_lan:int – identificador da variável estatística "número de tarefas lançadas";
- ve_num_tar_rej:int – identificador da variável estatística "número de tarefas rejeitadas";
- ve_num_tar_cancel:int – identificador da variável estatística "número de tarefas canceladas";
- ve_num_tar_ex:int – identificador da variável estatística "número de tarefas executadas";
- l_ve_contag_tar_lan:ap_no_l_int – apontador para a cabeça da lista de identificadores de variáveis estatísticas, referentes à contagem de tarefas lançadas de cada tipo;
- l_ve_contag_tar_ex:ap_no_l_int – apontador para a cabeça da lista de identificadores de variáveis estatísticas, referentes à contagem de tarefas executadas de cada tipo;
- l_ve_dt_disabled:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "tempo de permanência no estado DISABLED" de todos os AGVs;
- l_ve_dt_enabled:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "tempo de permanência no estado ENABLED" de todos os AGVs;
- l_ve_dt_espera:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "tempo de permanência no estado ESPERA" de todos os AGVs;
- l_ve_dt_bloqueado:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "tempo de permanência no estado BLOQUEADO" de todos os AGVs;
- l_ve_dt_movim_descarr:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "tempo de permanência no estado EX_MOVIM/DESCARREGADO" de todos os AGVs;

- l_ve_dt_movim_carr:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "tempo de permanência no estado EX_MOVIM/CARREGADO" de todos os AGVs;
- l_ve_dt_carga:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "tempo de permanência no estado CARGA_AGV" de todos os AGVs;
- l_ve_dt_descarga:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "tempo de permanência no estado DESCARGA_AGV " de todos os AGVs;
- l_ve_dt_erro:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "tempo de permanência no estado ERRO" de todos os AGVs;
- l_ve_n_arr_descarr:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "número de arranques descarregado" de todos os AGVs;
- l_ve_n_arr_carr:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "número de arranques carregado" de todos os AGVs;
- l_ve_n_mov_carr:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "número de trajectos executados carregado" de todos os AGVs;
- l_ve_n_mov_descarr:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "número de trajectos executados descarregado" de todos os AGVs;
- l_ve_n_cargas:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "número de operações de carga" de todos os AGVs;
- l_ve_n_descargas:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "número de operações de descarga" de todos os AGVs;
- id_prox_mens_SGM:ulint – identificador da próxima mensagem, que irá ser recebida do SGM; a variável é incrementada sempre que for recebida uma mensagem do SGM, ou sempre que for enviada ao SGM uma mensagem do tipo T_POSIC_ESPEC;
- buffer:xx_string – *buffer* utilizado para chamar a função 'prn_string()';
- l_ve_estado_AGV:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "estado do AGV" de todos os AGVs;
- l_ve_taxa_util:ap_no_l_int – apontador para a cabeça da lista de identificadores da variável estatística "taxa de utilização do AGV" de todos os AGVs;
- p_estado_filas:ap_EstadoFila – apontador para o primeiro elemento da lista com o estado das filas de saída (comprimento das filas) de todos os pontos de carga dos AGVs;
- u_estado_filas:ap_EstadoFila – apontador para o último elemento da lista com o estado das filas de saída (comprimento das filas) de todos os pontos de carga dos AGVs;

C.3.11 Posições partilhadas

- FIM_CONFIG_SGAGV:NUL;
- LISTA_AGVS:L_AGVS;
- ESTADO_LAYOUT:L_BLOQ;
- TAREFA_AGV_DESACTIVADO:APO_TAREFA;
- LISTA_TAREFAS:L_TAREFAS;
- AGV_ELIMIN_MOV_AVAN:APO_AGV;
- MOV_AVAN_CANCEL:NUL.

C.3.12 Entidades da RP principal da classe SGAGV

C.3.12.1 Posição 'ESTADO_FILAS:L_ESTADO_FILAS'

C.3.12.1.1 Código de inicialização do token da marcação inicial

```
ESTADO_FILAS->ap=NULL;
```

C.3.12.2 Transições

C.3.12.2.1 Acção de 'ACT_ESTADO_FILA_CARGA'

```
ap_EstadoFila fila=
    procura_EstadoFila(FILA_PALETE_INSERIDA->i, XX->p_estado_filas);

(fila->Lf)++;
(fila->Rf)++;

sprintf(XX->buffer, "\nSGAGV: insercao de uma palete na fila %u (Lf=%u,
Rf=%u)\n",
    fila->id, fila->Lf, fila->Rf);
prn_string(XX->buffer);

ESTADO_FILAS->ap=XX->p_estado_filas;

xx_setstate(63);
```

C.3.12.2.2 Acção de 'INICIALIZA_ESTADO_FILA'

```
sprintf(XX->buffer, "\nSGAGV: fila de carga %u tem capacidade igual a
%u\n",
    CAPACIDADE_FILA->fila, CAPACIDADE_FILA->cap);
prn_string(XX->buffer);

insere_EstadoFila(CAPACIDADE_FILA->fila, CAPACIDADE_FILA->cap,
    &XX->p_estado_filas, &XX->u_estado_filas);

xx_setstate(62);
```

C.3.13 Entidades da RP da página 'cancela_movim_avan_AGVs'

C.3.13.1 Posição 'AGV_ESPERA_ELIMIN_MOV_AVAN:APO_AGV'

C.3.13.2 Transições

C.3.13.2.1 Acção de 'CONF_CANCEL_MOV_AVAN'

```
xx_setstate(29);
```

C.3.13.2.2 ENVIA_CANCEL_MOV_AVAN

C.3.13.2.2.1 Predicado

```
( AGV_ELIMIN_MOV_AVAN->ap->num_mov_avan > 0)
```

C.3.13.2.2.2 Acção

```
MENS_ENV_SPAGV->mens.id_agv=AGV_ELIMIN_MOV_AVAN->ap->id_agv;
MENS_ENV_SPAGV->mens.tipo_mens=CANCEL_MOV_AVAN;

sprintf(XX->buffer,
"\nSGAGV: ordena ao SPAGV o cancelamento de todos os movimentos em
avanco do AGV %u,\n atraves do envio da mensagem:\n",
AGV_ELIMIN_MOV_AVAN->ap->id_agv);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_ENV_SPAGV->mens);

xx_setstate(27);
```

C.3.13.2.3 REC_CONF_CANCEL_MOV_AVAN

C.3.13.2.3.1 Predicado

```
(MENS_RECEB_SPAGV->mens.tipo_mens==MOV_AVAN_CANCEL &&
 MENS_RECEB_SPAGV->mens.id_agv==AGV_ESPERA_ELIMIN_MOV_AVAN->ap->id_agv
)
```

C.3.13.2.3.2 Acção

```
AGV_ESPERA_ELIMIN_MOV_AVAN->ap->num_mov_avan=0;

sprintf(XX->buffer,
"\nSGAGV: notificacao do cancelamento de todos os movimentos em avanco
do AGV %u,\n atraves da recepcao da mensagem do SPAGV:\n",
MENS_RECEB_SPAGV->mens.id_agv);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_RECEB_SPAGV->mens);

xx_setstate(28);
```

C.3.14 Entidades da RP da página 'cria_lista_AGVs'

C.3.14.1 Posição "ID_AGV:ULINT'

C.3.14.2 Transições

C.3.14.2.1 GERA_AGV_DISABLED

C.3.14.2.1.1 Predicado

```
(ID_AGV->i<=XX->numero_agvs)
```

C.3.14.2.1.2 Acção

```
if (insere_AGV_DISABLED(ID_AGV->i, &XX->p_agvs, &XX->u_agvs))
{
  sprintf(XX->buffer, "SGAGV: AGV %u no estado DISABLED criado\n",
ID_AGV->i);
  prn_string(XX->buffer);
  (ID_AGV->i)++;
  LISTA_AGVS->ap=XX->p_agvs;
}
else
{
  sprintf(XX->buffer, "SGAGV: erro ao criar o AGV %u no estado
DISABLED\n", ID_AGV->i);
  prn_string(XX->buffer);
}

xx_setstate(8);
```

C.3.14.2.2 Acção de 'INIC_ID_AGV'

```
ID_AGV->i=1;

xx_setstate(7);
```

C.3.14.2.3 Acção de 'LIMPA_ID_AGV'

```
xx_setstate(9);
```

C.3.15 Entidades da RP da página 'escalona_e_despacha_tar_normais'

C.3.15.1 Posições

- AGVS_REAFECTADOS:L_AP_AGVS;
- AGVS_REAF_RESERV:L_AP_AGVS;
- TAR_MUD_ESTADO:L_AP_TAREFAS.

C.3.15.2 Transições

C.3.15.2.1 Acção de 'ACT_AGVS_REAFECTADOS'

```
remove_cab_1_ap_AGVs(&AGVS_REAF_RESERV->ap);

xx_setstate(33);
```

C.3.15.2.2 CANCEL_MOV_AVAN_AGV_REAF

C.3.15.2.2.1 Predicado

```
(AGVS_REAFECTADOS->ap!=NULL)
```

C.3.15.2.2.2 Acção

```
AGV_ELIMIN_MOV_AVAN->ap=AGVS_REAFECTADOS->ap->agv;

xx_setstate(32);
```

C.3.15.2.3 ESCAL_DESP_TAR_NORMAIS

C.3.15.2.3.1 Predicado

```
(ha_1_AGV_disponivel(LISTA_AGVS->ap) &&
 ha_1_Tar_n_escalon(LISTA_TAREFAS->ap)
)
```

C.3.15.2.3.2 Acção

```
prn_string("\nSGAGV: o Escalonador vai ser chamado.\n");

escalon_despacho(XX->regra_ADCT, XX->regra_ADV,
    XX->p_tarefas, XX->p_agvs, XX->layout,
    XX->poss_acesso, XX->num_max_af, XX->reaf_dinam,
    XX->p_estado_filas, xx_gettime(),
    &TAR_MUD_ESTADO->ap, &AGVS_REAFECTADOS->ap);

xx_setstate(31);
```

C.3.15.2.4 FINALIZA_ESCAL_DESP

C.3.15.2.4.1 Predicado

```
(AGVS_REAFECTADOS->ap==NULL &&
 TAR_MUD_ESTADO->ap==NULL
)
```

C.3.15.2.4.2 Acção

```
LISTA_AGVS->ap=XX->p_agvs;
LISTA_TAREFAS->ap=XX->p_tarefas;

xx_setstate(35);
```

C.3.15.2.5 NOTIF_SGM_MUD_EST_TAR

C.3.15.2.5.1 Predicado

```
(TAR_MUD_ESTADO->ap!=NULL)
```

C.3.15.2.5.2 Acção

```
ap_EstadoFila fila;

if (TAR_MUD_ESTADO->ap->tarefa->estado_tar==REJEITADA)
{
 MENS_ENV_SGM->mens.id_mens=TAR_MUD_ESTADO->ap->tarefa->id_tar;
 MENS_ENV_SGM->mens.tipo_mens=CONFIRMACAO;
 MENS_ENV_SGM->mens.p1=INSUCESSO;
 MENS_ENV_SGM->mens.p2=TAR_MUD_ESTADO->ap->tarefa->prior;

 remove_tarefa(TAR_MUD_ESTADO->ap->tarefa->id_tar,
   &XX->p_tarefas, &XX->u_tarefas);

 sprintf(XX->buffer,
   "\nSGAGV: notificacao do SGM da rejeicao da Tarefa %u,\n atraves do
envio da mensagem:\n",
   MENS_ENV_SGM->mens.id_mens);
 prn_string(XX->buffer);
 prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGM->mens);
}
else
{
 preenche_mens_ESTADO_TAREFA(&MENS_ENV_SGM->mens,
   TAR_MUD_ESTADO->ap->tarefa);
 sprintf(XX->buffer,
   "\nSGAGV: notificacao do SGM da mudanca de estado da Tarefa %u,\n
atraves do envio da mensagem:\n",
   MENS_ENV_SGM->mens.p1);
 prn_string(XX->buffer);
 prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGM->mens);

 if (TAR_MUD_ESTADO->ap->tarefa->estado_tar!=TAR_MUD_ESTADO->ap-
>tarefa->est_antes_desp)
 {
   fila=procura_EstadoFila(TAR_MUD_ESTADO->ap->tarefa->no1->id_no, XX-
>p_estado_filas);
   if (fila!=NULL)
   {
    if (TAR_MUD_ESTADO->ap->tarefa->estado_tar==FILA)
    {
     (fila->Rf)++;
     sprintf(XX->buffer, "\nSGAGV: tarefa com origem no ponto %u passou
ao estado FILA (Lf=%u, Rf=%u)#\n",
   fila->id, fila->Lf, fila->Rf);
    }
    else
    {
     (fila->Rf)--;
     sprintf(XX->buffer, "\nSGAGV: tarefa com origem no ponto %u passou
ao estado PASSO1 (Lf=%u, Rf=%u)#\n",
   fila->id, fila->Lf, fila->Rf);
    }
```

```
        prn_string(XX->buffer);
        act_dados_MFCFS_fila(fila, XX->p_tarefas);
       }
     }
    }

   remove_cab_l_ap_Tarefas(&TAR_MUD_ESTADO->ap);

   xx_setstate(34);
```

C.3.16 Entidades da RP da página 'gere_AGVs_MOVIMENTO'

C.3.16.1 Posições

- AGV_FIM_MOVIM:APO_AGV;
- TAREFA_MUDOU_ESTADO:APO:TAREFA.

C.3.16.2 Transições

C.3.16.2.1 AGV_MANTEM_EST_EX_MOVIM

C.3.16.2.1.1 Predicado

```
(AGV_FIM_MOVIM->ap!=NULL &&
 AGV_FIM_MOVIM->ap->estado_movim==EX_MOVIM &&
 AGV_FIM_MOVIM->ap->num_mov_avan>0
)
```

C.3.16.2.1.2 Acção

```
act_estat_temporais_agv(AGV_FIM_MOVIM->ap, xx_gettime());
xx_writesample(
    obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_taxa_util),
    AGV_FIM_MOVIM->ap->taxa_util);

sprintf(XX->buffer,
    "\nSGAGV: AGV %u terminou o movimento %u->%u\n",
    AGV_FIM_MOVIM->ap->id_agv,
    AGV_FIM_MOVIM->ap->ult_posicao->id_no,
    AGV_FIM_MOVIM->ap->posicao->id_no);
prn_string(XX->buffer);

AGV_FIM_MOVIM->ap->ult_posicao=AGV_FIM_MOVIM->ap->posicao;

remove_bloqueios_agv(AGV_FIM_MOVIM->ap,
    &XX->p_bloqueios, &XX->u_bloqueios);
ESTADO_LAYOUT->ap=XX->p_bloqueios;

if (AGV_FIM_MOVIM->ap->tarefa_agv->estado_tar==PASSO1)
{
 AGV_FIM_MOVIM->ap->tarefa_agv->ctraj1-=AGV_FIM_MOVIM->ap->tarefa_agv-
>traj1->custo;
 AGV_FIM_MOVIM->ap->posicao=AGV_FIM_MOVIM->ap->tarefa_agv->traj1->no;
 remove_cabeca_l_apno(&AGV_FIM_MOVIM->ap->tarefa_agv->traj1);
```

```
}
else
{
 AGV_FIM_MOVIM->ap->tarefa_agv->ctraj2-=AGV_FIM_MOVIM->ap->tarefa_agv-
>traj2->custo;
 AGV_FIM_MOVIM->ap->posicao=AGV_FIM_MOVIM->ap->tarefa_agv->traj2->no;
 remove_cabeca_l_apno(&AGV_FIM_MOVIM->ap->tarefa_agv->traj2);
}

(AGV_FIM_MOVIM->ap->num_mov_avan)--;

sprintf(XX->buffer, " vai executar o movimento %u->%u\n",
   AGV_FIM_MOVIM->ap->ult_posicao->id_no,
   AGV_FIM_MOVIM->ap->posicao->id_no);
prn_string(XX->buffer);

LISTA_AGVS->ap=XX->p_agvs;

xx_setstate(48);
```

C.3.16.2.2 AGV_SAI_EST_EX_MOVIM

C.3.16.2.2.1 Predicado

```
(AGV_FIM_MOVIM!=NULL &&
 AGV_FIM_MOVIM->ap->estado_movim==EX_MOVIM &&
 AGV_FIM_MOVIM->ap->num_mov_avan==0
)
```

C.3.16.2.2.2 Acção

```
double t=xx_gettime();

sprintf(XX->buffer,
   "\nSGAGV: AGV %u terminou o movimento %u->%u\n",
   AGV_FIM_MOVIM->ap->id_agv,
   AGV_FIM_MOVIM->ap->ult_posicao->id_no,
   AGV_FIM_MOVIM->ap->posicao->id_no);
prn_string(XX->buffer);

act_estat_temporais_agv(AGV_FIM_MOVIM->ap, t);
if (AGV_FIM_MOVIM->ap->estado_carga==CARREGADO)
  xx_writesample(
   obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_dt_movim_carr),
   AGV_FIM_MOVIM->ap->dt_movim_carr);
else
  xx_writesample(
   obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_dt_movim_descarr),
   AGV_FIM_MOVIM->ap->dt_movim_descarr);

TAREFA_MUDOU_ESTADO->ap=AGV_FIM_MOVIM->ap->tarefa_agv;

if (TAREFA_MUDOU_ESTADO->ap!=NULL)
 if ((TAREFA_MUDOU_ESTADO->ap->estado_tar==PASSO1 &&
      TAREFA_MUDOU_ESTADO->ap->traj1!=NULL) ||
     (TAREFA_MUDOU_ESTADO->ap->estado_tar==PASSO3 &&
      TAREFA_MUDOU_ESTADO->ap->traj2!=NULL)
    )
```

```
{
  xx_writesample(
         obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_dt_bloqueado),
         AGV_FIM_MOVIM->ap->dt_bloqueado);
  AGV_FIM_MOVIM->ap->est_mov_ant=AGV_FIM_MOVIM->ap->estado_movim;
  AGV_FIM_MOVIM->ap->estado_movim=BLOQUEADO;
  TAREFA_MUDOU_ESTADO->ap=NULL;
}
else
  if (TAREFA_MUDOU_ESTADO->ap->estado_tar==PASSO1)
  {
    (AGV_FIM_MOVIM->ap->n_mov_descarr)++;
    xx_writesample(
  obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_n_mov_descarr),
  AGV_FIM_MOVIM->ap->n_mov_descarr);

    if (TAREFA_MUDOU_ESTADO->ap->tipo_tar==POSICION ||
    TAREFA_MUDOU_ESTADO->ap->tipo_tar==POSIC_ESPEC)
    {
  proc_tarefa_concluida(TAREFA_MUDOU_ESTADO->ap, t);
  xx_writesample(
         obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_dt_espera),
         AGV_FIM_MOVIM->ap->dt_espera);
    }
    else
    {
  AGV_FIM_MOVIM->ap->est_mov_ant=AGV_FIM_MOVIM->ap->estado_movim;
  AGV_FIM_MOVIM->ap->estado_movim=BLOQUEADO;
  act_estat_temporais_tarefa(TAREFA_MUDOU_ESTADO->ap, t);
  TAREFA_MUDOU_ESTADO->ap->estado_tar=PASSO2;
    }
  }
  else
  {
    (AGV_FIM_MOVIM->ap->n_mov_carr)++;
    xx_writesample(
  obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_n_mov_carr),
  AGV_FIM_MOVIM->ap->n_mov_carr);

    AGV_FIM_MOVIM->ap->est_mov_ant=AGV_FIM_MOVIM->ap->estado_movim;
    AGV_FIM_MOVIM->ap->estado_movim=BLOQUEADO;
    act_estat_temporais_tarefa(TAREFA_MUDOU_ESTADO->ap, t);
    TAREFA_MUDOU_ESTADO->ap->estado_tar=PASSO4;
  }
else
{
  if (AGV_FIM_MOVIM->ap->estado_carga==CARREGADO)
  {
   (AGV_FIM_MOVIM->ap->n_mov_carr)++;
   xx_writesample(
   obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_n_mov_carr),
   AGV_FIM_MOVIM->ap->n_mov_carr);
  }
  else
  {
   (AGV_FIM_MOVIM->ap->n_mov_descarr)++;
   xx_writesample(
   obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_n_mov_descarr),
   AGV_FIM_MOVIM->ap->n_mov_descarr);
```

```
    }
    xx_writesample(
      obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_dt_espera),
      AGV_FIM_MOVIM->ap->dt_espera);

    AGV_FIM_MOVIM->ap->est_mov_ant=AGV_FIM_MOVIM->ap->estado_movim;
    AGV_FIM_MOVIM->ap->estado_movim=PARADO;
  }
  xx_writesample(
      obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_taxa_util),
      AGV_FIM_MOVIM->ap->taxa_util);
  xx_writesample(
      obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_estado_AGV),
      AGV_FIM_MOVIM->ap->estado_movim);

  AGV_FIM_MOVIM->ap->ult_posicao=AGV_FIM_MOVIM->ap->posicao;

  remove_bloqueios_agv(AGV_FIM_MOVIM->ap,
      &XX->p_bloqueios, &XX->u_bloqueios);
  ESTADO_LAYOUT->ap=XX->p_bloqueios;

  LISTA_AGVS->ap=XX->p_agvs;

  preenche_mens_ESTADO_AGV(&MENS_ENV_SGM->mens,
      AGV_FIM_MOVIM->ap);
  sprintf(XX->buffer,
"\nSGAGV: notificacao do SGM sobre a alteracao do estado do AGV %u,\n
atraves do envio da mensagem:\n",
AGV_FIM_MOVIM->ap->id_agv);
  prn_string(XX->buffer);
  prn_m_SGM_SGAGV(t, MENS_ENV_SGM->mens);

  xx_setstate(49);
```

C.3.16.2.3 AGV_TERMINA_OP_CD

C.3.16.2.3.1 Predicado

```
(AGV_FIM_MOVIM->ap!=NULL &&
 (AGV_FIM_MOVIM->ap->estado_movim==CARGA_AGV ||
  AGV_FIM_MOVIM->ap->estado_movim==DESCARGA_AGV)
)
```

C.3.16.2.3.2 Acção

```
double t=xx_gettime();
ap_EstadoFila fila;

act_estat_temporais_agv(AGV_FIM_MOVIM->ap, t);

TAREFA_MUDOU_ESTADO->ap=AGV_FIM_MOVIM->ap->tarefa_agv;

if (AGV_FIM_MOVIM->ap->estado_movim==CARGA_AGV)
{
  fila=procura_EstadoFila(AGV_FIM_MOVIM->ap->posicao->id_no,
   XX->p_estado_filas);
  if (fila!=NULL)
  {
```

```
      (fila->Lf)--;
      sprintf(XX->buffer, "\nSGAGV: saiu uma palete da fila %u (Lf=%u,
Rf=%u)#\n",
     fila->id, fila->Lf, fila->Rf);
      prn_string(XX->buffer);
    }

    sprintf(XX->buffer,
     "\nSGAGV: AGV %u terminou a operacao de carga no ponto %u\n",
     AGV_FIM_MOVIM->ap->id_agv,
     AGV_FIM_MOVIM->ap->posicao->id_no);

    AGV_FIM_MOVIM->ap->estado_carga=CARREGADO;
    (AGV_FIM_MOVIM->ap->n_cargas)++;
    xx_writesample(
     obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_n_cargas),
     AGV_FIM_MOVIM->ap->n_cargas);
    xx_writesample(
     obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_dt_carga),
     AGV_FIM_MOVIM->ap->dt_carga);

    if (AGV_FIM_MOVIM->ap->tarefa_agv!=NULL)
      if (AGV_FIM_MOVIM->ap->tarefa_agv->tipo_tar==T_CARGA ||
        AGV_FIM_MOVIM->ap->tarefa_agv->tipo_tar==CARGA_AGV)
      {
      proc_tarefa_concluida(TAREFA_MUDOU_ESTADO->ap, t);
      xx_writesample(
            obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_dt_espera),
            AGV_FIM_MOVIM->ap->dt_espera);
      }
      else
      {
        act_estat_temporais_tarefa(TAREFA_MUDOU_ESTADO->ap, t);
        xx_writesample(
            obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_dt_bloqueado),
            AGV_FIM_MOVIM->ap->dt_bloqueado);
        TAREFA_MUDOU_ESTADO->ap->estado_tar=PASSO3;
        AGV_FIM_MOVIM->ap->est_mov_ant=AGV_FIM_MOVIM->ap->estado_movim;
        AGV_FIM_MOVIM->ap->estado_movim=BLOQUEADO;
      }
    else
    {
      xx_writesample(
            obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_dt_espera),
            AGV_FIM_MOVIM->ap->dt_espera);
     AGV_FIM_MOVIM->ap->est_mov_ant=AGV_FIM_MOVIM->ap->estado_movim;
     AGV_FIM_MOVIM->ap->estado_movim=PARADO;
    }
  }
  else
  {
    sprintf(XX->buffer,
     "\nSGAGV: AGV %u terminou a operacao de descarga no ponto %u\n",
     AGV_FIM_MOVIM->ap->id_agv,
     AGV_FIM_MOVIM->ap->posicao->id_no);

    AGV_FIM_MOVIM->ap->estado_carga=DESCARREGADO;
    (AGV_FIM_MOVIM->ap->n_descargas)++;
    xx_writesample(
```

```
      obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_n_descargas),
      AGV_FIM_MOVIM->ap->n_descargas);
    xx_writesample(
      obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_dt_descarga),
      AGV_FIM_MOVIM->ap->dt_descarga);
    xx_writesample(
      obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_dt_espera),
      AGV_FIM_MOVIM->ap->dt_espera);

    if (AGV_FIM_MOVIM->ap->tarefa_agv!=NULL)
      proc_tarefa_concluida(TAREFA_MUDOU_ESTADO->ap, t);
    else
    {
      AGV_FIM_MOVIM->ap->est_mov_ant=AGV_FIM_MOVIM->ap->estado_movim;
      AGV_FIM_MOVIM->ap->estado_movim=PARADO;
    }
  }

  prn_string(XX->buffer);

  xx_writesample(
      obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_taxa_util),
      AGV_FIM_MOVIM->ap->taxa_util);
  xx_writesample(
      obtem_int(AGV_FIM_MOVIM->ap->id_agv, XX->l_ve_estado_AGV),
      AGV_FIM_MOVIM->ap->estado_movim);

  LISTA_AGVS->ap=XX->p_agvs;

  preenche_mens_ESTADO_AGV(&MENS_ENV_SGM->mens,
      AGV_FIM_MOVIM->ap);
  sprintf(XX->buffer,
      "\nSGAGV: notificacao do SGM sobre a alteracao do estado do AGV
%u,\n atraves do envio da mensagem:\n",
      AGV_FIM_MOVIM->ap->id_agv);
  prn_string(XX->buffer);
  prn_m_SGM_SGAGV(t, MENS_ENV_SGM->mens);

  xx_setstate(50);
```

C.3.16.2.4 ENVIA_MOV_AVANCO

C.3.16.2.4.1 Predicado

```
(agv_enviar_mov_avan(LISTA_AGVS->ap, ESTADO_LAYOUT->ap,
    XX->num_max_mov_avan)!=NULL)
```

C.3.16.2.4.2 Acção

```
ap_AGV agv=agv_enviar_mov_avan(LISTA_AGVS->ap,
    ESTADO_LAYOUT->ap, XX->num_max_mov_avan);
ap_no_l_nos no_or_pr_mov;
ap_no_l_apno no_de_pr_mov=obtem_no_traj_mov_avan(agv);

if (no_de_pr_mov->anterior!=NULL)
    no_or_pr_mov=no_de_pr_mov->anterior->no;
  else no_or_pr_mov=agv->posicao;
```

```
(agv->num_mov_avan)++;

insere_bloqs_movim(agv, no_or_pr_mov,
   no_de_pr_mov->no->id_no, XX->layout,
   &XX->p_bloqueios, &XX->u_bloqueios);
ESTADO_LAYOUT->ap=XX->p_bloqueios;

MENS_ENV_SPAGV->mens.id_agv=agv->id_agv;
MENS_ENV_SPAGV->mens.tipo_mens=MOVIMENTO;
MENS_ENV_SPAGV->mens.p1=MOV_TRANSL;
MENS_ENV_SPAGV->mens.p2=no_de_pr_mov->no->id_no;
sprintf(XX->buffer,
"\nSGAGV: envia ao SPAGV um movim. em avanco para o AGV %u,\n atraves do
envio da mensagem:\n",
agv->id_agv);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_ENV_SPAGV->mens);

xx_setstate(46);
```

C.3.16.2.5 Acção de 'LIMPA_TAREFA_MUDOU_ESTADO'

```
LISTA_TAREFAS->ap=XX->p_tarefas;

xx_setstate(65);
```

C.3.16.2.6 NOTIF_SGM_MUD_EST_TAREFA

C.3.16.2.6.1 Predicado

```
(TAREFA_MUDOU_ESTADO->ap!=NULL)
```

C.3.16.2.6.2 Acção

```
double t_oper;
int ve;

if (TAREFA_MUDOU_ESTADO->ap->estado_tar!=CONCLUIDA)
{
  preenche_mens_ESTADO_TAREFA(&MENS_ENV_SGM->mens,
   TAREFA_MUDOU_ESTADO->ap);
  sprintf(XX->buffer,
   "\nSGAGV: notificacao do SGM sobre a mudanca do estado da Tarefa
%u,\n atraves do envio da mensagem:\n",
   MENS_ENV_SGM->mens.p1);
}
else
{
  MENS_ENV_SGM->mens.id_mens=TAREFA_MUDOU_ESTADO->ap->id_tar;
  MENS_ENV_SGM->mens.tipo_mens=CONFIRMACAO;
  MENS_ENV_SGM->mens.p1=SUCESSO;

  if (TAREFA_MUDOU_ESTADO->ap->tipo_tar==NORMAL)
  {
    xx_writesample(XX->ve_dt_fila, TAREFA_MUDOU_ESTADO->ap->dt_fila);
    xx_writesample(XX->ve_dt_p1, TAREFA_MUDOU_ESTADO->ap->dt_p1);
    xx_writesample(XX->ve_dt_p2, TAREFA_MUDOU_ESTADO->ap->dt_p2);
    xx_writesample(XX->ve_dt_p3, TAREFA_MUDOU_ESTADO->ap->dt_p3);
```

```
      xx_writesample(XX->ve_dt_p4, TAREFA_MUDOU_ESTADO->ap->dt_p4);
      xx_writesample(XX->ve_num_af_tar, (double) TAREFA_MUDOU_ESTADO->ap-
>num_af);
      t_oper=TAREFA_MUDOU_ESTADO->ap->dt_p1+
     TAREFA_MUDOU_ESTADO->ap->dt_p2+
     TAREFA_MUDOU_ESTADO->ap->dt_p3+
     TAREFA_MUDOU_ESTADO->ap->dt_p4;
      xx_writesample(XX->ve_t_oper_tar, t_oper);
      xx_writesample(XX->ve_t_exec_tar,
     TAREFA_MUDOU_ESTADO->ap->dt_fila+t_oper);
    }

    ve=obtem_int(TAREFA_MUDOU_ESTADO->ap->tipo_tar+1,
     XX->l_ve_contag_tar_ex);
    switch (TAREFA_MUDOU_ESTADO->ap->tipo_tar)
    {
     case NORMAL:
            (XX->num_tar_NORMAL_ex)++;
            xx_writesample(ve,
                   (double) XX->num_tar_NORMAL_ex);
            break;
     case NORMAL_AGV:
            (XX->num_tar_NORMAL_AGV_ex)++;
            xx_writesample(ve,
                   (double) XX->num_tar_NORMAL_AGV_ex);
            break;
     case POSICION:
            (XX->num_tar_POSICION_ex)++;
            xx_writesample(ve,
                   (double) XX->num_tar_POSICION_ex);
            break;
     case T_CARGA:
            (XX->num_tar_T_CARGA_ex)++;
            xx_writesample(ve,
                   (double) XX->num_tar_T_CARGA_ex);
            break;
     case CARGA_AGV:
            (XX->num_tar_CARGA_AGV_ex)++;
            xx_writesample(ve,
                   (double) XX->num_tar_CARGA_AGV_ex);
            break;
     case T_DESCARGA:
            (XX->num_tar_T_DESCARGA_ex)++;
            xx_writesample(ve,
                   (double) XX->num_tar_T_DESCARGA_ex);
            break;
     case POSIC_ESPEC:
            (XX->num_tar_POSIC_ESPEC_ex)++;
            xx_writesample(ve,
                   (double) XX->num_tar_POSIC_ESPEC_ex);
    }
    (XX->num_tar_ex)++;
    xx_writesample(XX->ve_num_tar_ex, (double) XX->num_tar_ex);

    remove_tarefa(TAREFA_MUDOU_ESTADO->ap->id_tar,
     &XX->p_tarefas, &XX->u_tarefas);
    sprintf(XX->buffer,
     "\nSGAGV: notificacao do SGM sobre a conclusao com sucesso da Tarefa
%u,\n atraves do envio da mensagem:\n",
```

```
    MENS_ENV_SGM->mens.id_mens);
}

LISTA_TAREFAS->ap=XX->p_tarefas;

prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGM->mens);

xx_setstate(51);
```

C.3.16.2.7 REC_FIM_MOVIM

C.3.16.2.7.1 Predicado

```
(MENS_RECEB_SPAGV->mens.tipo_mens==FIM_MOVIMENTO)
```

C.3.16.2.7.2 Acção

```
AGV_FIM_MOVIM->ap=obtem_ap_AGV(MENS_RECEB_SPAGV->mens.id_agv,
    XX->p_agvs);

sprintf(XX->buffer,
"\nSGAGV: notificacao pelo SPAGV do fim de um movimento do AGV %u,\n
atraves da mensagem:\n",
MENS_RECEB_SPAGV->mens.id_agv);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_RECEB_SPAGV->mens);

xx_setstate(47);
```

C.3.17 Entidades da RP da página 'gere_AGVs_est_BLOQUEADO'

C.3.17.1 Posição 'AGV_FIM_POSICION:APO_AGV'

C.3.17.2 Transições

C.3.17.2.1 ACT_EST_TAR_TRAJ_NULO

C.3.17.2.1.1 Predicado

```
(tarefa_traj_nulo(LISTA_AGVS->ap)!=NULL)
```

C.3.17.2.1.2 Acção

```
char s[15];
ap_Tarefa tar=tarefa_traj_nulo(LISTA_AGVS->ap);
double t_oper;
int ve;
double t=xx_gettime();

if (tar->tipo_tar!=POSICION &&
    tar->tipo_tar!=POSIC_ESPEC
   )
{
```

```
    if (tar->estado_tar==PASSO1)
      tar->estado_tar=PASSO2;
     else tar->estado_tar=PASSO4;
    preenche_mens_ESTADO_TAREFA(&MENS_ENV_SGM->mens, tar);
    escr_str_est_tarefa(s, tar->estado_tar);
    sprintf(XX->buffer,
      "\nSGAGV: notifica o SGM sobre a passagem da Tarefa %u ao estado
 %s,\n atraves do envio da mensagem:\n",
      tar->id_tar, s);

    AGV_FIM_POSICION->ap=NULL;
   }
   else
   {
    AGV_FIM_POSICION->ap=tar->agv_tarefa;

    proc_tarefa_concluida(tar, t);

    xx_writesample(
      obtem_int(tar->agv_tarefa->id_agv, XX->l_ve_dt_espera),
      tar->agv_tarefa->dt_espera);
    xx_writesample(
      obtem_int(tar->agv_tarefa->id_agv, XX->l_ve_estado_AGV),
      PARADO);

    MENS_ENV_SGM->mens.id_mens=tar->id_tar;
    MENS_ENV_SGM->mens.tipo_mens=CONFIRMACAO;
    MENS_ENV_SGM->mens.p1=SUCESSO;

    xx_writesample(XX->ve_dt_fila, tar->dt_fila);
    xx_writesample(XX->ve_dt_p1, tar->dt_p1);
    xx_writesample(XX->ve_dt_p2, tar->dt_p2);
    xx_writesample(XX->ve_dt_p3, tar->dt_p3);
    xx_writesample(XX->ve_dt_p4, tar->dt_p4);
    xx_writesample(XX->ve_num_af_tar, (double) tar->num_af);
    t_oper=tar->dt_p1+tar->dt_p2+tar->dt_p3+tar->dt_p4;
    xx_writesample(XX->ve_t_oper_tar, t_oper);
    xx_writesample(XX->ve_t_exec_tar, tar->dt_fila+t_oper);

    ve=obtem_int(tar->tipo_tar+1, XX->l_ve_contag_tar_ex);
    if (tar->tipo_tar==POSICION)
    {
     (XX->num_tar_POSICION_ex)++;
     xx_writesample(ve, (double) XX->num_tar_POSICION_ex);
    }
    else
    {
     (XX->num_tar_POSIC_ESPEC_ex)++;
     xx_writesample(ve, (double) XX->num_tar_POSIC_ESPEC_ex);
    }
    (XX->num_tar_ex)++;
    xx_writesample(XX->ve_num_tar_ex, (double) XX->num_tar_ex);

    remove_tarefa(tar->id_tar, &XX->p_tarefas, &XX->u_tarefas);

    sprintf(XX->buffer,
      "\nSGAGV: notificacao do SGM sobre a conclusao com sucesso da Tarefa
 %u,\n atraves do envio da mensagem:\n",
      MENS_ENV_SGM->mens.id_mens);
```

```
 LISTA_TAREFAS->ap=XX->p_tarefas;
}

prn_string(XX->buffer);
prn_m_SGM_SGAGV(t, MENS_ENV_SGM->mens);

xx_setstate(53);
```

C.3.17.2.2 AGV_BLOQUEADO_CAR_OU_DES

C.3.17.2.2.1 Predicado

```
(agv_BLOQUEADO_CD(LISTA_AGVS->ap)!=NULL)
```

C.3.17.2.2.2 Acção

```
ap_AGV agv=agv_BLOQUEADO_CD(LISTA_AGVS->ap);
char s1[9];
char s2[13];
double t=xx_gettime();

if (agv->t_mud_estado<t)
{
 act_estat_temporais_agv(agv, t);
 xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_bloqueado),
    agv->dt_bloqueado);
 xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_taxa_util),
    agv->taxa_util);
}

if (agv->tarefa_agv->estado_tar==PASSO2)
{
 xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_carga),
    agv->dt_carga);
 agv->est_mov_ant=agv->estado_movim;
 agv->estado_movim=CARGA_AGV;
 MENS_ENV_SPAGV->mens.p1=MOV_CARGA;
 sprintf(s1, "carga");
}
else
{
 xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_descarga),
    agv->dt_descarga);
 agv->est_mov_ant=agv->estado_movim;
 agv->estado_movim=DESCARGA_AGV;
 MENS_ENV_SPAGV->mens.p1=MOV_DESCARGA;
 sprintf(s1, "descarga");
}
xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_estado_AGV),
    agv->estado_movim);

preenche_mens_ESTADO_AGV(&MENS_ENV_SGM->mens, agv);
```

```
escr_str_est_movim_agv(s2, agv->estado_movim);
sprintf(XX->buffer,
"\nSGAGV: notificacao do SGM sobre a passagem do AGV %u BLOQUEADO\n ao
estado %s, atraves do envio da mensagem:\n",
agv->id_agv, s2);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(t, MENS_ENV_SGM->mens);

MENS_ENV_SPAGV->mens.id_agv=agv->id_agv;
MENS_ENV_SPAGV->mens.tipo_mens=MOVIMENTO;
MENS_ENV_SPAGV->mens.p2=(ulint)
   agv->posicao->pos_rel_mesa*agv->orient+1;
/* da 0 se o movimento for para a DIREITA ou da 2 se o
   movimento for para a ESQUERDA */

sprintf(XX->buffer,
"\nSGAGV: ordena ao SPAGV o inicio de um movim. de %s do AGV %u,\n
atraves do envio da mensagem:\n",
s1, agv->id_agv);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(t, MENS_ENV_SPAGV->mens);

xx_setstate(45);
```

C.3.17.2.3 AGV_BLOQUEADO_MOVIMENTO

C.3.17.2.3.1 Predicado

```
(agv_BLOQUEADO_MOVIMENTO(LISTA_AGVS->ap, ESTADO_LAYOUT->ap)!=NULL)
```

C.3.17.2.3.2 Acção

```
ap_AGV agv=agv_BLOQUEADO_MOVIMENTO(LISTA_AGVS->ap,
   ESTADO_LAYOUT->ap);
double t=xx_gettime();

if (agv->tarefa_agv->estado_tar==PASSO1)
{
 agv->posicao=agv->tarefa_agv->traj1->no;
 agv->tarefa_agv->ctraj1-=agv->tarefa_agv->traj1->custo;
 remove_cabeca_l_apno(&agv->tarefa_agv->traj1);
}
else
{
 agv->posicao=agv->tarefa_agv->traj2->no;
 agv->tarefa_agv->ctraj2-=agv->tarefa_agv->traj2->custo;
 remove_cabeca_l_apno(&agv->tarefa_agv->traj2);
}

if (agv->t_mud_estado<t || agv->est_mov_ant!=EX_MOVIM)
{
 if (agv->estado_carga==DESCARREGADO)
 {
  (agv->n_arr_descarr)++;
  xx_writesample(
   obtem_int(agv->id_agv, XX->l_ve_n_arr_descarr),
   (double) agv->n_arr_descarr);
 }
```

```
    else
    {
     (agv->n_arr_carr)++;
     xx_writesample(
       obtem_int(agv->id_agv, XX->l_ve_n_arr_carr),
       (double) agv->n_arr_carr);
    }
   }

   if (agv->t_mud_estado<t)
   {
    act_estat_temporais_agv(agv, t);
    xx_writesample(
       obtem_int(agv->id_agv, XX->l_ve_dt_bloqueado),
       agv->dt_bloqueado);
    xx_writesample(
       obtem_int(agv->id_agv, XX->l_ve_taxa_util),
       agv->taxa_util);
   }
   if (agv->estado_carga==DESCARREGADO)
     xx_writesample(
       obtem_int(agv->id_agv, XX->l_ve_dt_movim_descarr),
       agv->dt_movim_descarr);
   else
     xx_writesample(
       obtem_int(agv->id_agv, XX->l_ve_dt_movim_carr),
       agv->dt_movim_carr);
   xx_writesample(
       obtem_int(agv->id_agv, XX->l_ve_estado_AGV),
       EX_MOVIM);

   agv->est_mov_ant=agv->estado_movim;
   agv->estado_movim=EX_MOVIM;

   insere_bloqs_movim(agv, agv->ult_posicao,
       agv->posicao->id_no, XX->layout,
       &XX->p_bloqueios, &XX->u_bloqueios);
   ESTADO_LAYOUT->ap=XX->p_bloqueios;

   preenche_mens_ESTADO_AGV(&MENS_ENV_SGM->mens, agv);
   sprintf(XX->buffer,
"\nSGAGV: notificacao do SGM sobre a passagem do AGV %u BLOQUEADO\n ao
estado EX_MOVIM, atraves do envio da mensagem:\n",
agv->id_agv);
   prn_string(XX->buffer);
   prn_m_SGM_SGAGV(t, MENS_ENV_SGM->mens);

   MENS_ENV_SPAGV->mens.id_agv=agv->id_agv;
   MENS_ENV_SPAGV->mens.tipo_mens=MOVIMENTO;
   MENS_ENV_SPAGV->mens.p1=MOV_TRANSL;
   MENS_ENV_SPAGV->mens.p2=agv->posicao->id_no;
   sprintf(XX->buffer,
"\nSGAGV: ordena ao SPAGV o inicio de um movim. de transl. do AGV %u,\n
atraves do envio da mensagem:\n",
agv->id_agv);
   prn_string(XX->buffer);
   prn_m_SGAGV_SPAGV(t, MENS_ENV_SPAGV->mens);

   xx_setstate(44);
```

C.3.17.2.5 AGV_BLOQUEADO_PARADO

C.3.17.2.5.1 Predicado

```
(agv_BLOQUEADO_PARADO(LISTA_AGVS->ap)!=NULL)
```

C.3.17.2.5.2 Acção

```
ap_AGV agv=agv_BLOQUEADO_PARADO(LISTA_AGVS->ap);
double t=xx_gettime();

if (agv->t_mud_estado<t)
{
 act_estat_temporais_agv(agv, t);
 xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_bloqueado),
    agv->dt_bloqueado);
 xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_taxa_util),
    agv->taxa_util);
}
xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_espera),
    agv->dt_espera);
xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_estado_AGV),
    PARADO);

agv->est_mov_ant=agv->estado_movim;
agv->estado_movim=PARADO;

preenche_mens_ESTADO_AGV(&MENS_ENV_SGM->mens, agv);
sprintf(XX->buffer,
"\nSGAGV: notificacao do SGM sobre a passagem do AGV %u BLOQUEADO\n ao
estado PARADO, atraves do envio da mensagem:\n",
agv->id_agv);
 prn_string(XX->buffer);
 prn_m_SGM_SGAGV(t, MENS_ENV_SGM->mens);

 xx_setstate(52);
```

C.3.17.2.6 Acção de 'AGV_FIM_POSIC_PARADO'

```
 preenche_mens_ESTADO_AGV(&MENS_ENV_SGM->mens,
    AGV_FIM_POSICION->ap);
 sprintf(XX->buffer,
"\nSGAGV: notifica o SGM sobre a passagem do AGV %u BLOQUEADO ao estado
PARADO,\n atraves do envio da mensagem:\n",
AGV_FIM_POSICION->ap->id_agv);
 prn_string(XX->buffer);
 prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGM->mens);

 LISTA_AGVS->ap=XX->p_agvs;

 xx_setstate(55);
```

C.3.17.2.7 LIMPA_AGV_FIM_POSICION

C.3.17.2.7.1 Predicado

```
(AGV_FIM_POSICION->ap==NULL)
```

C.3.17.2.7.2 Acção

```
LISTA_AGVS->ap=XX->p_agvs;

xx_setstate(54);
```

C.3.18 Entidades da RP da página 'introducao_AGVs'

C.3.18.1 Posições

- AGV_MENS_INTROD:APO_AGV;
- MENS_ESTADO_INTR:MENS_SGAGV_SPAGV.

C.3.18.2 Transições

C.3.18.2.1 CONF_INTROD_AGV

C.3.18.2.1.1 Predicado

```
(MENS_ESTADO_INTR->mens.pl==AGV_INTROD)
```

C.3.18.2.1.2 Acção

```
double t=xx_gettime();

 sprintf(XX->buffer,
"\nSGAGV: notificacao do SPAGV sobre o sucesso da introducao do AGV
%u,\n atraves do envio da mensagem:\n",
MENS_ESTADO_INTR->mens.id_agv);
 prn_string(XX->buffer);
 prn_m_SGAGV_SPAGV(t, MENS_ENV_SPAGV->mens);

 preenche_mens_ESTADO_AGV(&MENS_ENV_SGM->mens,
    AGV_MENS_INTROD->ap);
 sprintf(XX->buffer,
"\nSGAGV: notificacao do SGM sobre a introducao do AGV %u,\n atraves do
envio da mensagem:\n",
AGV_MENS_INTROD->ap->id_agv);
 prn_string(XX->buffer);
 prn_m_SGM_SGAGV(t, MENS_ENV_SGM->mens);

 LISTA_AGVS->ap=XX->p_agvs;

 xx_setstate(37);
```

BIBLIOTECA F.E. PORTO

C.3.18.2.2 Acção de 'NOTIF_ERRO_INTROD'

```
sprintf(XX->buffer,
```

```
"\nSGAGV: notificacao do SPAGV sobre o insucesso da introducao do AGV
%u,\n atraves do envio da mensagem:\n",
MENS_ESTADO_INTR->mens.id_agv);
 prn_string(XX->buffer);
 prn_m_SGAGV_SPAGV(xx_gettime(), MENS_ENV_SPAGV->mens);

 LISTA_AGVS->ap=XX->p_agvs;

 xx_setstate(38);
```

C.3.18.2.3 REC_INTRODUCAO

C.3.18.2.3.1 Predicado

```
(MENS_RECEB_SPAGV->mens.tipo_mens==INTRODUCAO)
```

C.3.18.2.3.2 Acção

```
ap_no_l_nos no;
double t=xx_gettime();

prn_string("\nSGAGV: recepcao da mensagem do SPAGV:\n");
prn_m_SGAGV_SPAGV(t, MENS_RECEB_SPAGV->mens);

MENS_ESTADO_INTR->mens.id_agv=MENS_RECEB_SPAGV->mens.id_agv;
MENS_ESTADO_INTR->mens.tipo_mens=ESTADO_INTR;

AGV_MENS_INTROD->ap=obtem_ap_AGV(MENS_RECEB_SPAGV->mens.id_agv,
    XX->p_agvs);

if (AGV_MENS_INTROD->ap!=NULL &&
     AGV_MENS_INTROD->ap->estado_movim==ENABLED)
{
 no=get_no(XX->layout, MENS_RECEB_SPAGV->mens.p1);
 if (no!=NULL)
  if (bloq_no_indisponivel(no, AGV_MENS_INTROD->ap,
   XX->p_bloqueios)==NULL)
   {
    act_estat_temporais_agv(AGV_MENS_INTROD->ap, t);
    xx_writesample(
    obtem_int(AGV_MENS_INTROD->ap->id_agv, XX->l_ve_dt_enabled),
    AGV_MENS_INTROD->ap->dt_enabled);
    xx_writesample(
    obtem_int(AGV_MENS_INTROD->ap->id_agv, XX->l_ve_dt_espera),
    AGV_MENS_INTROD->ap->dt_espera);
    xx_writesample(
    obtem_int(AGV_MENS_INTROD->ap->id_agv, XX->l_ve_taxa_util),
    AGV_MENS_INTROD->ap->taxa_util);
    xx_writesample(
    obtem_int(AGV_MENS_INTROD->ap->id_agv, XX->l_ve_estado_AGV),
    PARADO);

    insere_bloqueios(no, AGV_MENS_INTROD->ap,
    &XX->p_bloqueios, &XX->u_bloqueios, XX->layout);

    AGV_MENS_INTROD->ap->posicao=no;
    AGV_MENS_INTROD->ap->ult_posicao=no;
    AGV_MENS_INTROD->ap->orient=MENS_RECEB_SPAGV->mens.p2;
```

```
    AGV_MENS_INTROD->ap->est_mov_ant=
    AGV_MENS_INTROD->ap->estado_movim;
    AGV_MENS_INTROD->ap->estado_movim=PARADO;
    AGV_MENS_INTROD->ap->estado_carga=DESCARREGADO;

    MENS_ESTADO_INTR->mens.p1=AGV_INTROD;

    ESTADO_LAYOUT->ap=XX->p_bloqueios;
   }
   else MENS_ESTADO_INTR->mens.p1=NO_OCUPADO;
 else MENS_ESTADO_INTR->mens.p1=NO_INEX;
}
else MENS_ESTADO_INTR->mens.p1=AGV_NOT_ENABLED;

xx_setstate(36);
```

C.3.19 Entidades da RP da página 'leitura_fich_config'

C.3.19.1 Posições

- FIM_LEITURA_FICH_LAYOUT:NUL;
- INICIO_CONFIG_SGAGV:NUL;
- LAYOUT:AP_LAYOUT;
- LAYOUT_BLOQUEIOS_COMPLETOS:NUL
- RESULT_VERIF_VAL_RAMOS_E_OC_NOS:BOOLEANO;

C.3.19.2 Transições

C.3.19.2.1 COMPLETA_BLOQUEIOS

C.3.19.2.1.1 Predicado

```
(RESULT_VERIF_VAL_RAMOS_E_OC_NOS->b!=FALSO)
```

C.3.19.2.1.2 Acção

```
completa_bloqueios(XX->layout);

xx_setstate(4);
```

C.3.19.2.2 Acção de 'LE_FICH_LAYOUT'

```
prn_string("\nSGAGV");
XX->layout=le_layout(XX->nome_fich_layout, XX->vel_frente,
    XX->vel_curva, XX->vel_tras);
LAYOUT->ap=XX->layout;

xx_setstate(1);
```

C.3.19.2.3 Acção de 'LE_POS_REL_MESAS'

```
le_f_pos_rel_pcd(XX->nome_fich_mesas, XX->layout);
```

```
LAYOUT_POS_MESAS_CONH->b=pos_mesas_conhecidas(XX->layout);

xx_setstate(5);
```

C.3.19.2.4 TERMINA_CONFIG_SGAGV

C.3.19.2.4.1 Predicado

```
(LAYOUT_POS_MESAS_CONH->b!=FALSO)
```

C.3.19.2.4.2 Acção

```
xx_setstate(6);
```

C.3.19.2.5 VERIF_UNICID_ID_NOS

C.3.19.2.5.1 Predicado

```
(XX->layout!=NULL)
```

C.3.19.2.5.2 Acção

```
RESULT_VERIF_UNICID_ID_NOS->b=unicidade_id_nos(XX->layout);

xx_setstate(2);
```

C.3.19.2.6 VERIF_VAL_RAMOS_E_OC_NOS

C.3.19.2.6.1 Predicado

```
(RESULT_VERIF_UNICID_ID_NOS->b!=FALSO)
```

C.3.19.2.6.2 Acção

```
RESULT_VERIF_VAL_RAMOS_E_OC_NOS->b=ocup_nos_e_ramos_validos(XX->layout);

xx_setstate(3);
```

C.3.20 Entidades da RP da página 'passagem_modo_manual_AGVs'

C.3.20.1 Posição 'MENS_ESTADO_AGV:MENS_SGM_SGAGV'

C.3.20.2 Transições

C.3.20.2.1 Acção de 'NOTIF_AGV_ENABLED'

```
LISTA_AGVS->ap=XX->p_agvs;

sprintf(XX->buffer,
"\nSGAGV: notifica o SGM da passagem do AGV %u ao estado ENABLED,\n
atraves do envio da mensagem:\n",
MENS_ENV_SGM->mens.p1);
prn_string(XX->buffer);
```

```
prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGM->mens);

xx_setstate(43);
```

C.3.20.2.2 REC_AGV_MANUAL

C.3.20.2.2.1 Predicado

```
(MENS_RECEB_SPAGV->mens.tipo_mens==AGV_MANUAL)
```

C.3.20.2.2.2 Acção

```
ap_AGV agv=obtem_ap_AGV(MENS_RECEB_SPAGV->mens.id_agv,
   XX->p_agvs);
double t=xx_gettime();

sprintf(XX->buffer,
"\nSGAGV: notificacao do SPAGV da passagem do AGV %u ao modo Manual,\n
atraves da recepcao da mensagem:\n",
MENS_RECEB_SPAGV->mens.id_agv);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(t, MENS_RECEB_SPAGV->mens);

act_estat_temporais_agv(agv, t);
switch (agv->estado_movim)
{
 case PARADO:
   xx_writesample(
          obtem_int(agv->id_agv, XX->l_ve_dt_espera),
          agv->dt_espera);
   break;
 case BLOQUEADO:
   xx_writesample(
          obtem_int(agv->id_agv, XX->l_ve_dt_bloqueado),
          agv->dt_bloqueado);
   break;
 case EX_MOVIM:
   if (agv->estado_carga==DESCARREGADO)
     xx_writesample(
          obtem_int(agv->id_agv, XX->l_ve_dt_movim_descarr),
          agv->dt_movim_descarr);
   else
     xx_writesample(
          obtem_int(agv->id_agv, XX->l_ve_dt_movim_carr),
          agv->dt_movim_carr);
   break;
 case CARGA_AGV:
   xx_writesample(
          obtem_int(agv->id_agv, XX->l_ve_dt_carga),
          agv->dt_carga);
   break;
 case DESCARGA_AGV:
   xx_writesample(
          obtem_int(agv->id_agv, XX->l_ve_dt_descarga),
          agv->dt_descarga);
   break;
 case ERRO:
   xx_writesample(
```

```
            obtem_int(agv->id_agv, XX->l_ve_dt_erro),
            agv->dt_erro);
}
xx_writesample(
   obtem_int(agv->id_agv, XX->l_ve_dt_enabled),
   agv->dt_enabled);
xx_writesample(
   obtem_int(agv->id_agv, XX->l_ve_taxa_util),
   agv->taxa_util);
xx_writesample(
   obtem_int(agv->id_agv, XX->l_ve_estado_AGV),
   ENABLED);

agv->posicao=NULL;
agv->ult_posicao=NULL;
agv->orient=0;
agv->est_mov_ant=agv->estado_movim;
agv->estado_movim=ENABLED;
TAREFA_AGV_DESACTIVADO->ap=agv->tarefa_agv;
agv->tarefa_agv=NULL;

remove_bloqueios_agv(agv, &XX->p_bloqueios, &XX->u_bloqueios);
ESTADO_LAYOUT->ap=XX->p_bloqueios;

AGV_ELIMIN_MOV_AVAN->ap=agv;

preenche_mens_ESTADO_AGV(&MENS_ESTADO_AGV->mens, agv);

xx_setstate(42);
```

C.3.21 Entidades da RP da página 'proc_cancel_tarefas'

C.3.21.1 Posições

- ID_MENS_CANCEL:ULINT;
- LISTA_TAR_CANCEL:L_AP_TAREFAS;
- LISTA_TAR_CANCEL_RESERV:L_AP_TAREFAS;
- NUM_TAR_CANCEL:ULINT;

C.3.21.2 Transições

C.3.21.2.1 CANCELA_TAR_EXEC

C.3.21.2.1.1 Predicado

```
(LISTA_TAR_CANCEL->ap!=NULL &&
 LISTA_TAR_CANCEL->ap->tarefa->estado_tar!=FILA
)
```

C.3.21.2.1.2 Acção

```
ap_EstadoFila fila=NULL;
double t=xx_gettime();
```

```
if ((LISTA_TAR_CANCEL->ap->tarefa->tipo_tar==NORMAL ||
     LISTA_TAR_CANCEL->ap->tarefa->tipo_tar==NORMAL_AGV ||
     LISTA_TAR_CANCEL->ap->tarefa->tipo_tar==T_CARGA ||
     LISTA_TAR_CANCEL->ap->tarefa->tipo_tar==CARGA_AGV) &&
    LISTA_TAR_CANCEL->ap->tarefa->estado_tar==PASSO1
   )
   fila=procura_EstadoFila(LISTA_TAR_CANCEL->ap->tarefa->no1->id_no,
          XX->p_estado_filas);
if (fila!=NULL)
{
   (fila->Rf)++;
   sprintf(XX->buffer, "\nSGAGV: tarefa com origem na fila %u foi
cancelada (Lf=%u, Rf=%u)#\n",
   fila->id, fila->Lf, fila->Rf);
   prn_string(XX->buffer);
}

act_estat_temporais_tarefa(LISTA_TAR_CANCEL->ap->tarefa, t);

MENS_ENV_SGM->mens.id_mens=LISTA_TAR_CANCEL->ap->tarefa->id_tar;
MENS_ENV_SGM->mens.tipo_mens=TAREFA_CANCEL;
MENS_ENV_SGM->mens.p1=LISTA_TAR_CANCEL->ap->tarefa->estado_tar;
MENS_ENV_SGM->mens.p2=LISTA_TAR_CANCEL->ap->tarefa->agv_tarefa->id_agv;
MENS_ENV_SGM->mens.p3=ORDEM_SGM;

LISTA_TAR_CANCEL->ap->tarefa->agv_tarefa->tarefa_agv=NULL;
AGV_ELIMIN_MOV_AVAN->ap=LISTA_TAR_CANCEL->ap->tarefa->agv_tarefa;

remove_tarefa(LISTA_TAR_CANCEL->ap->tarefa->id_tar,
   &XX->p_tarefas, &XX->u_tarefas);

remove_cab_l_ap_Tarefas(&LISTA_TAR_CANCEL->ap);

sprintf(XX->buffer,
"\nSGAGV: notificacao do cancelamento da Tarefa %u,\n atraves do envio
da mensagem ao SGM:\n",
MENS_ENV_SGM->mens.id_mens);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(t, MENS_ENV_SGM->mens);

xx_setstate(25);
```

C.3.21.2.2 CANCELA_TAR_FILA

C.3.21.2.2.1 Predicado

```
(LISTA_TAR_CANCEL->ap!=NULL &&
 LISTA_TAR_CANCEL->ap->tarefa->estado_tar==FILA
)
```

C.3.21.2.2.2 Acção

```
double t=xx_gettime();
ap_EstadoFila fila;
int act_MFCFS=0;

if (LISTA_TAR_CANCEL->ap->tarefa->tipo_tar==NORMAL ||
```

```
      LISTA_TAR_CANCEL->ap->tarefa->tipo_tar==T_CARGA)
 {
   fila=procura_EstadoFila(LISTA_TAR_CANCEL->ap->tarefa->no1->id_no,
    XX->p_estado_filas);
   if (fila!=NULL && fila->tar_mais_ant==LISTA_TAR_CANCEL->ap->tarefa)
act_MFCFS=1;
 }

 act_estat_temporais_tarefa(LISTA_TAR_CANCEL->ap->tarefa, t);

 MENS_ENV_SGM->mens.id_mens=LISTA_TAR_CANCEL->ap->tarefa->id_tar;
 MENS_ENV_SGM->mens.tipo_mens=TAREFA_CANCEL;
 MENS_ENV_SGM->mens.p1=LISTA_TAR_CANCEL->ap->tarefa->estado_tar;
 MENS_ENV_SGM->mens.p2=0;
 MENS_ENV_SGM->mens.p3=ORDEM_SGM;

 remove_tarefa(LISTA_TAR_CANCEL->ap->tarefa->id_tar,
    &XX->p_tarefas, &XX->u_tarefas);
 remove_cab_l_ap_Tarefas(&LISTA_TAR_CANCEL->ap);

 sprintf(XX->buffer,
"\nSGAGV: notificacao do cancelamento da Tarefa %u,\n atraves do envio
da mensagem ao SGM:\n",
MENS_ENV_SGM->mens.id_mens);
 prn_string(XX->buffer);
 prn_m_SGM_SGAGV(t, MENS_ENV_SGM->mens);

 if (act_MFCFS) act_dados_MFCFS_fila(fila, XX->p_tarefas);

 xx_setstate(24);
```

C.3.21.2.3 Acção de 'CONTINUA_CANCEL_TAR'

```
xx_setstate(64);
```

C.3.21.2.4 INICIA_CANCEL_TAREFAS

C.3.21.2.4.1 Predicado

```
(MENS_RECEB_SGM->mens.tipo_mens==CANCELA_TAREFA)
```

C.3.21.2.4.2 Acção

```
prn_string("\nSGAGV: recepcao da mensagem do SGM:\n");
prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGM->mens);
XX->id_prox_mens_SGM=MENS_RECEB_SGM->mens.id_mens+1;

NUM_TAR_CANCEL->i=deter_num_tar_cancel(XX->p_tarefas,
   MENS_RECEB_SGM->mens.p1, MENS_RECEB_SGM->mens.p2,
   &LISTA_TAR_CANCEL->ap);

ID_MENS_CANCEL->i=MENS_RECEB_SGM->mens.id_mens;

xx_setstate(23);
```

C.3.21.2.5 Acção de 'NOTIF_RESULT_PROC_CANCEL'

```
MENS_ENV_SGM->mens.id_mens=ID_MENS_CANCEL->i;
```

```
MENS_ENV_SGM->mens.tipo_mens=CONFIRMACAO;
if (NUM_TAR_CANCEL->i>0)
{
   MENS_ENV_SGM->mens.p1=SUCESSO;

   (XX->num_tar_cancel)+=NUM_TAR_CANCEL->i;
   xx_writesample(XX->ve_num_tar_cancel, (double) XX->num_tar_cancel);
}
else
{
   MENS_ENV_SGM->mens.p1=INSUCESSO;
   MENS_ENV_SGM->mens.p2=TAREFA_INEX;
}

LISTA_AGVS->ap=XX->p_agvs;
LISTA_TAREFAS->ap=XX->p_tarefas;

prn_string("\nSGAGV: notifica o SGM do resultado do cancelamento de
tarefas,\n atraves do envio da mensagem:\n");
prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGM->mens);

xx_setstate(26);
```

C.3.22 Transições da RP da página 'proc_erros_AGVs'

C.3.22.1 REC_ERRO_AGV

C.3.22.1.1 Predicado

```
(MENS_RECEB_SPAGV->mens.tipo_mens==ERRO_AGV)
```

C.3.22.1.2 Acção

```
ap_AGV agv=obtem_ap_AGV(MENS_RECEB_SPAGV->mens.id_agv,
   XX->p_agvs);
double t=xx_gettime();

sprintf(XX->buffer,
"\nSGAGV: notificacao pelo SPAGV da ocorrencia de um erro no AGV %u,\n
atraves da recepcao da mensagem:\n",
MENS_RECEB_SPAGV->mens.id_agv);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(t, MENS_RECEB_SPAGV->mens);

act_estat_temporais_agv(agv, t);
switch (agv->estado_movim)
{
 case PARADO:
   xx_writesample(
          obtem_int(agv->id_agv, XX->l_ve_dt_espera),
          agv->dt_espera);
   break;
 case BLOQUEADO:
   xx_writesample(
          obtem_int(agv->id_agv, XX->l_ve_dt_bloqueado),
          agv->dt_bloqueado);
```

```
    break;
  case EX_MOVIM:
    if (agv->estado_carga==DESCARREGADO)
      xx_writesample(
           obtem_int(agv->id_agv, XX->l_ve_dt_movim_descarr),
           agv->dt_movim_descarr);
    else
      xx_writesample(
           obtem_int(agv->id_agv, XX->l_ve_dt_movim_carr),
           agv->dt_movim_carr);
    break;
  case CARGA_AGV:
    xx_writesample(
           obtem_int(agv->id_agv, XX->l_ve_dt_carga),
           agv->dt_carga);
    break;
  case DESCARGA_AGV:
    xx_writesample(
           obtem_int(agv->id_agv, XX->l_ve_dt_descarga),
           agv->dt_descarga);
 }
 xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_erro),
    agv->dt_erro);
 xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_taxa_util),
    agv->taxa_util);
 xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_estado_AGV),
    ERRO);

 agv->est_mov_ant=agv->estado_movim;
 agv->estado_movim=ERRO;

 preenche_mens_ESTADO_AGV(&MENS_ENV_SGM->mens, agv);
 sprintf(XX->buffer,
"\nSGAGV: notificacao do SGM sobre a passagem do AGV %u\n ao estado
ERRO, atraves do envio da mensagem:\n",
agv->id_agv);
 prn_string(XX->buffer);
 prn_m_SGM_SGAGV(t, MENS_ENV_SGM->mens);

 xx_setstate(40);
```

C.3.22.2 REC_FIM_ERRO_AGV

C.3.22.2.1 Predicado

```
(MENS_RECEB_SPAGV->mens.tipo_mens==FIM_ERRO_AGV)
```

C.3.22.2.2 Acção

```
ap_AGV agv=obtem_ap_AGV(MENS_RECEB_SPAGV->mens.id_agv,
    XX->p_agvs);
double t=xx_gettime();

sprintf(XX->buffer,
```

```
"\nSGAGV: notificacao pelo SPAGV da resolucao de um erro no AGV %u,\n
atraves da recepcao da mensagem:\n",
MENS_RECEB_SPAGV->mens.id_agv);
 prn_string(XX->buffer);
 prn_m_SGAGV_SPAGV(t, MENS_RECEB_SPAGV->mens);

 act_estat_temporais_agv(agv, t);
 xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_erro),
    agv->dt_erro);
 xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_taxa_util),
    agv->taxa_util);
 xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_estado_AGV),
    agv->est_mov_ant);

/* if (agv->tarefa_agv==NULL)
    agv->estado_movim=PARADO;
 else
   switch (agv->tarefa_agv->estado_tar)
   {
    case PASSO2: agv->estado_movim=CARGA_AGV; break;
    case PASSO4: agv->estado_movim=DESCARGA_AGV; break;
    default:
    if (agv->posicao==agv->ult_posicao)
           agv->estado_movim=BLOQUEADO;
    else agv->estado_movim=EX_MOVIM;
   }*/

 agv->estado_movim=agv->est_mov_ant;
 agv->est_mov_ant=ERRO;

 switch (agv->estado_movim)
 {
    case PARADO: xx_writesample(
                  obtem_int(agv->id_agv, XX->l_ve_dt_espera),
                  agv->dt_espera);
            break;
    case EX_MOVIM: if (agv->estado_carga==DESCARREGADO)
                  xx_writesample(
                        obtem_int(agv->id_agv, XX-
>l_ve_dt_movim_descarr),
                        agv->dt_movim_descarr);
              else
                  xx_writesample(
                        obtem_int(agv->id_agv, XX-
>l_ve_dt_movim_carr),
                        agv->dt_movim_carr);
             break;
    case CARGA_AGV: xx_writesample(
                  obtem_int(agv->id_agv, XX->l_ve_dt_carga),
                  agv->dt_carga);
              break;
    case DESCARGA_AGV: xx_writesample(
                  obtem_int(agv->id_agv, XX->l_ve_dt_descarga),
                  agv->dt_descarga);
             break;
    case BLOQUEADO: xx_writesample(
```

```
                    obtem_int(agv->id_agv, XX->l_ve_dt_bloqueado),
                    agv->dt_bloqueado);
  }

  preenche_mens_ESTADO_AGV(&MENS_ENV_SGM->mens, agv);
  sprintf(XX->buffer,
"\nSGAGV: notificacao do SGM da resolucao do erro no AGV %u,\n atraves
do envio da mensagem:\n",
agv->id_agv);
  prn_string(XX->buffer);
  prn_m_SGM_SGAGV(t, MENS_ENV_SGM->mens);

  xx_setstate(41);
```

C.3.23 Entidades da RP da página 'proc_mens_relativas_a_AGVs'

C.3.23.1 Posições

- AGV_ENABLED:APO_AGV;
- ID_AGV_DISABLED:ULINT;
- MENS_ENABLE_AGV:MENS_SGM_SGAGV.

C.3.23.2 Transições

C.3.23.2.1 MUDA_EST_ALOC_AGV

C.3.23.2.1.1 Predicado

```
(AGV_ENABLED->ap!=NULL &&
 AGV_ENABLED->ap->estado_movim!=DISABLED
)
```

C.3.23.2.1.2 Acção

```
if (MENS_ENABLE_AGV->mens.p2==ALOCA_AGV)
   AGV_ENABLED->ap->estado_alocacao=ALOCADO;
  else     AGV_ENABLED->ap->estado_alocacao=NAO_ALOCADO;

LISTA_AGVS->ap=XX->p_agvs;

xx_setstate(12);
```

C.3.23.2.2 NOTIF_ENABLE_AGV_DISABLED

C.3.23.2.2.1 Predicado

```
(AGV_ENABLED->ap!=NULL &&
 AGV_ENABLED->ap->estado_movim==DISABLED
)
```

C.3.23.2.2.2 Acção

```
double t=xx_gettime();
```

```
act_estat_temporais_agv(AGV_ENABLED->ap, t);
xx_writesample(
    obtem_int(AGV_ENABLED->ap->id_agv, XX->l_ve_dt_disabled),
    AGV_ENABLED->ap->dt_disabled);
xx_writesample(
    obtem_int(AGV_ENABLED->ap->id_agv, XX->l_ve_dt_enabled),
    AGV_ENABLED->ap->dt_enabled);
xx_writesample(
    obtem_int(AGV_ENABLED->ap->id_agv, XX->l_ve_estado_AGV),
    ENABLED);

AGV_ENABLED->ap->est_mov_ant=AGV_ENABLED->ap->estado_movim;
AGV_ENABLED->ap->estado_movim=ENABLED;

if (MENS_ENABLE_AGV->mens.p2==ALOCA_AGV)
    AGV_ENABLED->ap->estado_alocacao=ALOCADO;
  else     AGV_ENABLED->ap->estado_alocacao=NAO_ALOCADO;

MENS_ENV_SGM->mens.id_mens=MENS_ENABLE_AGV->mens.id_mens;
MENS_ENV_SGM->mens.tipo_mens=CONFIRMACAO;
MENS_ENV_SGM->mens.p1=SUCESSO;

sprintf(XX->buffer,
"\nSGAGV: notificacao da passagem do AGV %u DISABLED ao estado
ENABLED,\n atraves do envio ao SGM da mensagem:\n",
AGV_ENABLED->ap->id_agv);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(t, MENS_ENV_SGM->mens);

MENS_ENV_SPAGV->mens.id_agv=AGV_ENABLED->ap->id_agv;
MENS_ENV_SPAGV->mens.tipo_mens=INF_EST_AGV;
MENS_ENV_SPAGV->mens.p1=ENABLED;

sprintf(XX->buffer,
"\nSGAGV: notificacao do SPAGV da mudanca do estado do AGV %u\n de
DISABLED para ENABLED, atraves do envio da mensagem:\n",
AGV_ENABLED->ap->id_agv);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(t, MENS_ENV_SPAGV->mens);

LISTA_AGVS->ap=XX->p_agvs;

xx_setstate(11);
```

C.3.23.2.3 Acção de 'NOTIF_ERRO_ENABLE_AGV'

```
MENS_ENV_SGM->mens.id_mens=MENS_ENABLE_AGV->mens.id_mens;
MENS_ENV_SGM->mens.tipo_mens=CONFIRMACAO;
MENS_ENV_SGM->mens.p1=INSUCESSO;
MENS_ENV_SGM->mens.p2=AGV_INEX;

sprintf(XX->buffer,
"\nSGM: notificacao de erro no enable do AGV %u,\n atraves do envio ao
SGM da mensagem:\n",
MENS_ENABLE_AGV->mens.p1);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGM->mens);
```

```
LISTA_AGVS->ap=XX->p_agvs;

xx_setstate(13);
```

C.3.23.2.4 Acção de 'NOTIF_SPAGV_DISABLE_AGV'

```
LISTA_AGVS->ap=XX->p_agvs;

 MENS_ENV_SPAGV->mens.id_agv=ID_AGV_DISABLED->i;
 MENS_ENV_SPAGV->mens.tipo_mens=INF_EST_AGV;
 MENS_ENV_SPAGV->mens.p1=DISABLED;

 sprintf(XX->buffer,
"\nSGAGV: notifica o SPAGV da passagem do AGV %u ao estado DISABLED,\n
atraves do envio da mensagem:\n",
ID_AGV_DISABLED->i);
 prn_string(XX->buffer);
 prn_m_SGAGV_SPAGV(xx_gettime(), MENS_ENV_SPAGV->mens);

 xx_setstate(30);
```

C.3.23.2.5 REC_MENS_DISABLE_AGV

C.3.23.2.5.1 Predicado

```
(MENS_RECEB_SGM->mens.tipo_mens==DISABLE_AGV)
```

C.3.23.2.5.2 Acção

```
ap_AGV agv=obtem_ap_AGV(MENS_RECEB_SGM->mens.p1,
   LISTA_AGVS->ap);
double t=xx_gettime();

prn_string("\nSGM: recepcao da mensagem DISABLE_AGV:\n");
prn_m_SGM_SGAGV(t, MENS_RECEB_SGM->mens);
XX->id_prox_mens_SGM=MENS_RECEB_SGM->mens.id_mens+1;

act_estat_temporais_agv(agv, t);
switch (agv->estado_movim)
{
 case ENABLED:
   xx_writesample(
          obtem_int(agv->id_agv, XX->l_ve_dt_enabled),
          agv->dt_enabled);
   break;
 case PARADO:
   xx_writesample(
          obtem_int(agv->id_agv, XX->l_ve_dt_espera),
          agv->dt_espera);
   break;
 case BLOQUEADO:
   xx_writesample(
          obtem_int(agv->id_agv, XX->l_ve_dt_bloqueado),
          agv->dt_bloqueado);
   break;
 case EX_MOVIM:
   if (agv->estado_carga==DESCARREGADO)
     xx_writesample(
          obtem_int(agv->id_agv, XX->l_ve_dt_movim_descarr),
          agv->dt_movim_descarr);
```

```
    else
      xx_writesample(
            obtem_int(agv->id_agv, XX->l_ve_dt_movim_carr),
            agv->dt_movim_carr);
    break;
  case CARGA_AGV:
    xx_writesample(
            obtem_int(agv->id_agv, XX->l_ve_dt_carga),
            agv->dt_carga);
    break;
  case DESCARGA_AGV:
    xx_writesample(
            obtem_int(agv->id_agv, XX->l_ve_dt_descarga),
            agv->dt_descarga);
    break;
  case ERRO:
    xx_writesample(
            obtem_int(agv->id_agv, XX->l_ve_dt_erro),
            agv->dt_erro);
}
xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_disabled),
    agv->dt_disabled);
xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_taxa_util),
    agv->taxa_util);
xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_estado_AGV),
    DISABLED);

agv->est_mov_ant=agv->estado_movim;
agv->estado_movim=DISABLED;

remove_bloqueios_agv(agv, &XX->p_bloqueios,
    &XX->u_bloqueios);
ESTADO_LAYOUT->ap=XX->p_bloqueios;

agv->ult_posicao=agv->posicao;
agv->posicao=NULL;

TAREFA_AGV_DESACTIVADO->ap=agv->tarefa_agv;
agv->tarefa_agv=NULL;

AGV_ELIMIN_MOV_AVAN->ap=agv;

ID_AGV_DISABLED->i=agv->id_agv;

xx_setstate(14);
```

C.3.23.2.6 REC_MENS_ENABLE_AGV

C.3.23.2.6.1 Predicado

```
(MENS_RECEB_SGM->mens.tipo_mens==ENABLE_AGV)
```

C.3.23.2.6.2 Acção

```
prn_string("\nSGAGV: recepcao da mensagem ENABLE_AGV:\n");
```

```
prn_m_SGM_SGAGV(xx_gettime(), MENS_RECEB_SGM->mens);
XX->id_prox_mens_SGM=MENS_RECEB_SGM->mens.id_mens+1;

AGV_ENABLED->ap=obtem_ap_AGV(MENS_RECEB_SGM->mens.p1,
   LISTA_AGVS->ap);

xx_setstate(10);
```

C.3.24 Entidades da RP da página 'proc_novas_tarefas'

C.3.24.1 Posições

- DISPONIVEL_TESTE_NOS:NUL;
- ERRO_FASIBIL_TAR:BOOLEANO;
- ERRO_NOS:BOOLEANO;
- NOVA_TAREFA:TAREFA;
- NOVA_TAREFA_ESCAL_SGM:TAREFA.

C.3.24.2 Transições

C.3.24.2.1 INICIA_EXEC_TAR_ESC_SGM

C.3.24.2.1.1 Predicado

```
(ERRO_FASIBIL_TAR->b==FALSO)
```

C.3.24.2.1.2 Acção

```
char s[15];
ap_EstadoFila fila;

if ((NOVA_TAREFA_ESCAL_SGM->tarefa.tipo_tar==NORMAL_AGV ||
     NOVA_TAREFA_ESCAL_SGM->tarefa.tipo_tar==T_CARGA ||
     NOVA_TAREFA_ESCAL_SGM->tarefa.tipo_tar==CARGA_AGV)
    )
{
 fila=procura_EstadoFila(NOVA_TAREFA_ESCAL_SGM->tarefa.no1->id_no,
   XX->p_estado_filas);
 if (fila!=NULL)
 {
   (fila->Rf)--;
   sprintf(XX->buffer, "\nSGAGV: tarefa com origem na fila %u passou ao
estado PASSO1 (Lf=%u, Rf=%u)#\n",
   fila->id, fila->Lf, fila->Rf);
   prn_string(XX->buffer);
 }
}

if (NOVA_TAREFA_ESCAL_SGM->tarefa.tipo_tar!=T_DESCARGA)
   NOVA_TAREFA_ESCAL_SGM->tarefa.estado_tar=PASSO1;
  else NOVA_TAREFA_ESCAL_SGM->tarefa.estado_tar=PASSO3;
NOVA_TAREFA_ESCAL_SGM->tarefa.num_af=1;
```

```
 insere_tarefa(&NOVA_TAREFA_ESCAL_SGM->tarefa,
    &XX->p_tarefas, &XX->u_tarefas);
 sprintf(XX->buffer,
"\nSGAGV: nova Tarefa %u escal. pelo SGM inserida na Lista de
Tarefas.\n",
NOVA_TAREFA_ESCAL_SGM->tarefa.id_tar);
 prn_string(XX->buffer);
 XX->u_tarefas->agv_tarefa->tarefa_agv=XX->u_tarefas;

 LISTA_TAREFAS->ap=XX->p_tarefas;
 LISTA_AGVS->ap=XX->p_agvs;

 preenche_mens_ESTADO_TAREFA(&MENS_ENV_SGM->mens,
    XX->u_tarefas);
 escr_str_est_tarefa(s, XX->u_tarefas->estado_tar);
 sprintf(XX->buffer,
"\nSGAGV: notifica o SGM sobre a passagem da Tarefa %u ao estado %s,\n
atraves do envio da mensagem:\n",
XX->u_tarefas->id_tar, s);
 prn_string(XX->buffer);
 prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGM->mens);



 xx_setstate(20);
```

C.3.24.2.2 INS_LISTA_NOVA_TAR_N_ESCAL

C.3.24.2.2.1 Predicado

```
(ERRO_NOS->b==FALSO &&
 (NOVA_TAREFA->tarefa.tipo_tar==NORMAL ||
  NOVA_TAREFA->tarefa.tipo_tar==T_CARGA
 )
)
```

C.3.24.2.2.2 Acção

```
ap_EstadoFila fila=procura_EstadoFila(NOVA_TAREFA->tarefa.no1->id_no,
   XX->p_estado_filas);

insere_tarefa(&NOVA_TAREFA->tarefa, &XX->p_tarefas, &XX->u_tarefas);

LISTA_TAREFAS->ap=XX->p_tarefas;

sprintf(XX->buffer,
"\nSGAGV: nova Tarefa %u no estado FILA inserida na Lista de
Tarefas.\n",
NOVA_TAREFA->tarefa.id_tar);
prn_string(XX->buffer);

if (fila!=NULL && fila->tar_mais_ant==NULL)
{
  fila->tar_mais_ant=XX->u_tarefas;
  fila->t_MFCFS=xx_gettime();
}
```

```
xx_setstate(18);
```

C.3.24.2.3 Acção de 'NOTIF_ERRO_FASIBIL_TAR'

```
(XX->num_tar_rej)++;
xx_writesample(XX->ve_num_tar_rej, (double) XX->num_tar_rej);

MENS_ENV_SGM->mens.id_mens=NOVA_TAREFA_ESCAL_SGM->tarefa.id_tar;
MENS_ENV_SGM->mens.tipo_mens=CONFIRMACAO;
MENS_ENV_SGM->mens.p1=INSUCESSO;
MENS_ENV_SGM->mens.p2=ERRO_FASIBIL_TAR->b;

LISTA_AGVS->ap=XX->p_agvs;

sprintf(XX->buffer,
"\nSGAGV: notifica o SGM da nao fasibilidade da Tarefa %u,\n enviando a
mensagem:\n",
NOVA_TAREFA_ESCAL_SGM->tarefa.id_tar);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGM->mens);

xx_setstate(21);
```

C.3.24.2.4 NOTIF_ERRO_NOS_TAR

C.3.24.2.4.1 Predicado

```
(ERRO_NOS->b!=FALSO)
```

C.3.24.2.4.2 Acção

```
(XX->num_tar_rej)++;
xx_writesample(XX->ve_num_tar_rej, (double) XX->num_tar_rej);

MENS_ENV_SGM->mens.id_mens=NOVA_TAREFA->tarefa.id_tar;
MENS_ENV_SGM->mens.tipo_mens=CONFIRMACAO;
MENS_ENV_SGM->mens.p1=INSUCESSO;
MENS_ENV_SGM->mens.p2=ERRO_NOS->b;

sprintf(XX->buffer,
"\nSGAGV: notifica o SGM de erros nos Nos da nova Tarefa %u,\n enviando
a mensagem:\n",
NOVA_TAREFA->tarefa.id_tar);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGM->mens);

xx_setstate(22);
```

C.3.24.2.5 VERIFICA_FASIBIL_TAR_ESC_SGM

C.3.24.2.5.1 Predicado

```
(ERRO_NOS->b==FALSO &&
 NOVA_TAREFA->tarefa.tipo_tar!=NORMAL &&
 NOVA_TAREFA->tarefa.tipo_tar!=T_CARGA
)
```

C.3.24.2.5.2 Acção

```
ERRO_FASIBIL_TAR->b=FALSO;

if (NOVA_TAREFA->tarefa.agv_tarefa!=NULL)
  if (NOVA_TAREFA->tarefa.agv_tarefa->tarefa_agv==NULL)
    if (
     (
      NOVA_TAREFA->tarefa.tipo_tar!=T_DESCARGA &&
      NOVA_TAREFA->tarefa.agv_tarefa->estado_carga==DESCARREGADO
     ) ||
     (
      NOVA_TAREFA->tarefa.tipo_tar==T_DESCARGA &&
      NOVA_TAREFA->tarefa.agv_tarefa->estado_carga==CARREGADO
     ) ||
     NOVA_TAREFA->tarefa.tipo_tar==POSICION
    )
     {
    if (NOVA_TAREFA->tarefa.tipo_tar!=T_DESCARGA)
    {
      if (NOVA_TAREFA->tarefa.agv_tarefa->posicao!=NOVA_TAREFA-
>tarefa.no1)
      {
        NOVA_TAREFA->tarefa.ctraj1=traj_optimo(
           XX->layout, NOVA_TAREFA->tarefa.agv_tarefa->posicao,
           NOVA_TAREFA->tarefa.no1, NULL, NULL, XX->poss_acesso,
           &NOVA_TAREFA->tarefa.traj1
           );
        if (NOVA_TAREFA->tarefa.traj1==NULL)
           ERRO_FASIBIL_TAR->b=CAMINHO1_INEX;
      }
      else
      {
        NOVA_TAREFA->tarefa.ctraj1=0;
        if (NOVA_TAREFA->tarefa.tipo_tar!=POSICION && XX->poss_acesso!=0
&&
              NOVA_TAREFA->tarefa.agv_tarefa->orient*
                 NOVA_TAREFA->tarefa.agv_tarefa->posicao-
>pos_rel_mesa!=XX->poss_acesso)
                        ERRO_FASIBIL_TAR->b=OP_CAR_IMP;
      }
      if (ERRO_FASIBIL_TAR->b==FALSO && NOVA_TAREFA-
>tarefa.tipo_tar==NORMAL_AGV)
        if (NOVA_TAREFA->tarefa.no2!=NOVA_TAREFA->tarefa.no1)
        {
          NOVA_TAREFA->tarefa.ctraj2=traj_optimo(
           XX->layout, NOVA_TAREFA->tarefa.no1,
           NOVA_TAREFA->tarefa.no2, NULL, NULL, XX->poss_acesso,
           &NOVA_TAREFA->tarefa.traj2
           );
          if (NOVA_TAREFA->tarefa.traj2==NULL)
          {
           ERRO_FASIBIL_TAR->b=CAMINHO2_INEX;
           NOVA_TAREFA->tarefa.ctraj1=MAXINT;
           limpa_l_apno(&NOVA_TAREFA->tarefa.traj1);
          }
        }
        else NOVA_TAREFA->tarefa.ctraj2=0;
    }
```

```
    else
      if (NOVA_TAREFA->tarefa.agv_tarefa->posicao!=NOVA_TAREFA-
>tarefa.no2)
      {
        NOVA_TAREFA->tarefa.ctraj2=traj_optimo(
           XX->layout, NOVA_TAREFA->tarefa.agv_tarefa->posicao,
           NOVA_TAREFA->tarefa.no2, NULL, NULL, XX->poss_acesso,
           &NOVA_TAREFA->tarefa.traj2
           );
        if (NOVA_TAREFA->tarefa.traj2==NULL)
           ERRO_FASIBIL_TAR->b=CAMINHO2_INEX;
      }
      else
      {
        NOVA_TAREFA->tarefa.ctraj2=0;
        if (XX->poss_acesso!=0 &&
              NOVA_TAREFA->tarefa.agv_tarefa->orient*
                  NOVA_TAREFA->tarefa.agv_tarefa->posicao-
>pos_rel_mesa!=XX->poss_acesso)
                        ERRO_FASIBIL_TAR->b=OP_DES_IMP;
      }
     }
     else
       if (NOVA_TAREFA->tarefa.tipo_tar!=T_DESCARGA)
           ERRO_FASIBIL_TAR->b=AGV_CARR;
       else ERRO_FASIBIL_TAR->b=AGV_DESCARR;
  else ERRO_FASIBIL_TAR->b=AGV_INDISPON;
else ERRO_FASIBIL_TAR->b=AGV_INEX;

xx_setstate(19);
```

C.3.24.2.6 VERIFICA_NOS_NOVA_TAREFA

C.3.24.2.6.1 Predicado

```
(MENS_RECEB_SGM->mens.tipo_mens==TAREFA_NORMAL ||
 MENS_RECEB_SGM->mens.tipo_mens==TAREFA_POSICION ||
 MENS_RECEB_SGM->mens.tipo_mens==TAREFA_CARGA ||
 MENS_RECEB_SGM->mens.tipo_mens==TAREFA_DESCARGA
)
```

C.3.24.2.6.2 Acção

```
ulint tipo_tar, prior;
ap_no_l_nos no1, no2;
int ve;
double t=xx_gettime();

prn_string("\nSGAGV: recepcao da mensagem relativa a uma nova
tarefa:\n");
prn_m_SGM_SGAGV(t, MENS_RECEB_SGM->mens);
XX->id_prox_mens_SGM=MENS_RECEB_SGM->mens.id_mens+1;

switch (MENS_RECEB_SGM->mens.tipo_mens)
{
 case TAREFA_NORMAL:
   if (MENS_RECEB_SGM->mens.p1==QUALQUER)
   {
```

```
        tipo_tar=NORMAL;
        prior=MENS_RECEB_SGM->mens.p4;
          }
      else
      {
        tipo_tar=NORMAL_AGV;
        prior=0;
      }
      break;
   case TAREFA_POSICION:
      tipo_tar=POSICION;
      prior=0;
      break;
   case TAREFA_CARGA:
      if (MENS_RECEB_SGM->mens.p1==QUALQUER)
      {
        tipo_tar=T_CARGA;
        prior=MENS_RECEB_SGM->mens.p4;
          }
      else
      {
        tipo_tar=CARGA_AGV;
        prior=0;
      }
      break;
   case TAREFA_DESCARGA:
      tipo_tar=T_DESCARGA;
      prior=0;
}

ve=obtem_int(tipo_tar+1, XX->l_ve_contag_tar_lan);
switch (tipo_tar)
{
 case NORMAL:
            (XX->num_tar_NORMAL_lan)++;
            xx_writesample(ve,
                    (double) XX->num_tar_NORMAL_lan);
            break;
 case NORMAL_AGV:
            (XX->num_tar_NORMAL_AGV_lan)++;
            xx_writesample(ve,
                    (double) XX->num_tar_NORMAL_AGV_lan);
            break;
 case POSICION:
            (XX->num_tar_POSICION_lan)++;
            xx_writesample(ve,
                    (double) XX->num_tar_POSICION_lan);
            break;
 case T_CARGA:
            (XX->num_tar_T_CARGA_lan)++;
            xx_writesample(ve,
                    (double) XX->num_tar_T_CARGA_lan);
            break;
 case CARGA_AGV:
            (XX->num_tar_CARGA_AGV_lan)++;
            xx_writesample(ve,
                    (double) XX->num_tar_CARGA_AGV_lan);
           break;
 case T_DESCARGA:
```

```
            (XX->num_tar_T_DESCARGA_lan)++;
            xx_writesample(ve,
                   (double) XX->num_tar_T_DESCARGA_lan);
}
(XX->num_tar_lan)++;
xx_writesample(XX->ve_num_tar_lan, (double) XX->num_tar_lan);

ERRO_NOS->b=FALSO;

if (MENS_RECEB_SGM->mens.tipo_mens!=TAREFA_DESCARGA)
{
  if ((no1=get_no(XX->layout, MENS_RECEB_SGM->mens.p2))==NULL ||
      (MENS_RECEB_SGM->mens.tipo_mens!=TAREFA_POSICION &&
   (no1->tipo_no==TRANSITO || no1->tipo_no==DESCARGA))
     )
    if (MENS_RECEB_SGM->mens.tipo_mens==TAREFA_NORMAL)
        ERRO_NOS->b=ORIGEM_INVAL;
    else ERRO_NOS->b=NO_INVALIDO;

  if (MENS_RECEB_SGM->mens.tipo_mens==TAREFA_NORMAL)
  {
   if ((no2=get_no(XX->layout, MENS_RECEB_SGM->mens.p3))==NULL ||
       no2->tipo_no==TRANSITO || no2->tipo_no==CARGA
      ) ERRO_NOS->b=DESTINO_INVAL;
  }
  else no2=NULL;
}
else
{
  no1=NULL;
  if ((no2=get_no(XX->layout, MENS_RECEB_SGM->mens.p2))==NULL ||
      no2->tipo_no==TRANSITO || no2->tipo_no==CARGA
     ) ERRO_NOS->b=NO_INVALIDO;
}

inicial_tarefa(&NOVA_TAREFA->tarefa, MENS_RECEB_SGM->mens.id_mens,
   tipo_tar, prior, no1, no2, t);

if (tipo_tar!=NORMAL && tipo_tar!=T_CARGA)
   NOVA_TAREFA->tarefa.agv_tarefa=obtem_ap_AGV(
          MENS_RECEB_SGM->mens.p1, XX->p_agvs);

xx_setstate(17);
```

C.3.25 Transições da RP da página 'proc_tarefas_AGVs_desactivados'

C.3.25.1 Acção de 'LIMPA_TAREFA_AGV_DESACTIVADO'

```
xx_setstate(16);
```

C.3.25.2 TRATA_TAR_AGV_DESACTIVADO

C.3.25.2.1 Predicado

```
(TAREFA_AGV_DESACTIVADO->ap!=NULL)
```

C.3.25.2.2 Acção

```
ap_EstadoFila fila=NULL;
double t=xx_gettime();

if ((TAREFA_AGV_DESACTIVADO->ap->tipo_tar==NORMAL ||
     TAREFA_AGV_DESACTIVADO->ap->tipo_tar==NORMAL_AGV ||
     TAREFA_AGV_DESACTIVADO->ap->tipo_tar==T_CARGA ||
     TAREFA_AGV_DESACTIVADO->ap->tipo_tar==CARGA_AGV) &&
    (TAREFA_AGV_DESACTIVADO->ap->estado_tar==PASSO1 ||
     TAREFA_AGV_DESACTIVADO->ap->estado_tar==PASSO2)
   )
{
   fila=procura_EstadoFila(TAREFA_AGV_DESACTIVADO->ap->no1->id_no,
   XX->p_estado_filas);
   if (fila!=NULL)
   {
   (fila->Rf)++;
   sprintf(XX->buffer, "\nSGAGV: tarefa com origem na fila %u foi
interrompida (Lf=%u, Rf=%u)#\n",
           fila->id, fila->Lf, fila->Rf);
   prn_string(XX->buffer);
   }
}

act_estat_temporais_tarefa(TAREFA_AGV_DESACTIVADO->ap, t);

if (TAREFA_AGV_DESACTIVADO->ap->estado_tar==PASSO1 &&
    (TAREFA_AGV_DESACTIVADO->ap->tipo_tar==NORMAL ||
     TAREFA_AGV_DESACTIVADO->ap->tipo_tar==T_CARGA
    )
   )
{
 TAREFA_AGV_DESACTIVADO->ap->estado_tar=FILA;
 TAREFA_AGV_DESACTIVADO->ap->agv_tarefa=NULL;
 limpa_l_apno(&TAREFA_AGV_DESACTIVADO->ap->traj1);
 limpa_l_apno(&TAREFA_AGV_DESACTIVADO->ap->traj2);

 preenche_mens_ESTADO_TAREFA(&MENS_ENV_SGM->mens,
   TAREFA_AGV_DESACTIVADO->ap);
 sprintf(XX->buffer,
   "\nSGAGV: notificacao do SGM da passagem da tarefa %u ao estado
FILA\n apos desactivacao de um AGV, atraves do envio da mensagem:\n",
   TAREFA_AGV_DESACTIVADO->ap->id_tar);

 if (fila!=NULL && fila->tar_mais_ant==NULL) act_dados_MFCFS_fila(fila,
XX->p_tarefas);
}
else
{
 MENS_ENV_SGM->mens.id_mens=TAREFA_AGV_DESACTIVADO->ap->id_tar;
 MENS_ENV_SGM->mens.tipo_mens=TAREFA_CANCEL;
 MENS_ENV_SGM->mens.p1=TAREFA_AGV_DESACTIVADO->ap->estado_tar;
 MENS_ENV_SGM->mens.p2=TAREFA_AGV_DESACTIVADO->ap->agv_tarefa->id_agv;
 MENS_ENV_SGM->mens.p3=AGV_DESACTIVADO;

 sprintf(XX->buffer,
   "\nSGAGV: notificacao do SGM do cancelamento da tarefa %u\n apos
desactivacao de um AGV, atraves do envio da mensagem:\n",
```

```
    TAREFA_AGV_DESACTIVADO->ap->id_tar);

  remove_tarefa(TAREFA_AGV_DESACTIVADO->ap->id_tar, &XX->p_tarefas, &XX-
>u_tarefas);
  LISTA_TAREFAS->ap=XX->p_tarefas;
 }

 prn_string(XX->buffer);
 prn_m_SGM_SGAGV(t, MENS_ENV_SGM->mens);

 xx_setstate(15);
```

C.3.26 Transições da RP da página 'resolve_Deadlock_Circular'

C.3.26.1 RESOLVE_DEADLOCK_CIRCULAR

C.3.26.1.1 Predicado

```
(existe_deadlock_circular(LISTA_AGVS->ap, XX->layout,
   ESTADO_LAYOUT->ap, XX->poss_acesso))
```

C.3.26.1.2 Acção

```
ap_no_l_nos p_nos_oc_agvs_erro;

resolve_deadlock_circular(
   obtem_deadlock_circular(LISTA_AGVS->ap, XX->layout,
          ESTADO_LAYOUT->ap, XX->poss_acesso,
          &p_nos_oc_agvs_erro),
   XX->layout,
   XX->poss_acesso, p_nos_oc_agvs_erro);

xx_setstate(57);
```

C.3.27 Entidades da RP da página 'resolve_conflitos_AGVs_ERRO'

C.3.27.1 Posições

- AGVS_REDIR:L_AP_AGVS;
- AGVS_REDIR_RESERV:L_AP_AGVS.

C.3.27.2 Transições

C.3.27.2.1 Acção de 'ACT_AGVS_REDIREC'

```
remove_cab_l_ap_AGVs(&AGVS_REDIR_RESERV->ap);

xx_setstate(60);
```

C.3.27.2.2 CANCEL_MOV_AVAN_AGV_REDIR

C.3.27.2.2.1 Predicado

```
(AGVS_REDIR->ap!=NULL)
```

C.3.27.2.2.2 Acção

```
AGV_ELIMIN_MOV_AVAN->ap=AGVS_REDIR->ap->agv;

xx_setstate(59);
```

C.3.27.2.3 Acção de 'FINALIZA_RESOL_CONFL_AGVS_ERRO'

```
LISTA_AGVS->ap=XX->p_agvs;
ESTADO_LAYOUT->ap=XX->p_bloqueios;

xx_setstate(61);
```

C.3.27.2.4 RES_CONFL_C_AGVS_EST_ERRO

C.3.27.2.4.1 Predicado

```
(ha_agvs_bloq_por_agvs_ERRO(LISTA_AGVS->ap,
   ESTADO_LAYOUT->ap, XX->layout, XX->poss_acesso))
```

C.3.27.2.4.2 Acção

```
resolve_bloq_por_agvs_ERRO(LISTA_AGVS->ap,
   ESTADO_LAYOUT->ap, XX->layout, XX->poss_acesso,
   &AGVS_REDIR->ap);

xx_setstate(58);
```

C.3.28 Transições da RP da página 'resolve_conflitos_AGVs_PARADOS'

C.3.28.1 MOVE_AGV_PARADO_BLOQUEANTE

C.3.28.1.1 Predicado

```
(ha_agv_bloq_por_agv_PARADO(LISTA_AGVS->ap, XX->layout,
   ESTADO_LAYOUT->ap, XX->num_max_mov_avan)
)
```

C.3.28.1.2 Acção

```
ap_AGV agv_bloqueado, agv_bloqueante;
ap_no_l_ramos r_no_escape;
Tarefa nova_tar;
ap_no_l_nos no_escape;

agv_bloqueado=agv_bloq_por_agv_PARADO(LISTA_AGVS->ap,
   XX->layout,
   ESTADO_LAYOUT->ap,
   XX->num_max_mov_avan,
```

```
        &agv_bloqueante,
        &r_no_escape);

    sprintf(XX->buffer,
        "\nSGAGV: foi detectado que o AGV %u esta bloqueado pelo AGV PARADO
%u.\n",
        agv_bloqueado->id_agv, agv_bloqueante->id_agv);
    prn_string(XX->buffer);
    sprintf(XX->buffer,
        " vai ser ordenado o movimento %u->%u para o AGV bloqueante, com um
custo %u.\n",
        agv_bloqueante->posicao->id_no,
        r_no_escape->destino, r_no_escape->custo);
    prn_string(XX->buffer);

    no_escape=get_no(XX->layout, r_no_escape->destino);
    inicial_tarefa(&nova_tar, XX->id_prox_mens_SGM,
        POSIC_ESPEC, 0,
        no_escape,
        NULL,
        xx_gettime());
    nova_tar.estado_tar=PASSO1;
    nova_tar.agv_tarefa=agv_bloqueante;
    nova_tar.ctraj1=r_no_escape->custo;
    insere_a_cabeca_l_apno(no_escape, r_no_escape->custo, &nova_tar.traj1);
    nova_tar.num_af=1;
    insere_tarefa(&nova_tar, &XX->p_tarefas, &XX->u_tarefas);
    LISTA_TAREFAS->ap=XX->p_tarefas;
    sprintf(XX->buffer,
        " A nova Tarefa %u foi inserida na lista de Tarefas.\n",
        XX->id_prox_mens_SGM);
    prn_string(XX->buffer);

    agv_bloqueante->tarefa_agv=XX->u_tarefas;

    MENS_ENV_SGM->mens.id_mens=0;
    MENS_ENV_SGM->mens.tipo_mens=T_POSIC_ESPEC;
    MENS_ENV_SGM->mens.p1=XX->id_prox_mens_SGM;
    (XX->id_prox_mens_SGM)++;
    MENS_ENV_SGM->mens.p2=agv_bloqueante->id_agv;
    MENS_ENV_SGM->mens.p3=r_no_escape->destino;
    prn_string(" O SGM vai ser notificado desse facto, atraves do envio da
mensagem:\n");
    prn_m_SGM_SGAGV(xx_gettime(), MENS_ENV_SGM->mens);

    (XX->num_tar_POSIC_ESPEC_lan)++;
    xx_writesample(obtem_int(POSIC_ESPEC+1, XX->l_ve_contag_tar_lan),
        (double) XX->num_tar_POSIC_ESPEC_lan);

    (XX->num_tar_lan)++;
    xx_writesample(XX->ve_num_tar_lan, (double) XX->num_tar_lan);

    xx_setstate(56);
```

C.3.29 Transições da RP da página 'transicao_PARADO_BLOQUEADO'

C.3.29.1 AGV_PARADO_BLOQUEADO

C.3.29.1.1 Predicado

```
(agv_PARADO_BLOQUEADO(LISTA_AGVS->ap)!=NULL)
```

C.3.29.1.2 Acção

```
ap_AGV agv=agv_PARADO_BLOQUEADO(LISTA_AGVS->ap);
double t=xx_gettime();

if (agv->t_mud_estado<t)
{
  act_estat_temporais_agv(agv, t);
  xx_writesample(
   obtem_int(agv->id_agv, XX->l_ve_dt_espera),
   agv->dt_espera);
  xx_writesample(
   obtem_int(agv->id_agv, XX->l_ve_taxa_util),
   agv->taxa_util);
}
xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_dt_bloqueado),
    agv->dt_bloqueado);
xx_writesample(
    obtem_int(agv->id_agv, XX->l_ve_estado_AGV),
    BLOQUEADO);

agv->est_mov_ant=agv->estado_movim;
agv->estado_movim=BLOQUEADO;

preenche_mens_ESTADO_AGV(&MENS_ENV_SGM->mens, agv);
sprintf(XX->buffer,
"\nSGAGV: notificacao do SGM sobre a passagem do AGV %u PARADO\n ao
estado BLOQUEADO, atraves do envio da mensagem:\n",
agv->id_agv);
prn_string(XX->buffer);
prn_m_SGM_SGAGV(t, MENS_ENV_SGM->mens);

xx_setstate(39);
```

C.4 Classe SPAGV

C.4.1 CMN units incluídas

- MENS_SGAGV_SPAGV;
- TEMPORIZACAO;
- BOOLEANO;
- ULINT.

C.4.2 Posições de entrada

- MENS_RECEB_SGAGV:MENS_SGAGV_SPAGV;
- FIM_OPERACAO_CARGA:ULINT;
- FIM_OPERACAO_DESCARGA:ULINT.

C.4.3 Posições de saída

- MENS_ENV_SGAGV:MENS_SGAGV_SPAGV;
- INICIO_OP_CARGA:ULINT;
- INICIO_OP_DESCARGA:ULINT;
- OP_TRANSF_CARGA_CANCEL:ULINT;
- OP_TRANSF_CARGA_INTERR:ULINT;
- OP_TRANSF_CARGA_RETOM:ULINT.

C.4.4 Parâmetros

- nome_fich_layout:xx_string – nome do ficheiro de configuração de *layout* (com a estrutura do *layout*);
- nome_fich_nos_intr:xx_string – nome do ficheiro de configuração de nós de introdução e de nós para tarefas de posicionamento; contém as probabilidades de estes tipos de operações ocorrerem em cada nó;
- numero_agvs:unsigned long int – número máximo de AGVs no sistema;
- vel_frente:unsigned long int – velocidade média de deslocamento de um AGV em movimento rectilíneo para a frente (em cm/s);
- vel_curva:unsigned long int – velocidade média de deslocamento de um AGV em movimento curvilíneo para a frente (em cm/s);
- vel_tras:unsigned long int – velocidade média de deslocamento de um AGV em movimento rectilíneo para trás (em cm/s);
- t_med_introducao:double – valor médio do tempo que decorre entre a activação de um AGV (passagem ao estado ENABLED) e a sua introdução num nó do *layout* (passagem ao estado PARADO); admite-se que este tempo segue uma FDP exponencial;
- t_med_manual:double – valor médio do tempo que decorre entre a introdução de um AGV e a passagem do mesmo ao modo manual (ao estado ENABLED); admite-se que este tempo segue uma FDP exponencial;
- t_med_erro:double – valor médio do tempo que decorre entre a introdução de um AGV e a ocorrência de um erro interno no mesmo, ou do tempo entre avarias de um AGV; admite-se que este tempo segue uma FDP exponencial;
- p_erro_manual:double – probabilidade de um AGV passar ao modo manual (ao estado ENABLED), quando ocorrer um erro interno no AGV;

- p_introd_or_pos:double – probabilidade de um AGV ser introduzido no *layout* com uma orientação positiva;
- poss_acesso:signed int – define as possibilidades de acesso aos pontos de carga/descarga por parte dos AGVs; vale 0 se puderem operar para ambos os lados, vale +1 se só puderem operar para o lado esquerdo, ou vale -1 se só puderem operar para o lado direito.

C.4.5 Ficheiros de cabeçalho incluídos

```
#include "rand_fun.h"
#include "listas.h"
#include "mens_log.h"
#include "layout.h"
#include "spagv.h"
#include "sistagvs.h"
```

C.4.6 Funções definidas internamente

```
/* funcao para permitir a visualizacao do layout, a partir de
uma posicao que contenha um token do tipo AP_LAYOUT
(utilizando o comando "examine Queue") */
void prn_AP_LAYOUT(a)
 AP_LAYOUT *a;
{ prn_lay_SP(a->ap); }


/* funcao para permitir a visualizacao do valor da variavel
booleana contida num token do tipo BOOLEANO
(utilizando o comando "examine Queue") */
void prn_BOOLEANO_SP(bool)
 BOOLEANO *bool;
{if (bool->b==FALSO) prn_string("FALSO\n"); else
prn_string("VERDADE\n");}

/* funcao para permitir a visualizacao da lista dos nos de
introducao, a partir de uma posicao que contenha um token do tipo
AP_NOS_INTR (utilizando o comando "examine Queue") */
void prn_AP_NOS_INTR(a)
 AP_NOS_INTR *a;
{
 prn_string("Nos de Introducao de AGV's:\n");
 prn_string("(<id_no>,<probabilidade de ocorrencia>)\n");
 prn_l_uid(a->ap);
 prn_string("\n");
}

/* funcao para permitir a visualizacao da informacao de um
AGV (utilizando o comando "examine Queue") */
void prn_AGV(agv)
 AGV *agv;
{
 char s[255];

 sprintf(s, "AGV %u, estado=", agv->id);
 prn_string(s);
 escr_str_est_movim_agv(s, agv->estado);
```

```
 prn_string(s);
 prn_string(", est_ant=");
 escr_str_est_movim_agv(s, agv->est_ant);
 prn_string(s);
 prn_string("\nposicao=");
 prn_ap_No_SP(agv->posicao);
 sprintf(s,
"(t_intr,t_man,t_erro,t_p_erro,t_f_mov)=\n(%.3f,%.3f,%.3f,%.3f,%.3f)\n",
    agv->t_intr, agv->t_manual, agv->t_erro,
    agv->t_perm_erro, agv->t_f_mov);
 prn_string(s);
}

/* funcao para permitir a visualizacao do conteudo de uma
mensagem SGAGV <-> SPAGV, a partir de uma posicao que contenha
um token do tipo MENS_SGAGV_SPAGV
(utilizando o comando "examine Queue") */
void prn_MENS_SPAGV_SGAGV(a)
  MENS_SGAGV_SPAGV *a;
{ prn_m_SGAGV_SPAGV(a->mens); }

/* funcao para permitir a visualizacao de um token do tipo
ULINT (utilizando o comando "examine Queue") */
void prn_ULINT_SP(a)
 ULINT *a;
{
  char s[255];

  sprintf(s, "(ulint) i=%u\n", a->i);
  prn_string(s);
}
```

C.4.7 Acções iniciais

```
/* Examine Queue em posicoes de tipos nao geridos automatica/ */
 XW_AP_LAYOUT_print = prn_AP_LAYOUT;
 XW_BOOLEANO_print = prn_BOOLEANO_SP;
 XW_AP_NOS_INTR_print = prn_AP_NOS_INTR;
 XW_AGV_print = prn_AGV;
 XW_MENS_SGAGV_SPAGV_print=prn_MENS_SPAGV_SGAGV;
 XW_ULINT_print=prn_ULINT_SP;

/* inicializa o estado inicial do objecto */
 xx_setstate(0);
```

C.4.8 Acções finais

```
/* destroi as listas alocadas dinamicamente */
 limpa_l_nos_SP(XX->layout);
 limpa_l_uid(XX->nos_intr);
```

C.4.9 Tipos de tokens

C.4.9.1 AP_LAYOUT

```
ap_No_SP ap;
```

C.4.9.2 AP_NOS_INTR

C.4.9.2.1 Definição

```
ap_no_l_uid ap;
```

C.4.9.2.2 Descrição

Tipo utilizado para apontar a cabeça de uma lista de nós de introdução, em que cada elemento contém os campos seguintes:

- i – identificador do nó do *layout*;
- d – probabilidade de ocorrência de uma introdução no nó.

C.4.9.3 AGV

C.4.9.3.1 Definição

```
ulint id;
ulint estado;
ulint est_ant;
ap_No_SP posicao;
double t_intr;
double t_manual;
double t_erro;
double t_perm_erro;
double t_f_mov;
```

C.4.9.3.2 Descrição

Tipo utilizado para representar um AGV. Os campos do tipo double permitem determinar a evolução do estado do AGV em função do tempo. Estes têm a seguinte funcionalidade:

- t_intr – instante em que o AGV poderá sair do estado ENABLED e ser introduzido;
- t_manual – instante em que o AGV poderá passar ao estado ENABLED, depois de ter sido introduzido;
- t_erro – instante em que o AGV poderá passar ao estado ERRO, depois de ter sido introduzido;
- t_perm_erro – instante em que o AGV com erro poderá sair desse estado;
- t_f_mov – instante em que o AGV em movimento poderá passar ao estado PARADO.

C.4.10 Variáveis locais

- layout:ap_No_SP – apontador para a cabeça da lista de nós do *layout* do SPAGV;

- nos_intr:ap_no_l_uid – apontador para a cabeça da lista em que cada elemento possui um identificador de um nó do *layout* e a respectiva probabilidade de ocorrência de uma introdução;
- buffer:xx_string – buffer utilizado para chamar a função 'prn_string()';

C.4.11 Posições partilhadas

- FIM_CONFIG_SPAGV:NUL;
- AGV_DISABLED:AGV;
- AGV_ENABLED:AGV;
- AGV_PARADO:AGV;
- AGV_C_ERRO:AGV;
- AGV_MOVIMENTO:AGV;
- TEMPO_INTR_AGV:TEMPORIZACAO;
- FIM_TEMP_INTR_AGV:ULINT;
- INTERR_TEMP_INTR_AGV:ULINT;
- TEMPO_ACT_MAN:TEMPORIZACAO;
- TEMPO_ACT_ERRO:TEMPORIZACAO;
- TEMPO_MOVIMENTO:TEMPORIZACAO;
- INTERR_TEMP_ACT_MANUAL:ULINT;
- INTERR_TEMP_ACT_ERRO:MANUAL:
- FIM_TEMP_ACT_MANUAL:ULINT;
- FIM_TEMP_ACT_ERRO:ULINT;
- FIM_TEMP_MOVIMENTO:ULINT;
- TEMPO_RESOL_ERRO:TEMPORIZACAO;
- FIM_TEMP_RESOL_ERRO:ULINT;
- INTERR_TEMP_MOVIM:ULINT;
- AGV_DESACT:AGV.

C.4.12 Entidades da RP da página principal

C.4.12.1 Objectos

- TEMP_ACT_ERRO:TEMPORIZADOR – temporizador utilizado para gerir as decisões relativas à colocação de AGVs activos (no estado PARADO ou em movimento) no estado ERRO;

- TEMP_ACT_MANUAL:TEMPORIZADOR – temporizador utilizado para gerir as decisões relativas à colocação de AGVs activos (no estado PARADO ou em movimento) em modo manual (estado ENABLED);
- TEMP_INTR_AGVS:TEMPORIZADOR – temporizador utilizado para gerir as decisões relativas à introdução de AGVs (passagem do modo manual ao modo automático);
- TEMP_MOVIM_AGVS:TEMPORIZADOR – temporizador utilizado para gerir os tempos de execução dos movimentos elementares de translacção dos AGVs;
- TEMP_RESOL_ERRO:TEMPORIZADOR – temporizador utilizado para gerir os tempos de resolução dos erros internos que ocorram nos AGVs.

C.4.12.2 Transições

C.4n1v.2.1 Acção de 'LIMPA_FIM_TEMP_ACT_ERRO'

```
xx_setstate(46);
```

C.4.12.2.1 Acção de 'LIMPA_FIM_TEMP_ACT_MANUAL'

```
xx_setstate(45);
```

C.4.12.2.1 Acção de 'LIMPA_FIM_TEMP_MOVIMENTO'

```
xx_setstate(47);
```

C.4.13 Entidades da RP da página 'cancela_movim_avanco'

C.4.13.1 Posição 'ID_AGV_MOV_AVAN_CANCEL:ULINT'

C.4.13.2 Transições

C.4.13.2.1 CANCELA_MOV_AVANCO

C.4.13.2.1.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_agv==ID_AGV_MOV_AVAN_CANCEL->i &&
 MENS_RECEB_SGAGV->mens.tipo_mens==MOVIMENTO
)
```

C.4.13.2.1.2 Acção

```
sprintf(XX->buffer,
"\nSPAGV: movimento em avanco do AGV %u cancelado, correspondente a
mensagem:\n",
ID_AGV_MOV_AVAN_CANCEL->i);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(-1.0, MENS_RECEB_SGAGV->mens);

xx_setstate(43);
```

C.4.13.2.2 Acção de 'NOTIF_SGAGV_CANCEL_MOV_AVAN'

```
MENS_ENV_SGAGV->mens.id_agv=ID_AGV_MOV_AVAN_CANCEL->i;
MENS_ENV_SGAGV->mens.tipo_mens=MOV_AVAN_CANCEL;

sprintf(XX->buffer,
"\nSPAGV: notifica cancelamento dos movimentos em avanco do AGV %u,\n
atraves do envio de:\n",
ID_AGV_MOV_AVAN_CANCEL->i);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_ENV_SGAGV->mens);

xx_setstate(44);
```

C.4.13.2.3 REC_CANCEL_MOV_AVAN

C.4.13.2.3.1 Predicado

```
(MENS_RECEB_SGAGV->mens.tipo_mens==CANCEL_MOV_AVAN)
```

C.4.13.2.3.2 Acção

```
sprintf(XX->buffer,
"\nSPAGV: inicio do cancelamento de movimentos em avanco do AGV %u,\n
apos a recepcao de:\n",
MENS_RECEB_SGAGV->mens.id_agv);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

ID_AGV_MOV_AVAN_CANCEL->i=MENS_RECEB_SGAGV->mens.id_agv;

xx_setstate(42);
```

C.4.14 Entidades da RP da página 'gera_AGVs_DISABLED'

C.4.14.1 Posição 'ID_AGV:ULINT'

C.4.14.2 Transições

C.4.14.2.1 GERA_AGV_DISABLED

C.4.14.2.1.1 Predicado

```
(ID_AGV->i<=XX->numero_agvs)
```

C.4.14.2.1.2 Acção

```
AGV_DISABLED->id=ID_AGV->i;
AGV_DISABLED->est_ant=DISABLED;
AGV_DISABLED->estado=DISABLED;
AGV_DISABLED->posicao=NULL;
AGV_DISABLED->t_intr=0.0;
AGV_DISABLED->t_manual=0.0;
AGV_DISABLED->t_erro=0.0;
```

```
AGV_DISABLED->t_perm_erro=0.0;
AGV_DISABLED->t_f_mov=0.0;

sprintf(XX->buffer,
    "SPAGV: AGV %u no estado DISABLED criado\n",
    AGV_DISABLED->id);
prn_string(XX->buffer);

(ID_AGV->i)++;

xx_setstate(6);
```

C.4.14.2.2 Acção de 'INIC_ID_AGV'

```
ID_AGV->i=1;

xx_setstate(5);
```

C.4.14.2.3 Acção de 'LIMPA_ID_AGV'

```
xx_setstate(7);
```

C.4.15 Transições da RP da página 'gere_AGVs_DISABLED'

C.4.15.1 COLOCA_AGV_ENABLED

C.4.15.1.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_agv==AGV_DISABLED->id &&
 MENS_RECEB_SGAGV->mens.tipo_mens==INF_EST_AGV &&
 MENS_RECEB_SGAGV->mens.p1==ENABLED
)
```

C.4.15.1.2 Acção

```
double dt=rnd_exp_neg(XX->t_med_introducao);
double t=xx_gettime();

AGV_ENABLED->t_intr=t+dt;
AGV_ENABLED->est_ant=DISABLED;
AGV_ENABLED->estado=ENABLED;
TEMPO_INTR_AGV->id=AGV_DISABLED->id;
TEMPO_INTR_AGV->t=dt;

sprintf(XX->buffer,
    "\nSPAGV: AGV %u DISABLED passa a estar ENABLED apos a recepcao
de:\n",
     AGV_DISABLED->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(t, MENS_RECEB_SGAGV->mens);

xx_setstate(8);
```

C.4.15.2 IGN_MENS_MOV_AGV_DIS

C.4.15.2.1 Predicado

```
(AGV_DISABLED->id==MENS_RECEB_SGAGV->mens.id_agv &&
 MENS_RECEB_SGAGV->mens.tipo_mens==MOVIMENTO
)
```

C.4.15.2.2 Acção

```
sprintf(XX->buffer,
    "\nSPAGV: mensagem para o AGV DISABLED %u ignorada:\n",
    AGV_DISABLED->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

xx_setstate(9);
```

C.4.16 Entidades da RP da página 'gere_AGVs_ENABLED'

C.4.16.1 Posição 'AGV_ESPERA_CONF_INTR:AGV'

C.4.16.2 Transições

C.4.16.2.1 ACTIVA_AGV

C.4.16.2.1.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_agv==AGV_ESPERA_CONF_INTR->id &&
 MENS_RECEB_SGAGV->mens.tipo_mens==ESTADO_INTR &&
 MENS_RECEB_SGAGV->mens.p1==AGV_INTROD
)
```

C.4.16.2.1.2 Acção

```
double dt_manual=rnd_exp_neg(XX->t_med_manual);
double dt_erro=rnd_exp_neg(XX->t_med_erro);
double t=xx_gettime();

AGV_PARADO->est_ant=ENABLED;
AGV_PARADO->estado=PARADO;
AGV_PARADO->t_manual=t+dt_manual;
AGV_PARADO->t_erro=t+dt_erro;

TEMPO_ACT_MAN->id=AGV_PARADO->id;
TEMPO_ACT_MAN->t=dt_manual;
TEMPO_ACT_ERRO->id=AGV_PARADO->id;
TEMPO_ACT_ERRO->t=dt_erro;

sprintf(XX->buffer,
    "\nSPAGV: AGV %u ENABLED passa a estar PARADO apos a recepcao
de:\n",
     AGV_ESPERA_CONF_INTR->id);
```

```
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(t, MENS_RECEB_SGAGV->mens);

xx_setstate(14);
```

C.4.16.2.2 DISABLE_AGV_ENABLED

C.4.16.2.2.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_agv==AGV_ENABLED->id &&
 MENS_RECEB_SGAGV->mens.tipo_mens==INF_EST_AGV &&
 MENS_RECEB_SGAGV->mens.p1==DISABLED
)
```

C.4.16.2.2.2 Acção

```
AGV_DISABLED->est_ant=ENABLED;
AGV_DISABLED->estado=DISABLED;

INTERR_TEMP_INTR_AGV->i=AGV_ENABLED->id;

sprintf(XX->buffer,
    "\nSPAGV: AGV %u ENABLED passa a estar DISABLED apos a recepcao
de:\n",
     AGV_ENABLED->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

xx_setstate(12);
```

C.4.16.2.3 DISABLE_AGV_ESP_CONF_INTR

C.4.16.2.3.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_agv==AGV_ESPERA_CONF_INTR->id &&
 MENS_RECEB_SGAGV->mens.tipo_mens==INF_EST_AGV &&
 MENS_RECEB_SGAGV->mens.p1==DISABLED
)
```

C.4.16.2.3.2 Acção

```
AGV_DISABLED->est_ant=ENABLED;
AGV_DISABLED->estado=DISABLED;
AGV_DISABLED->posicao=NULL;

sprintf(XX->buffer,
    "\nSPAGV: AGV %u a espera de confirm. de introducao\npassa a estar
DISABLED apos a recepcao de:\n",
     AGV_ESPERA_CONF_INTR->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

xx_setstate(13);
```

C.4.16.2.4 IGN_MENS_MOV_AGV_EN

C.4.16.2.4.1 Predicado

```
(AGV_ENABLED->id==MENS_RECEB_SGAGV->mens.id_agv &&
 MENS_RECEB_SGAGV->mens.tipo_mens==MOVIMENTO
)
```

C.4.16.2.4.2 Acção

```
sprintf(XX->buffer,
    "\nSPAGV: mensagem para o AGV ENABLED %u ignorada:\n",
    AGV_ENABLED->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

xx_setstate(15);
```

C.4.16.2.5 INTRODUZ_AGV

C.4.16.2.5.1 Predicado

```
(AGV_ENABLED->id==FIM_TEMP_INTR_AGV->i)
```

C.4.16.2.5.2 Acção

```
MENS_ENV_SGAGV->mens.id_agv=AGV_ENABLED->id;
MENS_ENV_SGAGV->mens.tipo_mens=INTRODUCAO;
MENS_ENV_SGAGV->mens.p1=decide_no_intr(XX->nos_intr);

AGV_ESPERA_CONF_INTR->posicao=
    get_no_lay_SP(MENS_ENV_SGAGV->mens.p1,
            XX->layout);
if (XX->poss_acesso==0 ||
    AGV_ESPERA_CONF_INTR->posicao->tipo_no==TRANSITO)
 if (rnd_binomial(XX->p_introd_or_pos))
    MENS_ENV_SGAGV->mens.p2=1;
  else MENS_ENV_SGAGV->mens.p2=-1;
else MENS_ENV_SGAGV->mens.p2=XX->poss_acesso*
    AGV_ESPERA_CONF_INTR->posicao->pos_rel_mesa;

sprintf(XX->buffer,
    "\nSPAGV: Notifica introd. AGV %u, enviando a mensagem:\n",
    AGV_ENABLED->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_ENV_SGAGV->mens);

xx_setstate(10);
```

C.4.16.2.6 RECEBE_ERRO_INTR_AGV

C.4.16.2.6.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_agv==AGV_ESPERA_CONF_INTR->id &&
 MENS_RECEB_SGAGV->mens.tipo_mens==ESTADO_INTR &&
 MENS_RECEB_SGAGV->mens.p1!=AGV_INTROD
)
```

C.4.16.2.6.2 Acção

```
double dt=rnd_exp_neg(XX->t_med_introducao);
double t=xx_gettime();

AGV_ENABLED->posicao=NULL;
AGV_ENABLED->t_intr=t+dt;

TEMPO_INTR_AGV->id=AGV_ESPERA_CONF_INTR->id;
TEMPO_INTR_AGV->t=dt;

sprintf(XX->buffer,
    "\nSPAGV: AGV %u a espera da conf. da sua introducao mantem-se
ENABLED, apos a recepcao de:\n",
     AGV_ESPERA_CONF_INTR->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(t, MENS_RECEB_SGAGV->mens);

xx_setstate(11);
```

C.4.17 Entidades da RP da página 'gere_AGVs_ERRO'

C.4.17.1 Posições

- AGV_RESOLV_ERRO:AGV;
- DEmISAO_AGV_ERRO_MANUAL:AGV.

C.4.17.2 Transições

C.4.17.2.1 Acção de 'AGV_CONTINUA_ERRO'

```
TEMPO_RESOL_ERRO->id=DECISAO_AGV_ERRO_MANUAL->id;
TEMPO_RESOL_ERRO->t=DECISAO_AGV_ERRO_MANUAL->t_perm_erro-
   xx_gettime();

xx_setstate(32);
```

C.4.17.2.2 AGV_ERRO_MANUAL

C.4.17.2.2.1 Predicado

```
(DECISAO_AGV_ERRO_MANUAL->t_perm_erro==0.0)
```

C.4.17.2.2.2 Acção

```
double dt=rnd_exp_neg(XX->t_med_introducao);
double t=xx_gettime();

MENS_ENV_SGAGV->mens.id_agv=DECISAO_AGV_ERRO_MANUAL->id;
MENS_ENV_SGAGV->mens.tipo_mens=AGV_MANUAL;

AGV_DESACT->estado=ENABLED;
AGV_DESACT->t_intr=t+dt;
```

```
TEMPO_INTR_AGV->id=DECISAO_AGV_ERRO_MANUAL->id;
TEMPO_INTR_AGV->t=dt;

sprintf(XX->buffer,
"\nSPAGV: Notifica colocacao em manual do AGV %u c/ ERRO, enviando a
mensagem:\n",
DECISAO_AGV_ERRO_MANUAL->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(t, MENS_ENV_SGAGV->mens);

xx_setstate(33);
```

C.4.17.2.3 AGV_ERRO_MOVIM_TRANS

C.4.17.2.3.1 Predicado

```
(AGV_RESOLV_ERRO->est_ant==EX_MOVIM &&
AGV_RESOLV_ERRO->id==FIM_TEMP_RESOL_ERRO->i)
```

C.4.17.2.3.2 Acção

```
double dt_manual=rnd_exp_neg(XX->t_med_manual);
double dt_erro=rnd_exp_neg(XX->t_med_erro);
double t=xx_gettime();

MENS_ENV_SGAGV->mens.id_agv=AGV_RESOLV_ERRO->id;
MENS_ENV_SGAGV->mens.tipo_mens=FIM_ERRO_AGV;

AGV_MOVIMENTO->estado=EX_MOVIM;
AGV_MOVIMENTO->est_ant=ERRO;
AGV_MOVIMENTO->t_f_mov=t+AGV_RESOLV_ERRO->t_f_mov-
   AGV_RESOLV_ERRO->t_erro;
AGV_MOVIMENTO->t_manual=t+dt_manual;
AGV_MOVIMENTO->t_erro=t+dt_erro;

TEMPO_MOVIMENTO->id=AGV_RESOLV_ERRO->id;
TEMPO_MOVIMENTO->t=AGV_RESOLV_ERRO->t_f_mov-t;
TEMPO_ACT_MAN->id=AGV_RESOLV_ERRO->id;
TEMPO_ACT_MAN->t=t+dt_manual;
TEMPO_ACT_ERRO->id=AGV_RESOLV_ERRO->id;
TEMPO_ACT_ERRO->t=dt_erro;

sprintf(XX->buffer,
"\nSPAGV: Notifica a resolucao do erro no AGV em MOVIMENTO %u, enviando
a mensagem:\n",
AGV_RESOLV_ERRO->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(t, MENS_ENV_SGAGV->mens);

xx_setstate(36);
```

C.4.17.2.4 AGV_ERRO_OP_TRCAR

C.4.17.2.4.1 Predicado

```
((AGV_RESOLV_ERRO->est_ant==CARGA_AGV ||
 AGV_RESOLV_ERRO->est_ant==DESCARGA_AGV) &&
AGV_RESOLV_ERRO->id==FIM_TEMP_RESOL_ERRO->i)
```

C.4.17.2.4.2 Acção

```
double dt_manual=rnd_exp_neg(XX->t_med_manual);
double dt_erro=rnd_exp_neg(XX->t_med_erro);
double t=xx_gettime();

MENS_ENV_SGAGV->mens.id_agv=AGV_RESOLV_ERRO->id;
MENS_ENV_SGAGV->mens.tipo_mens=FIM_ERRO_AGV;

AGV_MOVIMENTO->estado=AGV_RESOLV_ERRO->est_ant;
AGV_MOVIMENTO->est_ant=ERRO;
AGV_MOVIMENTO->t_manual=t+dt_manual;
AGV_MOVIMENTO->t_erro=t+dt_erro;

OP_TRANSF_CARGA_RETOM->i=AGV_RESOLV_ERRO->posicao->id_no;

TEMPO_ACT_MAN->id=AGV_RESOLV_ERRO->id;
TEMPO_ACT_MAN->t=t+dt_manual;
TEMPO_ACT_ERRO->id=AGV_RESOLV_ERRO->id;
TEMPO_ACT_ERRO->t=dt_erro;

sprintf(XX->buffer,
"\nSPAGV: Notifica a resolucao do erro no AGV %u,\n a executar uma op.
de transf. de carga, enviando a mensagem:\n",
AGV_RESOLV_ERRO->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(t, MENS_ENV_SGAGV->mens);

xx_setstate(37);
```

C.4.17.2.5 AGV_ERRO_PARADO

C.4.17.2.5.1 Predicado

```
(AGV_RESOLV_ERRO->id==FIM_TEMP_RESOL_ERRO->i)
```

C.4.17.2.5.2 Acção

```
double dt_manual=rnd_exp_neg(XX->t_med_manual);
double dt_erro=rnd_exp_neg(XX->t_med_erro);
double t=xx_gettime();

MENS_ENV_SGAGV->mens.id_agv=AGV_RESOLV_ERRO->id;
MENS_ENV_SGAGV->mens.tipo_mens=FIM_ERRO_AGV;

AGV_PARADO->est_ant=ERRO;
AGV_PARADO->estado=PARADO;
AGV_PARADO->t_manual=t+dt_manual;
AGV_PARADO->t_erro=t+dt_erro;

TEMPO_ACT_MAN->id=AGV_RESOLV_ERRO->id;
TEMPO_ACT_MAN->t=dt_manual;
TEMPO_ACT_ERRO->id=AGV_RESOLV_ERRO->id;
TEMPO_ACT_ERRO->t=dt_erro;

sprintf(XX->buffer,
"\nSPAGV: Notifica a resolucao do erro no AGV PARADO %u, enviando a
mensagem:\n",
AGV_RESOLV_ERRO->id);
```

```
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(t, MENS_ENV_SGAGV->mens);

xx_setstate(35);
```

C.4.17.2.6 Acção de 'DECIDE_AGV_ERRO_MANUAL'

```
int decisao=rnd_binomial(XX->p_erro_manual);

sprintf(XX->buffer,
"\nSPAGV: decide se o AGV %u passa a manual ou continua no estado
ERRO...\n",
AGV_C_ERRO->id);
prn_string(XX->buffer);

if (decisao)
{
 DECISAO_AGV_ERRO_MANUAL->t_perm_erro=0.0;
 prn_string("\tvai passar para manual\n");
}
else
{
 DECISAO_AGV_ERRO_MANUAL->t_perm_erro=
    xx_gettime()+rnd_exp_neg(XX->t_med_perm_erro);
 prn_string("\tvai continuar no estado ERRO\n");
}

xx_setstate(31);
```

C.4.17.2.7 DISABLE_AGV_C_ERRO

C.4.17.2.7.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_agv==AGV_RESOLV_ERRO->id &&
 MENS_RECEB_SGAGV->mens.tipo_mens==INF_EST_AGV &&
 MENS_RECEB_SGAGV->mens.p1==DISABLED
)
```

C.4.17.2.7.2 Acção

```
AGV_DESACT->estado=DISABLED;
AGV_DESACT->posicao=NULL;

INTERR_TEMP_RESOL_ERRO->i=AGV_RESOLV_ERRO->id;

sprintf(XX->buffer,
"\nSPAGV: AGV %u no estado ERRO passa a estar DISABLED apos a recepcao
de:\n",
AGV_RESOLV_ERRO->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

xx_setstate(34);
```

C.4.18 Entidades da RP da página 'gere_AGVs_MOVIMENTO'

C.4.18.1 Posição 'AGV_MOV_ERRO:AGV'

C.4.18.2 Transições

C.4.18.2.1 AGV_MOVIM_ERRO

C.4.18.2.1.1 Predicado

```
(AGV_MOVIMENTO->id==FIM_TEMP_ACT_ERRO->i)
```

C.4.18.2.1.2 Acção

```
AGV_MOV_ERRO->est_ant=AGV_MOVIMENTO->estado;
AGV_MOV_ERRO->estado=ERRO;

MENS_ENV_SGAGV->mens.id_agv=AGV_MOVIMENTO->id;
MENS_ENV_SGAGV->mens.tipo_mens=ERRO_AGV;

INTERR_TEMP_MOVIM->i=AGV_MOVIMENTO->id;
INTERR_TEMP_ACT_MANUAL->i=AGV_MOVIMENTO->id;

sprintf(XX->buffer,
    "\nSPAGV: Notifica a ocorrencia de um erro no AGV em MOVIMENTO %u,
enviando a mensagem:\n",
    AGV_MOVIMENTO->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_ENV_SGAGV->mens);

xx_setstate(28);
```

C.4.18.2.2 AGV_MOVIM_MANUAL

C.4.18.2.2.1 Predicado

```
(AGV_MOVIMENTO->id==FIM_TEMP_ACT_MANUAL->i)
```

C.4.18.2.2.2 Acção

```
double dt=rnd_exp_neg(XX->t_med_introducao);
double t=xx_gettime();

MENS_ENV_SGAGV->mens.id_agv=AGV_MOVIMENTO->id;
MENS_ENV_SGAGV->mens.tipo_mens=AGV_MANUAL;

AGV_DESACT->est_ant=AGV_DESACT->estado;
AGV_DESACT->estado=ENABLED;
AGV_DESACT->t_intr=t+dt;

TEMPO_INTR_AGV->id=AGV_MOVIMENTO->id;
TEMPO_INTR_AGV->t=dt;
INTERR_TEMP_MOVIM->i=AGV_MOVIMENTO->id;
INTERR_TEMP_ACT_ERRO->i=AGV_MOVIMENTO->id;
```

```
 sprintf(XX->buffer,
"\nSPAGV: Notifica colocacao em manual do AGV %u em MOVIMENTO, enviando
a mensagem:\n",
AGV_MOVIMENTO->id);
 prn_string(XX->buffer);
 prn_m_SGAGV_SPAGV(t, MENS_ENV_SGAGV->mens);

 xx_setstate(27);
```

C.4.18.2.3 DISABLE_AGV_MOVIMENTO

C.4.18.2.3.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_agv==AGV_MOVIMENTO->id &&
 MENS_RECEB_SGAGV->mens.tipo_mens==INF_EST_AGV &&
 MENS_RECEB_SGAGV->mens.pl==DISABLED
)
```

C.4.18.2.3.2 Acção

```
AGV_DESACT->est_ant=AGV_MOVIMENTO->estado;
AGV_DESACT->estado=DISABLED;

INTERR_TEMP_MOVIM->i=AGV_MOVIMENTO->id;
INTERR_TEMP_ACT_MANUAL->i=AGV_MOVIMENTO->id;
INTERR_TEMP_ACT_ERRO->i=AGV_MOVIMENTO->id;

 sprintf(XX->buffer,
"\nSPAGV: AGV %u em MOVIMENTO passa a estar DISABLED apos a recepcao
de:\n",
AGV_MOVIMENTO->id);
 prn_string(XX->buffer);
 prn_m_SGAGV_SPAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

 xx_setstate(26);
```

C.4.18.2.4 INTERR_OP_TRCAR

C.4.18.2.4.1 Predicado

```
(AGV_MOV_ERRO->est_ant!=EX_MOVIM)
```

C.4.18.2.4.2 Acção

```
OP_TRANSF_CARGA_INTERR->i=AGV_MOV_ERRO->posicao->id_no;

xx_setstate(29);
```

C.4.18.2.5 Acção de 'LIMPA_AGV_MOV_ERRO'

```
xx_setstate(30);
```

C.4.18.2.6 PARA_AGV

C.4.18.2.6.1 Predicado

```
(AGV_MOVIMENTO->id==FIM_TEMP_MOVIMENTO->i)
```

C.4.18.2.6.2 Acção

```
MENS_ENV_SGAGV->mens.id_agv=AGV_MOVIMENTO->id;
MENS_ENV_SGAGV->mens.tipo_mens=FIM_MOVIMENTO;

AGV_PARADO->est_ant=EX_MOVIM;
AGV_PARADO->estado=PARADO;

sprintf(XX->buffer,
"\nSPAGV: Notifica o fim do movimento do AGV %u, enviando a
mensagem:\n",
AGV_MOVIMENTO->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_ENV_SGAGV->mens);

xx_setstate(23);
```

C.4.18.2.7 TERMINA_OP_CARGA

C.4.18.2.7.1 Predicado

```
(AGV_MOVIMENTO->posicao->id_no==FIM_OPERACAO_CARGA->i)
```

C.4.18.2.7.2 Acção

```
MENS_ENV_SGAGV->mens.id_agv=AGV_MOVIMENTO->id;
MENS_ENV_SGAGV->mens.tipo_mens=FIM_MOVIMENTO;

AGV_PARADO->est_ant=CARGA_AGV;
AGV_PARADO->estado=PARADO;

sprintf(XX->buffer,
"\nSPAGV: Notifica o fim do movimento de carga do AGV %u, enviando a
mensagem:\n",
AGV_MOVIMENTO->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_ENV_SGAGV->mens);

xx_setstate(24);
```

C.4.18.2.8 TERMINA_OP_DESCARGA

C.4.18.2.8.1 Predicado

```
(AGV_MOVIMENTO->posicao->id_no==FIM_OPERACAO_DESCARGA->i)
```

C.4.18.2.8.2 Acção

```
MENS_ENV_SGAGV->mens.id_agv=AGV_MOVIMENTO->id;
MENS_ENV_SGAGV->mens.tipo_mens=FIM_MOVIMENTO;

AGV_PARADO->est_ant=DESCARGA_AGV;
AGV_PARADO->estado=PARADO;

sprintf(XX->buffer,
"\nSPAGV: Notifica o fim do movimento de descarga do AGV %u, enviando a
mensagem:\n",
AGV_MOVIMENTO->id);
```

```
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_ENV_SGAGV->mens);

xx_setstate(25);
```

C.4.19 Entidades de 'gere_AGVs_PARADOS'

C.4.19.1 Posição 'AGV_TRANSF_CARGA:AGV'

C.4.19.2 Transições

C.4.19.2.1 AGV_PARADO_ERRO

C.4.19.2.2 Predicado

```
(AGV_PARADO->id==FIM_TEMP_ACT_ERRO->i)
```

C.4.19.2.3 Acção

```
AGV_C_ERRO->est_ant=PARADO;
AGV_C_ERRO->estado=ERRO;

MENS_ENV_SGAGV->mens.id_agv=AGV_PARADO->id;
MENS_ENV_SGAGV->mens.tipo_mens=ERRO_AGV;

INTERR_TEMP_ACT_MANUAL->i=AGV_PARADO->id;

sprintf(XX->buffer,
"\nSPAGV: Notifica a ocorrencia de um erro no AGV PARADO %u, enviando a
mensagem:\n",
AGV_PARADO->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_ENV_SGAGV->mens);

xx_setstate(22);
```

C.4.19.2.2 AGV_PARADO_MANUAL

C.4.19.2.2.1 Predicado

```
(AGV_PARADO->id==FIM_TEMP_ACT_MANUAL->i)
```

C.4.19.2.2.2 Acção

```
double dt=rnd_exp_neg(XX->t_med_introducao);
double t=xx_gettime();

MENS_ENV_SGAGV->mens.id_agv=AGV_PARADO->id;
MENS_ENV_SGAGV->mens.tipo_mens=AGV_MANUAL;

AGV_ENABLED->est_ant=PARADO;
AGV_ENABLED->estado=ENABLED;
AGV_ENABLED->posicao=NULL;
AGV_ENABLED->t_intr=t+dt;
```

```
TEMPO_INTR_AGV->id=AGV_PARADO->id;
TEMPO_INTR_AGV->t=dt;
INTERR_TEMP_ACT_ERRO->i=AGV_PARADO->id;

sprintf(XX->buffer,
"\nSPAGV: Notifica colocacao em manual do AGV %u PARADO, enviando a
mensagem:\n",
AGV_PARADO->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(t, MENS_ENV_SGAGV->mens);

xx_setstate(21);
```

C.4.19.2.3 DISABLE_AGV_PARADO

C.4.19.2.3.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_agv==AGV_PARADO->id &&
 MENS_RECEB_SGAGV->mens.tipo_mens==INF_EST_AGV &&
 MENS_RECEB_SGAGV->mens.p1==DISABLED
)
```

C.4.19.2.3.2 Acção

```
AGV_DISABLED->est_ant=PARADO;
AGV_DISABLED->estado=DISABLED;
AGV_DISABLED->posicao=NULL;

INTERR_TEMP_ACT_MANUAL->i=AGV_PARADO->id;
INTERR_TEMP_ACT_ERRO->i=AGV_PARADO->id;

sprintf(XX->buffer,
"\nSPAGV: AGV %u PARADO passa a estar DISABLED apos a recepcao de:\n",
AGV_PARADO->id);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

xx_setstate(20);
```

C.4.19.2.4 INICIA_OP_CARGA

C.4.19.2.4.1 Predicado

```
(AGV_TRANSF_CARGA->estado==CARGA_AGV)
```

C.4.19.2.4.2 Acção

```
INICIO_OP_CARGA->i=AGV_TRANSF_CARGA->posicao->id_no;

xx_setstate(18);
```

C.4.19.2.5 Acção de 'INICIA_OP_DESCARGA'

```
INICIO_OP_DESCARGA->i=AGV_TRANSF_CARGA->posicao->id_no;

xx_setstate(19);
```

C.4.19.2.6 RECEBE_ORDEM_MOV_TRANSL

C.4.19.2.6.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_agv==AGV_PARADO->id &&
 MENS_RECEB_SGAGV->mens.tipo_mens==MOVIMENTO &&
 MENS_RECEB_SGAGV->mens.p1==MOV_TRANSL
)
```

C.4.19.2.6.2 Acção

```
double t=xx_gettime();
double t_mov;

sprintf(XX->buffer,
"\nSPAGV: recebe uma ordem de movimento de translaccao para o AGV %u,
atraves da mensagem:\n",
MENS_RECEB_SGAGV->mens.id_agv);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(t, MENS_RECEB_SGAGV->mens);

t_mov=processa_mov_transl(MENS_RECEB_SGAGV->mens, &AGV_PARADO-
>posicao);

AGV_MOVIMENTO->est_ant=PARADO;
AGV_MOVIMENTO->estado=EX_MOVIM;
AGV_MOVIMENTO->t_f_mov=t+t_mov;

TEMPO_MOVIMENTO->id=AGV_PARADO->id;
TEMPO_MOVIMENTO->t=t_mov;

xx_setstate(16);
```

C.4.19.2.7 RECEBE_ORDEM_MOV_TR_CARGA

C.4.19.2.7.1 Predicado

```
(MENS_RECEB_SGAGV->mens.id_agv==AGV_PARADO->id &&
 MENS_RECEB_SGAGV->mens.tipo_mens==MOVIMENTO &&
 MENS_RECEB_SGAGV->mens.p1!=MOV_TRANSL
)
```

C.4.19.2.7.2 Acção

```
sprintf(XX->buffer,
"\nSPAGV: recebe uma ordem de movimento de transf. de carga para o AGV
%u, atraves da mensagem:\n",
MENS_RECEB_SGAGV->mens.id_agv);
prn_string(XX->buffer);
prn_m_SGAGV_SPAGV(xx_gettime(), MENS_RECEB_SGAGV->mens);

processa_mov_car_des(MENS_RECEB_SGAGV->mens, AGV_PARADO->posicao);

AGV_TRANSF_CARGA->est_ant=PARADO;
if (MENS_RECEB_SGAGV->mens.p1==MOV_CARGA)
   AGV_TRANSF_CARGA->estado=CARGA_AGV;
else AGV_TRANSF_CARGA->estado=DESCARGA_AGV;
AGV_TRANSF_CARGA->t_f_mov=0.0;
```

```
xx_setstate(17);
```

C.4.20 Entidades da RP da página 'leitura_fich_config'

C.4.20.1 Posições

- FIM_LEITURA_FICH_LAYOUT:NUL;
- FIM_LEITURA_FICH_NOS_INTR:NUL;
- INICIO_CONFIG_SPAGV:NUL;
- LAYOUT:AP_LAYOUT;
- NOS_INTR:AP_NOS_INTR;
- RES_VER_SOM_PROB_NOS_INTR:BOOLEANO.

C.4.20.2 Transições

C.4.20.2.1 Acção de 'LE_FICH_LAYOUT'

```
XX->layout=le_layout_SP(XX->nome_fich_layout,
    XX->nome_fich_mesas,
    XX->vel_frente, XX->vel_curva, XX->vel_tras);
LAYOUT->ap=XX->layout;

xx_setstate(1);
```

C.4.20.2.2 LE_FICH_NOS_INTRODUCAO

C.4.20.2.2.1 Predicado

```
(XX->layout!=NULL)
```

C.4.20.2.2.2 Acção

```
XX->nos_intr=le_f_nos_introd(XX->nome_fich_nos_intr);
NOS_INTR->ap=XX->nos_intr;

xx_setstate(2);
```

C.4.20.2.3 TERMINA_CONFIG_SPAGV

C.4.20.2.3.1 Predicado

```
(RES_VER_SOM_PROB_NOS_INTR->b!=FALSO)
```

C.4.20.2.3.2 Acção

```
xx_setstate(4);
```

C.4.20.2.4 VERIF_SOMAT_PROB_NOS_INTR

C.4.20.2.4.1 Predicado

```
(XX->nos_intr!=NULL)
```

C.4.20.2.4.2 Acção

```
RES_VER_SOM_PROB_NOS_INTR->b=som_prob_unit(XX->nos_intr);

xx_setstate(3);
```

C.4.21 Entidades da RP da página 'processa_AGVs_desactivados'

C.4.21.1 Posição 'AGV_DISABLED_OU_ENABLED:AGV'

C.4.21.2 Transições

C.4.21.2.1 Acção de 'AGV_DESACT_DISABLED'

```
AGV_DISABLED->posicao=NULL;

xx_setstate(41);
```

C.4.21.2.2 AGV_DESACT_ENABLED

C.4.21.2.2.1 Predicado

```
(AGV_DISABLED_OU_ENABLED->estado==ENABLED)
```

C.4.21.2.2.2 Acção

```
AGV_ENABLED->posicao=NULL;

xx_setstate(40);
```

C.4.21.2.3 CANCELA_OP_TRCAR

C.4.21.2.3.1 Predicado

```
(AGV_DESACT->est_ant==CARGA_AGV ||
AGV_DESACT->est_ant==DESCARGA_AGV)
```

C.4.21.2.3.2 Acção

```
OP_TRANSF_CARGA_CANCEL->i=AGV_DESACT->posicao->id_no;

xx_setstate(38);
```

C.4.21.2.4 Acção de 'LIMPA_AGV_DESACT'

```
xx_setstate(39);
```

Anexo D – Classes Artifex do Sistema de Fabrico

D.1 Classe CENTRO_RECEPCAO

D.1.1 CMN units incluídas

- ULINT;
- PALETE;
- TEMPORIZACAO;
- CAPAC_FILA.

D.1.2 Posições de entrada

- INICIO_OPERACAO_CARGA:ULINT;
- ERRO_TEMP_AGV:ULINT;
- ERRO_PERM_AGV:ULINT;
- ERRO_TEMP_RESOLV:ULINT.

D.1.3 Posições de saída

- PALETE_FILA_SAIDA:PALETE;
- FIM_OPERACAO_CARGA:ULINT;
- PONTOS_SAIDA_DISPONIVEIS:ULINT;
- PAL_ACABA_SAIR_ARMAZEM:PALETE;
- FILA_PALETE_INSERIDA:ULINT;
- CAPACIDADE_FILA:CAPAC_FILA.

D.1.4 Parâmetros

- nome:xx_string – nome identificador do objecto da classe instanciado;
- fila1:unsigned long int – identificador da 1ª fila de saída;
- filan:unsigned long int – identificador da n-ésima fila de saída;
- capacidade_filas:unsigned long int – capacidade máxima das filas de saída (dos *buffers* de saída);

- tm_recep_pal:double – valor médio do tempo associado à colocação de uma palete de um lote recebido, numa das filas de saída; admite-se que este tempo segue uma FDP normal;
- dp_trecep_pal:double – desvio padrão do tempo associado à colocação de uma palete proveniente de um lote recebido, numa das filas de saída; admite-se que este tempo segue uma FDP normal;
- vm_tcarga:double – valor médio do tempo associado à carga de uma palete de uma fila de saída num AGV; admite-se que este tempo segue uma FDP normal;
- dp_tcarga:double – desvio padrão do tempo associado à carga de uma palete de uma fila de saída num AGV;
- tm_entre_lotes:double – valor médio do tempo que medeia entre a recepção de dois lotes consecutivos; admite-se que este tempo segue uma FDP exponencial;
- tam_lotes_paletes:double – número de paletes existentes num lote recebido;
- num_recursos:double – número de recursos existentes para o transporte de paletes até às filas de saída do centro de recepção (operários ou empilhadores, etc).

D.1.5 Ficheiros de cabeçalho incluídos

```
#include "listas.h"
#include "mens_log.h"
#include "rand_fun.h"
```

D.1.6 Funções definidas internamente

```
/* funcao para permitir a visualizacao de um token do tipo
ULINT (utilizando o comando "examine Queue") */
void prn_ULINT_CENTRO_RECEPCAO(a)
 ULINT *a;
{
  char s[255];

  sprintf(s, "(ulint) i=%u\n", a->i);
  prn_string(s);
}
```

D.1.7 Acções iniciais

```
ap_no_l_int u;
ap_no_l_ulint t;
unsigned long int i;

/* inicializa variaveis utilizadas para processamento de estatisticas */

sprintf(XX->s, "num_paletes_exist_%s", XX->nome);
XX->ve_num_paletes_exist=xx_initmeasure(XX->s);
xx_writesample(XX->ve_num_paletes_exist, 0);



/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o estado das filas e
uma lista com o numero de paletes existentes em cada fila */
```

```
XX->l_ve_estado_filas=NULL;
u=NULL;
XX->p_num_paletes_filas=NULL;
t=NULL;
for (i=XX->fila1;(i<=XX->filan || i==XX->fila1);i++)
{
  sprintf(XX->s, "estado_fila_%u", i);
  insere_int(xx_initmeasure(XX->s), &XX->l_ve_estado_filas, &u);
  xx_writesample(u->i, 0);
  insere_ulint(0, &XX->p_num_paletes_filas, &t, 1);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o numero de
paletes retiradas das filas e uma lista com o numero de
paletes retiradas de cada fila */
XX->l_ve_paletes_retiradas_filas=NULL;
u=NULL;
XX->p_num_paletes_retiradas_filas=NULL;
t=NULL;
for (i=XX->fila1;(i<=XX->filan || i==XX->fila1);i++)
{
  sprintf(XX->s, "paletes_retiradas_fila_%u", i);
  insere_int(xx_initmeasure(XX->s), &XX->l_ve_paletes_retiradas_filas,
&u);
  xx_writesample(u->i, 0);
  insere_ulint(0, &XX->p_num_paletes_retiradas_filas, &t, 1);
}


/* Examine Queue em posicoes de tipos nao geridos automatica/ */

 XW_ULINT_print = prn_ULINT_CENTRO_RECEPCAO;


/* inicializa o estado do objecto */
/* (indica o WIP do Centro de Recepcao */

xx_setstate(0);
```

D.1.8 Acções finais

```
/* destroi as listas de identificadores de variaveis estatisticas */

limpa_l_int(&XX->l_ve_estado_filas);
limpa_l_int(&XX->l_ve_paletes_retiradas_filas);


/* destroi as listas com os contadores de paletes das filas */

limpa_l_ulint(XX->p_num_paletes_filas);
limpa_l_ulint(XX->p_num_paletes_retiradas_filas);
```

D.1.9 Variáveis locais

- l_ve_estado_filas:ap_no_l_int – lista de identificadores das variáveis estatísticas relativas ao estado de cada fila;

- ve_num_paletes_exist:int – identificador da variável estatística "número de paletes existentes";
- l_ve_paletes_retiradas_filas:ap_no_l_int – lista de identificadores das variáveis estatísticas relativas ao número de paletes retiradas de cada fila;
- p_num_paletes_filas:ap_no_l_ulint – apontador para a cabeça da lista que guarda o número de paletes existentes em cada fila;
- p_num_paletes_retiradas_filas:ap_no_l_int – apontador para a cabeça da lista que guarda o número de paletes retiradas de cada fila;
- s:xx_string – buffer de caracteres utilizado para chamar a função 'prn_string()';
- t:double – variável utilizada para guardar o tempo de simulação.

D.1.10 Posições partilhadas

- POSIÇÕES_LIVRES_FILAS:ULINT;
- ESPERA_CHEGADA_NOVO_LOTE:NUL;
- RECURSOS_LIVRES:ULINT.

D.1.11 Entidades da RP da página principal

D.1.11.1 Objectos

- TEMP_CARGA_AGV:TEMPORIZADOR – temporiza as operações de carga de paletes nos AGVs;
- TEMP_RECEPCAO_PALETES:TEMPORIZADOR – temporiza as operações de encaminhamento de paletes desde o cais até às várias filas de saída.

D.1.11.2 Posições

- CARGA_AGV_INTERR:ULINT;
- CARREGANDO_AGV:PALETE;
- FIM_TEMP_CARGA_AGV:ULINT;
- FIM_TEMP_RECEP_PALETE:ULINT;
- NOVO_LOTE_PALETES:NUL;
- NUM_PALETES_A_RECEBER:ULINT – o token da marcação inicial é inicializado pela instrução NUM_PALETES_A_RECEBER->i=0;
- OPERACAO_CARGA_INTERR:PALETE;
- PALETE_ESPERA_COLOC_FILA:PALETE;
- PALETE_SENDO_TRANSP_FILA:PALETE;
- TEMPO_CARGA_AGV:TEMPORIZACAO;
- TEMPO_RECEP_PALETE:TEMPORIZACAO.

D.1.11.3 Transições

D.1.11.3.1 CANCELA_CARGA_AGV

D.1.11.3.1.1 Predicado

```
(CARREGANDO_AGV->fila==ERRO_PERM_AGV->i)
```

D.1.11.3.1.2 Acção

```
CARGA_AGV_INTERR->i=CARREGANDO_AGV->fila;

PONTOS_SAIDA_DISPONIVEIS->i=CARREGANDO_AGV->fila;

sprintf(XX->s, "\n%s: operacao de carga cancelada na fila %u
(t=%.3f)\n",
    XX->nome, CARREGANDO_AGV->fila, xx_gettime());
prn_string(XX->s);
```

D.1.11.3.2 CANCELA_CARGA_INTERR

D.1.11.3.2.1 Predicado

```
(OPERACAO_CARGA_INTERR->fila==ERRO_PERM_AGV->i)
```

D.1.11.3.2.2 Acção

```
CARGA_AGV_INTERR->i=OPERACAO_CARGA_INTERR->fila;

PONTOS_SAIDA_DISPONIVEIS->i=OPERACAO_CARGA_INTERR->fila;

sprintf(XX->s, "\n%s: cancelamento da carga na fila %u, apos %.3f s da
ocorr. de um erro (t=%.3f)\n",
    XX->nome, OPERACAO_CARGA_INTERR->fila,
    XX->t-OPERACAO_CARGA_INTERR->t_mud_est, xx_gettime());
prn_string(XX->s);
```

D.1.11.3.3 Acção de 'COLOCA_PALETE_FILA_SAIDA'

```
ulint pos=POSICOES_LIVRES_FILAS->i-XX->fila1+1;
ulint n=elem_l_ulint(pos, XX->p_num_paletes_filas);

XX->t=xx_gettime();

PALETE_FILA_SAIDA->fila=POSICOES_LIVRES_FILAS->i;
PALETE_FILA_SAIDA->t_mud_est=XX->t;

RECURSOS_LIVRES->i=PALETE_ESPERA_COLOC_FILA->recurso;

FILA_PALETE_INSERIDA->i=POSICOES_LIVRES_FILAS->i;

n++;
atribui_ulint_pos(n, pos, XX->p_num_paletes_filas);
xx_writesample(obtem_int(pos, XX->l_ve_estado_filas), n);

sprintf(XX->s, "\n%s: insercao de uma palete na fila %u (t=%.3f)\n",
    XX->nome, POSICOES_LIVRES_FILAS->i, XX->t);
```

```
prn_string(XX->s);
```

D.1.11.3.4 INICIA_CARGA_AGV

D.1.11.3.4.1 Predicado

```
(INICIO_OPERACAO_CARGA->i>=XX->fila1 &&
INICIO_OPERACAO_CARGA->i<=XX->filan
)
```

D.1.11.3.4.2 Acção

```
XX->t=xx_gettime();

CARREGANDO_AGV->fila=INICIO_OPERACAO_CARGA->i;
CARREGANDO_AGV->recurso=0;
CARREGANDO_AGV->dt_prox_oper=rnd_normal(XX->vm_tcarga,
                        XX->dp_tcarga);
CARREGANDO_AGV->B_mud_est=XX->t;

TEMPO_CARGA_AGV->id=INICIO_OPERACAO_CARGA->i;
TEMPO_CARGA_AGV->t=CARREGANDO_AGV->dt_prox_oper;

/* xx_fdelay(XT_TERMINA_CARGA_AGV, CARREGANDO_AGV->dt_prox_oper);*/

sprintf(XX->s, "\n%s: inicio da carga de um AGV na fila %u (t=%.3f)\n",
   XX->nome, CARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  vai demorar %.3f s\n", CARREGANDO_AGV->dt_prox_oper);
prn_string(XX->s);
```

D.1.11.3.5 INICIA_RECEPCAO_PALETE

D.1.11.3.5.1 Predicado

```
(NUM_PALETES_A_RECEBER->i>0)
```

D.1.11.3.5.2 Acção

```
double delay=rnd_normal(XX->tm_recep_pal, XX->dp_trecep_pal);

XX->t=xx_gettime();

(NUM_PALETES_A_RECEBER->i)--;

PALETE_SENDO_TRANSP_FILA->fila=0;
PALETE_SENDO_TRANSP_FILA->destino=0;
PALETE_SENDO_TRANSP_FILA->recurso=RECURSOS_LIVRES->i;
PALETE_SENDO_TRANSP_FILA->dt_prox_oper=0.0;
PALETE_SENDO_TRANSP_FILA->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: inicio da recepcao de uma palete (t=%.3f)\n",
   XX->nome, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  chegara a zona das filas de saida em t=%.3f\n",
   XX->t+delay);
prn_string(XX->s);
```

```
/* xx_fdelay(XT_PALETE_CHEGA_FILA, delay);*/
 TEMPO_RECEP_PALETE->id=RECURSOS_LIVRES->i;
 TEMPO_RECEP_PALETE->t=delay;
```

D.1.11.3.6 INTERROMPE_CARGA

D.1.11.3.6.1 Predicado

```
(CARREGANDO_AGV->fila==ERRO_TEMP_AGV->i)
```

D.1.11.3.6.2 Acção

```
XX->t=xx_gettime();

(OPERACAO_CARGA_INTERR->dt_prox_oper)-=XX->t-CARREGANDO_AGV->t_mud_est;
OPERACAO_CARGA_INTERR->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: operacao de carga interrompida na fila %u
(t=%.3f)\n",
    XX->nome, CARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);
```

D.1.11.3.7 Predicado de 'LIMPA_ERRO_PERM_AGV'

```
(ERRO_PERM_AGV->i<XX->fila1 || ERRO_PERM_AGV->i>XX->filan)
```

D.1.11.3.8 Predicado de 'LIMPA_ERRO_TEMP_AGV'

```
(ERRO_TEMP_AGV->i<XX->fila1 || ERRO_TEMP_AGV->i>XX->filan)
```

D.1.11.3.9 Predicado de 'LIMPA_ERRO_TEMP_RESOLV'

```
(ERRO_TEMP_AGV->i<XX->fila1 || ERRO_TEMP_AGV->i>XX->filan)
```

D.1.11.3.10 LIMPA_INICIO_OPERACAO_CARGA

D.1.11.3.11 PALETE_CHEGA_FILA

D.1.11.3.12.1 Predicado

```
(PALETE_SENDO_TRANSP_FILA->recurso==FIM_TEMP_RECEP_PALETE->i)
```

D.1.11.3.12.2 Acção

```
XX->t=xx_gettime();

PALETE_ESPERA_COLOC_FILA->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: nova palete pronta a ser colocada numa das filas
de saida (t=%.3f)\n",
    XX->nome, XX->t);
prn_string(XX->s);
```

D.1.11.3.13 RECEBE_LOTE_PALETES

D.1.11.3.14 RETOMA_CARGA

D.1.11.3.14.1 Predicado

```
(OPERACAO_CARGA_INTERR->fila==ERRO_TEMP_RESOLV->i)
```

D.1.11.3.14.2 Acção

```
XX->t=xx_gettime();

TEMPO_CARGA_AGV->id=OPERACAO_CARGA_INTERR->fila;
TEMPO_CARGA_AGV->t=OPERACAO_CARGA_INTERR->dt_prox_oper;

/* xx_fdelay(XT_TERMINA_CARGA_AGV,
    CARREGANDO_AGV->dt_prox_oper);*/

sprintf(XX->s, "\n%s: fim da interrupcao da carga na fila %u, passados
%.3f s (t=%.3f)\n",
    XX->nome, CARREGANDO_AGV->fila,
    XX->t-CARREGANDO_AGV->t_mud_est, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  a operacao de carga vai durar mais %.3f s\n",
    CARREGANDO_AGV->dt_prox_oper);
prn_string(XX->s);

CARREGANDO_AGV->t_mud_est=XX->t;
```

D.1.11.3.15 TERMINA_CARGA_AGV

D.1.11.3.15.1 Predicado

```
(CARREGANDO_AGV->fila==FIM_TEMP_CARGA_AGV->i)
```

D.1.11.3.15.2 Acção

```
ulint pos=CARREGANDO_AGV->fila-XX->fila1+1;
ulint n_ins=elem_l_ulint(pos, XX->p_num_paletes_filas);
ulint n_ret=elem_l_ulint(pos, XX->p_num_paletes_retiradas_filas);

n_ins--;
atribui_ulint_pos(n_ins, pos, XX->p_num_paletes_filas);
xx_writesample(obtem_int(pos, XX->l_ve_estado_filas), n_ins);
n_ret++;
atribui_ulint_pos(n_ret, pos, XX->p_num_paletes_retiradas_filas);
xx_writesample(obtem_int(pos, XX->l_ve_paletes_retiradas_filas),
n_ret);

xx_setstate(xx_getstate()-1);
xx_writesample(XX->ve_num_paletes_exist, xx_getstate());

XX->t=xx_gettime();

POSICOES_LIVRES_FILAS->i=CARREGANDO_AGV->fila;
FIM_OPERACAO_CARGA->i=CARREGANDO_AGV->fila;
PONTOS_SAIDA_DISPONIVEIS->i=CARREGANDO_AGV->fila;
```

```
PAL_ACABA_SAIR_ARMAZEM->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: fim da carga de um AGV na fila %u (t=%.3f)\n",
   XX->nome, CARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);
```

D.1.11.3.16 Acção de 'TERMINA_RECEPCAO_LOTE'

```
double delay=rnd_exp_neg(XX->tm_entre_lotes);

XX->t=xx_gettime();

(NUM_PALETES_A_RECEBER->i)+=XX->tam_lote_paletes;

xx_setstate(xx_getstate()+XX->tam_lote_paletes);
xx_writesample(XX->ve_num_paletes_exist, xx_getstate());

xx_fdelay(XT_RECEBE_LOTE_PALETES, delay);

sprintf(XX->s, "\n%s: chegada de um novo lote de %u paletes
(t=%.3f)\n",
    XX->nome, XX->tam_lote_paletes, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  proximo lote chegara em t=%.3f\n", XX->t+delay);
prn_string(XX->s);
```

D.1.12 Entidades da RP da página 'inicializa_modelo'

D.1.12.1 Posições

- CONTADOR_FILA:ULINT;
- CONTADOR_POS:ULINT;
- CONTADOR_RECURSOS:ULINT;
- CONT_PS_DISP_E_CAP:ULINT;
- INICIO_MODELO:NUL.

D.1.12.2 Transições

D.1.12.2.1 CRIA_POSICAO_LIVRE

D.1.12.2.1.1 Predicado

```
(CONTADOR_FILA->i<=XX->filan)
```

D.1.12.2.1.2 Acção

```
POSICOES_LIVRES_FILAS->i=CONTADOR_FILA->i;
(CONTADOR_FILA->i)++;
```

D.1.12.2.2 CRIA_RECURSO_LIVRE

D.1.12.2.2.1 Predicado

```
(CONTADOR_RECURSOS->i<=XX->num_recursos)
```

D.1.12.2.2.2 Acção

```
RECURSOS_LIVRES->i=CONTADOR_RECURSOS->i;
(CONTADOR_RECURSOS->i)++;
```

D.1.12.2.3 GERA_PS_DISPONIVEL_E_CAP

D.1.12.2.3.1 Predicado

```
(CONT_PS_DISP_E_CAP->i<=XX->filan)
```

D.1.12.2.3.2 Acção

```
PONTOS_SAIDA_DISPONIVEIS->i=CONT_PS_DISP_E_CAP->i;

CAPACIDADE_FILA->fila=CONT_PS_DISP_E_CAP->i;
CAPACIDADE_FILA->cap=XX->capacidade_filas;

(CONT_PS_DISP_E_CAP->i)++;
```

D.1.12.2.4 Acção de 'INICIA_CONT_FILA_E_CONT_POS'

```
CONTADOR_FILA->i=XX->fila1;
CONTADOR_POS->i=1;
```

D.1.12.2.5 Acção de 'INICIA_CRIACAO_RECURSOS'

```
CONTADOR_RECURSOS->i=1;
```

D.1.12.2.6 Acção de 'INICIA_GER_PS_DISP_E_CAP'

```
CONT_PS_DISP_E_CAP->i=XX->fila1;
```

D.1.12.2.7 PREP_GER_OUTRA_POSICAO

D.1.12.2.7.1 Predicado

```
(CONTADOR_POS->i<XX->capacidade_filas)
```

D.1.12.2.7.2 Acção

```
CONTADOR_FILA->i=XX->fila1;
(CONTADOR_POS->i)++;
```

D.2 Classe ARMAZEM_AUTOMATICO

D.2.1 CMN units incluídas

- PALETE;
- ULINT;
- TEMPORIZACAO;
- CAPAC_FILA.

D.2.2 Posições de entrada

- ERRO_TEMP_AGV:ULINT;
- ERRO_PERM_AGV:ULINT;
- ERRO_TEMP_RESOLV:ULINT;
- INICIO_OPERACAO_DESCARGA:ULINT;
- INICIO_OPERACAO_CARGA:ULINT;
- PEDIDO_SAIDA:PALETE.

D.2.3 Posições de saída

- FIM_OPERACAO_DESCARGA:ULINT;
- POSICOES_LIVRES_FILAS_ENTR:ULINT;
- PALETES_ARMAZENADAS:PALETE;
- FIM_OPERACAO_CARGA:ULINT;
- POSICOES_LIVRES_FILAS_SAIDA:ULINT;
- PONTOS_SAIDA_DISPONIVEIS:ULINT;
- PAL_RETIDAS_SAIDA_ARMAZEM:PALETE;
- PAL_ACABA_SAIR_ARMAZEM:PALETE;
- FILA_PALETE_INSERIDA:ULINT;
- CAPACIDADE_FILA:CAPAC_FILA.

D.2.4 Parâmetros

- vm_tcd:double – valor médio do tempo de carga/descarga de uma palete num AGV;
- dp_tcd:double – desvio padrão do tempo de carga/descarga de uma palete num AGV;
- capac_filas_entrada:unsigned long int – capacidade das filas de entrada;
- capac_filas_saida:unsigned long int – capacidade das filas de saída;

- fila_entr_1:unsigned long int – identificador da 1ª fila de entrada;
- fila_entr_n:unsigned long int – identificador da n-ésima fila de entrada;
- fila_saida_1:unsigned long int – identificador da 1ª fila de saída;
- fila_saida_n:unsigned long int – identificador da n-ésima fila de saída;
- nome:xx_string – nome identificador da entidade;
- capac_armazem:unsigned long int – capacidade de armazenamento;
- vm_tstoc_vazio:double – valor médio do tempo associado ao movimento do stocador, quando vazio;
- dp_tstoc_vazio:double – desvio padrão do tempo associado ao movimento do stocador, quando vazio;
- vm_tstoc_carreg:double – valor médio do tempo associado ao movimento do stocador, quando carregado;
- dp_tstoc_carreg:double – desvio padrão do tempo associado ao movimento do stocador, quando carregado;
- vm_tcd_stoc:double – valor médio do tempo de carga/descarga do stocador;
- dp_tcd_stoc:double – desvio padrão do tempo de carga/descarga do stocador.

D.2.5 Ficheiros de cabeçalho incluídos

```
#include "listas.h"
#include "mens_log.h"
#include "rand_fun.h"
```

D.2.6 Funções definidas internamente

```
/* funcao para permitir a visualizacao de um token do tipo
ULINT (utilizando o comando "examine Queue") */
void prn_ULINT_ARMAZ_AUTOM(a)
 ULINT *a;
{
  char s[255];

  sprintf(s, "(ulint) i=%u\n", a->i);
  prn_string(s);
}
```

D.2.7 Acções iniciais

```
ap_no_1_int u1, u2;
ap_no_1_ulint t1, t2;
unsigned long int i;

/* inicializa variaveis utilizadas para processamento de estatisticas */

sprintf(XX->s, "num_paletes_armaz_%s", XX->nome);
XX->ve_num_paletes_armaz=xx_initmeasure(XX->s);
xx_writesample(XX->ve_num_paletes_armaz, 0);
```

```
/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o estado das filas
de entrada, uma lista com os identificadores retornados
pelas declaracoes de medidas relacionadas com o numero de
paletes retiradas das filas de entrada, uma lista com o
numero de paletes existentes em cada fila de entrada e
uma lista com o numero de paletes retiradas de cada fila
de entrada */
XX->l_ve_est_filas_entr=NULL;
u1=NULL;
XX->l_ve_pal_ret_filas_entr=NULL;
u2=NULL;
XX->p_num_pal_filas_entr=NULL;
t1=NULL;
XX->p_num_pal_ret_filas_entr=NULL;
t2=NULL;
for (i=XX->fila_entr_1;(i<=XX->fila_entr_n || i==XX->fila_entr_1);i++)
{
  sprintf(XX->s, "estado_fila_%u", i);
  insere_int(xx_initmeasure(XX->s), &XX->l_ve_est_filas_entr, &u1);
  xx_writesample(u1->i, 0);

  sprintf(XX->s, "paletes_retiradas_fila_%u", i);
  insere_int(xx_initmeasure(XX->s), &XX->l_ve_pal_ret_filas_entr, &u2);
  xx_writesample(u2->i, 0);

  insere_ulint(0, &XX->p_num_pal_filas_entr, &t1, 1);

  insere_ulint(0, &XX->p_num_pal_ret_filas_entr, &t2, 1);
}


/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o estado das filas
de saida, uma lista com os identificadores retornados
pelas declaracoes de medidas relacionadas com o numero de
paletes retiradas das filas de saida, uma lista com o
numero de paletes existentes em cada fila de saida e
uma lista com o numero de paletes retiradas de cada fila
de saida */
XX->l_ve_est_filas_saida=NULL;
u1=NULL;
XX->l_ve_pal_ret_filas_saida=NULL;
u2=NULL;
XX->p_num_pal_filas_saida=NULL;
t1=NULL;
XX->p_num_pal_ret_filas_saida=NULL;
t2=NULL;
for (i=XX->fila_saida_1;(i<=XX->fila_saida_n || i==XX-
>fila_saida_1);i++)
{
  sprintf(XX->s, "estado_fila_%u", i);
  insere_int(xx_initmeasure(XX->s), &XX->l_ve_est_filas_saida, &u1);
  xx_writesample(u1->i, 0);

  sprintf(XX->s, "paletes_retiradas_fila_%u", i);
  insere_int(xx_initmeasure(XX->s), &XX->l_ve_pal_ret_filas_saida, &u2);
  xx_writesample(u2->i, 0);
```

```
  insere_ulint(0, &XX->p_num_pal_filas_saida, &t1, 1);

  insere_ulint(0, &XX->p_num_pal_ret_filas_saida, &t2, 1);
}


/* Examine Queue em posicoes de tipos nao geridos automatica/ */

 XW_ULINT_print = prn_ULINT_ARMAZ_AUTOM;


/* inicializa o estado do objecto */
/* (indica o numero de paletes existentes no Armazem,
incluindo as filas de entrada e de saida (WIP) */

xx_setstate(0);
```

D.2.8 Acções finais

```
/* destroi as listas de identificadores de variaveis estatisticas */

limpa_l_int(&XX->l_ve_est_filas_entr);
limpa_l_int(&XX->l_ve_pal_ret_filas_entr);
limpa_l_int(&XX->l_ve_est_filas_saida);
limpa_l_int(&XX->l_ve_pal_ret_filas_saida);


/* destroi as listas com os contadores de paletes das filas */

limpa_l_ulint(XX->p_num_pal_filas_entr);
limpa_l_ulint(XX->p_num_pal_ret_filas_entr);
limpa_l_ulint(XX->p_num_pal_filas_saida);
limpa_l_ulint(XX->p_num_pal_ret_filas_saida);
```

D.2.9 Variáveis locais

- t:double – variável utilizada para guardar o tempo de simulação;
- l_ve_est_filas_entr:ap_no_l_int – lista de identificadores das variáveis estatísticas relacionadas com o estado das filas de entrada;
- l_ve_pal_ret_filas_entr:ap_no_l_int – lista de identificadores das variáveis estatísticas relacionadas com o número de paletes retiradas das filas de entrada;
- l_ve_est_filas_saida:ap_no_l_int – lista de identificadores das variáveis estatísticas relacionadas com o estado das filas de saída;
- l_ve_pal_ret_filas_saida:ap_no_l_int – lista de identificadores das variáveis estatísticas relacionadas com o número de paletes retiradas das filas de saída;
- p_num_pal_filas_entr:ap_no_l_ulint – apontador para a cabeça da lista que guarda o numero de paletes existentes em cada fila de entrada;
- p_num_pal_filas_saida:ap_no_l_ulint – apontador para a cabeça da lista que guarda o número de paletes existentes em cada fila de saída;
- p_num_pal_ret_filas_entr:ap_no_l_ulint – apontador para a cabeça da lista que guarda o número de paletes retiradas de cada fila de entrada;

- p_num_pal_ret_filas_saida:ap_no_l_ulint – apontador para a cabeça da lista que guarda o número de paletes retiradas de cada fila de saída;
- s:xx_string – buffer de caracteres utilizado em várias partes do modelo;
- ve_num_paletes_armaz:int – identificador da variável estatística "número de paletes armazenadas".

D.2.10 Posições partilhadas

- CELULAS_LIVRES:NUL;
- STOCADOR_LIVRE:NUL;
- TEMPO_CD_AGV:TEMPORIZACAO;
- OP_CD_INTERR:ULINT;
- FIM_OP_CD:ULINT.

D.2.11 Entidades da RP da página principal

D.2.11.1 Objecto 'TEMP_CD_AGV:TEMPORIZADOR'

É utilizado para temporizar operações de carga/descarga de AGVs.

D.2.11.2 Transições

D.2.11.2.1 Predicado de 'LIMPA_ERRO_PERM_AGV'

```
((ERRO_PERM_AGV->i<XX->fila_entr_1 || ERRO_PERM_AGV->i>XX->fila_entr_n)
&&
 (ERRO_PERM_AGV->i<XX->fila_saida_1 || ERRO_PERM_AGV->i>XX-
>fila_saida_n)
)
```

D.2.11.2.2 Predicado de 'LIMPA_ERRO_TEMP_AGV'

```
 ((ERRO_TEMP_AGV->i<XX->fila_entr_1 || ERRO_TEMP_AGV->i>XX->fila_entr_n)
&&
 (ERRO_TEMP_AGV->i<XX->fila_saida_1 || ERRO_TEMP_AGV->i>XX-
>fila_saida_n)
)
```

D.2.11.2.3 Predicado de 'LIMPA_ERRO_TEMP_RESOLV'

```
((ERRO_TEMP_RESOLV->i<XX->fila_entr_1 || ERRO_TEMP_RESOLV->i>XX-
>fila_entr_n) &&
 (ERRO_TEMP_RESOLV->i<XX->fila_saida_1 || ERRO_TEMP_RESOLV->i>XX-
>fila_saida_n)
)
```

D.2.12 Entidades da RP da página 'entrada_paletes'

D.2.12.1 Posições

- DESCARREGANDO_AGV:PALETE;
- OPERACAO_DESCARGA_INTERR:PALETE;
- PALETE_ESPERANDO_ARMAZEN:PALETE;
- STOC_CARREG_PAL:PALETE;
- STOC_MOVIM_FILA_ENTR:NUL;
- STOC_MOVIM_PAL_ARMAZ:PALETE;
- STOC_TERMINOU_MOVIM_FILA_ENTR:NUL.

D.2.12.2 Transições

D.2.12.2.1 CANCELA_DESCARGA

D.2.12.2.1.1 Predicado

```
(DESCARREGANDO_AGV->fila==ERRO_PERM_AGV->i)
```

D.2.12.2.1.2 Acção

```
ulint pos=DESCARREGANDO_AGV->fila-XX->fila_entr_1+1;
ulint n=elem_1_ulint(pos, XX->p_num_pal_filas_entr);

XX->t=xx_gettime();

n++;
atribui_ulint_pos(n, pos, XX->p_num_pal_filas_entr);
xx_writesample(obtem_int(pos, XX->l_ve_est_filas_entr), n);

PALETE_ESPERANDO_ARMAZEN->dt_prox_oper=0.0;
PALETE_ESPERANDO_ARMAZEN->t_mud_est=XX->t;

OP_CD_INTERR->i=DESCARREGANDO_AGV->fila;

sprintf(XX->s, "\n%s: operacao de descarga cancelada na fila %u
(t=%.3f)\n",
    XX->nome, DESCARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);

xx_setstate(xx_getstate()+1);
xx_writesample(XX->ve_num_paletes_armaz, xx_getstate());
```

D.2.12.2.2 CANCELA_DESCARGA_INTERR

D.2.12.2.2.1 Predicado

```
(OPERACAO_DESCARGA_INTERR->fila==ERRO_PERM_AGV->i)
```

D.2.12.2.2.2 Acção

```
ulint pos=OPERACAO_DESCARGA_INTERR->fila-XX->fila_entr_1+1;
ulint n=elem_1_ulint(pos, XX->p_num_pal_filas_entr);

XX->t=xx_gettime();

n++;
atribui_ulint_pos(n, pos, XX->p_num_pal_filas_entr);
xx_writesample(obtem_int(pos, XX->l_ve_est_filas_entr), n);

sprintf(XX->s, "\n%s: cancelamento da descarga na fila %u, apos %.3f s
da ocorr. de um erro (t=%.3f)\n",
    XX->nome, PALETE_ESPERANDO_ARMAZEN->fila,
    XX->t-PALETE_ESPERANDO_ARMAZEN->t_mud_est, XX->t);
prn_string(XX->s);

PALETE_ESPERANDO_ARMAZEN->t_mud_est=XX->t;
PALETE_ESPERANDO_ARMAZEN->dt_prox_oper=0.0;

OP_CD_INTERR->i=OPERACAO_DESCARGA_INTERR->fila;

xx_setstate(xx_getstate()+1);
xx_writesample(XX->ve_num_paletes_armaz, xx_getstate());
```

D.2.12.2.3 Acção de 'INICIA_ARMAZEN_PALETE'

```
double delay=rnd_normal(XX->vm_tcd_stoc, XX->dp_tcd_stoc);

XX->t=xx_gettime();

STOC_INSERE_PAL_ARMAZ->t_mud_est=XX->t;
xx_fdelay(XT_TERMINA_ARMAZEN_PAL, delay);

sprintf(XX->s, "\n%s: o stocador chegou com a palete a uma das celulas
do armaz. (t=%.3f)\n",
    XX->nome, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  vai iniciar a descarga da palete, que demorara %.3f
s\n", delay);
prn_string(XX->s);
```

D.2.12.2.4 Acção de 'INICIA_CARGA_PALETE_STOCADOR'

```
double delay=rnd_normal(XX->vm_tcd_stoc, XX->dp_tcd_stoc);

XX->t=xx_gettime();

xx_fdelay(XT_TERMINA_CARGA_PAL_STOC, delay);

STOC_CARREG_PAL->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: o stocador vai iniciar a carga de uma palete da
fila %u (t=%.3f)\n",
    XX->nome, PAL_ESPERA_STOC_FILA_ENTR->fila, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  vai demorar %.3f s\n", delay);
```

D.2.12.2.5 INICIA_DESCARGA_AGV

D.2.12.2.5.1 Predicado

```
(INICIO_OPERACAO_DESCARGA->i>=XX->fila_entr_1 &&
INICIO_OPERACAO_DESCARGA->i<=XX->fila_entr_n
)
```

D.2.12.2.5.2 Acção

```
XX->t=xx_gettime();

DESCARREGANDO_AGV->fila=INICIO_OPERACAO_DESCARGA->i;
DESCARREGANDO_AGV->destino=0;
DESCARREGANDO_AGV->recurso=0;
DESCARREGANDO_AGV->dt_prox_oper=rnd_normal(XX->vm_tcd,
                                XX->dp_tcd);
DESCARREGANDO_AGV->t_mud_est=XX->t;

TEMPO_CD_AGV->id=INICIO_OPERACAO_DESCARGA->i;
TEMPO_CD_AGV->t=DESCARREGANDO_AGV->dt_prox_oper;

/* xx_fdelay(XT_TERMINA_DESCARGA_AGV,
    DESCARREGANDO_AGV->dt_prox_oper);*/

sprintf(XX->s, "\n%s: inicio da descarga de um AGV na fila %u
(t=%.3f)\n",
    XX->nome, DESCARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  vai demorar %.3f s\n", DESCARREGANDO_AGV-
>dt_prox_oper);
prn_string(XX->s);
```

D.2.12.2.6 Acção de 'INICIA_MOV_STOC_FILA_ENTR'

```
double delay=rnd_normal(XX->vm_tstoc_vazio, XX->dp_tstoc_vazio);

XX->t=xx_gettime();

PAL_ESPERA_STOC_FILA_ENTR->t_mud_est=XX->t;

xx_fdelay(XT_STOC_TERMINA_MOVIM_FILA_ENTR, delay);

sprintf(XX->s, "\n%s: o stocador vazio iniciou o movimento ate\n as
filas de entrada (t=%.3f)\n",
    XX->nome, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  vai demorar %.3f s\n", delay);
prn_string(XX->s);
```

D.2.12.2.7 INTERROMPE_DESCARGA

D.2.12.2.7.1 Predicado

```
(DESCARREGANDO_AGV->fila==ERRO_TEMP_AGV->i)
```

D.2.12.2.7.2 Acção

```
XX->t=xx_gettime();

(OPERACAO_DESCARGA_INTERR->dt_prox_oper)-=XX->t-DESCARREGANDO_AGV-
>t_mud_est;
OPERACAO_DESCARGA_INTERR->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: operacao de descarga interrompida na fila %u
(t=%.3f)\n",
    XX->nome, DESCARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);
```

D.2.12.2.8 LIMPA_INICIO_OPERACAO_DESCARGA

D.2.12.2.9 RETOMA_DESCARGA

D.2.12.2.9.1 Predicado

```
(OPERACAO_DESCARGA_INTERR->fila==ERRO_TEMP_RESOLV->i)
```

D.2.12.2.9.2 Acção

```
XX->t=xx_gettime();

TEMPO_CD_AGV->id=OPERACAO_DESCARGA_INTERR->fila;
TEMPO_CD_AGV->t=OPERACAO_DESCARGA_INTERR->dt_prox_oper;

/* xx_fdelay(XT_TERMINA_DESCARGA_AGV,
    DESCARREGANDO_AGV->dt_prox_oper);*/

sprintf(XX->s, "\n%s: fim da interrupcao da descarga na fila %u,
passados %.3f s (t=%.3f)\n",
    XX->nome, DESCARREGANDO_AGV->fila,
    XX->t-OPERACAO_DESCARGA_INTERR->t_mud_est, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  a operacao de descarga vai durar mais %.3f s\n",
    DESCARREGANDO_AGV->dt_prox_oper);
prn_string(XX->s);

DESCARREGANDO_AGV->t_mud_est=XX->t;
```

D.2.12.2.10 Acção de 'STOC_TERMINA_MOVIM_FILA_ENTR'

```
XX->t=xx_gettime();

sprintf(XX->s, "\n%s: o stocador terminou o movimento\n",
    XX->nome);
prn_string(XX->s);
```

D.2.12.2.11 Acção de 'TERMINA_ARMAZEN_PAL'

```
XX->t=xx_gettime();

PALETES_ARMAZENADAS->t_mud_est=XX->t;
```

```
 sprintf(XX->s, "\n%s: o stocador terminou a insercao de uma palete no
armazem (t=%.3f)\n",
    XX->nome, XX->t);
 prn_string(XX->s);
```

D.2.12.2.12 Acção de 'TERMINA_CARGA_PAL_STOC'

```
 ulint pos=STOC_CARREG_PAL->fila-XX->fila_entr_1+1;
 ulint n_ins=elem_1_ulint(pos, XX->p_num_pal_filas_entr);
 ulint n_ret=elem_1_ulint(pos, XX->p_num_pal_ret_filas_entr);
 double delay=rnd_normal(XX->vm_tstoc_carreg, XX->dp_tstoc_carreg);

 XX->t=xx_gettime();

 n_ins--;
 atribui_ulint_pos(n_ins, pos, XX->p_num_pal_filas_entr);
 xx_writesample(obtem_int(pos, XX->l_ve_est_filas_entr), n_ins);
 n_ret++;
 atribui_ulint_pos(n_ret, pos, XX->p_num_pal_ret_filas_entr);
 xx_writesample(obtem_int(pos, XX->l_ve_pal_ret_filas_entr), n_ret);

 STOC_MOVIM_PAL_ARMAZ->t_mud_est=XX->t;
 POSICOES_LIVRES_FILAS_ENTR->i=STOC_CARREG_PAL->fila;

 xx_fdelay(XT_INICIA_ARMAZEN_PALETE, delay);

 sprintf(XX->s, "\n%s: o stocador acabou de carregar uma palete da fila
%u (t=%.3f)\n",
    XX->nome, STOC_CARREG_PAL->fila, XX->t);
 prn_string(XX->s);
 sprintf(XX->s, "  vai iniciar a movim. da palete ate ao armazem, que
demorara %.3f s\n", delay);
 prn_string(XX->s);

 STOC_MOVIM_PAL_ARMAZ->fila=0;
```

D.2.12.2.13 TERMINA_DESCARGA_AGV

D.2.12.2.13.1 Predicado

```
(DESCARREGANDO_AGV->fila==FIM_OP_CD->i)
```

D.2.12.2.13.2 Acção

```
ulint pos=DESCARREGANDO_AGV->fila-XX->fila_entr_1+1;
ulint n=elem_1_ulint(pos, XX->p_num_pal_filas_entr);

XX->t=xx_gettime();

n++;
atribui_ulint_pos(n, pos, XX->p_num_pal_filas_entr);
xx_writesample(obtem_int(pos, XX->l_ve_est_filas_entr), n);

PALETE_ESPERANDO_ARMAZEN->dt_prox_oper=0.0;
PALETE_ESPERANDO_ARMAZEN->t_mud_est=XX->t;

FIM_OPERACAO_DESCARGA->i=DESCARREGANDO_AGV->fila;
```

```
sprintf(XX->s, "\n%s: fim da descarga de um AGV na fila %u (t=%.3f)\n",
   XX->nome, DESCARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);

xx_setstate(xx_getstate()+1);
xx_writesample(XX->ve_num_paletes_armaz, xx_getstate());
```

D.2.13 Entidades da RP da página 'inicializa_modelo'

D.2.13.1 Posições

- CONTADOR_CELULAS:ULINT;
- CONTADOR_POS_FE:ULINT;
- CONTADOR_POS_FS:ULINT;
- CONT_FILA_ENTR:ULINT;
- CONT_FILA_SAIDA:ULINT;
- CONT_PS_DISP_E_CAP:ULINT;
- INICIO_MODELO:NUL.

D.2.13.2 Transições

D.2.13.2.1 CRIA_POSICAO_LIVRE_FE

D.2.13.2.1.1 Predicado

```
(CONT_FILA_ENTR->i<=XX->fila_entr_n)
```

D.2.13.2.1.2 Acção

```
POSICOES_LIVRES_FILAS_ENTR->i=CONT_FILA_ENTR->i;
(CONT_FILA_ENTR->i)++;
```

D.2.13.2.2 CRIA_POSICAO_LIVRE_FS

D.2.13.2.2.1 Predicado

```
(CONT_FILA_SAIDA->i<=XX->fila_saida_n)
```

D.2.13.2.2.2 Acção

```
POSICOES_LIVRES_FILAS_SAIDA->i=CONT_FILA_SAIDA->i;
(CONT_FILA_SAIDA->i)++;
```

D.2.13.2.3 GERA_CELULAS_LIVRES

D.2.13.2.3.1 Predicado

```
(CONTADOR_CELULAS->i<XX->capac_armazem)
```

D.2.13.2.3.2 Acção

```
(CONTADOR_CELULAS->i)++;
```

D.2.13.2.4 GERA_PS_DISPONIVEL_E_CAP

D.2.13.2.4.1 Predicado

```
(CONT_PS_DISP_E_CAP->i<=XX->fila_saida_n)
```

D.2.13.2.4.2 Acção

```
PONTOS_SAIDA_DISPONIVEIS->i=CONT_PS_DISP_E_CAP->i;

CAPACIDADE_FILA->fila=CONT_PS_DISP_E_CAP->i;
CAPACIDADE_FILA->cap=XX->capac_filas_saida;

(CONT_PS_DISP_E_CAP->i)++;
```

D.2.13.2.5 Acção de 'INICIA_CONT_FILA_ENTR_E_CONT_POS'

```
CONT_FILA_ENTR->i=XX->fila_entr_1;
CONTADOR_POS_FE->i=1;
```

D.2.13.2.6 Acção de 'INICIA_CONT_FS_E_CONT_POS'

```
CONT_FILA_SAIDA->i=XX->fila_saida_1;
CONTADOR_POS_FS->i=1;
```

D.2.13.2.7 Acção de 'INICIA_GERAC_CELULAS_LIVRES'

```
CONTADOR_CELULAS->i=0;
```

D.2.13.2.8 Acção de 'INICIA_GER_PS_DISP_E_CAP'

```
CONT_PS_DISP_E_CAP->i=XX->fila_saida_1;
```

D.2.13.2.9 PREP_GER_OUTRA_POS_FE

D.2.13.2.9.1 Predicado

```
(CONTADOR_POS_FE->i<XX->capac_filas_entrada)
```

D.2.13.2.9.2 Acção

```
CONT_FILA_ENTR->i=XX->fila_entr_1;
(CONTADOR_POS_FE->i)++;
```

D.2.13.2.10 PREP_GER_OUTRA_POS_FS

D.2.13.2.10.1 Predicado

```
(CONTADOR_POS_FS->i<XX->capac_filas_saida)
```

D.2.13.2.10.2 Acção

```
CONT_FILA_SAIDA->i=XX->fila_saida_1;
```

```
(CONTADOR_POS_FS->i)++;
```

D.2.13.2.11 TERMINA_INICIAL_MODELO

D.2.14 Entidades da RP da página 'saida_paletes'

D.2.14.1 Posições

- PALETE_PEDIDO_SAIDA:PALETE;
- PAL_AGUARD_SAIDA_ARMAZEM:PALETE;
- PAL_CARGA_AGV_INTERR:PALETE;
- PAL_ESPERA_STOC_CELULA:PALETE;
- PAL_SENDO_CARREG_AGV:PALETE;
- PAL_SENDO_CARREG_STOC:PALETE;
- PAL_SENDO_DESCARR_FILA_SAIDA:PALETE;
- PAL_SENDO_MOVIM_FILA_SAIDA:PALETE;
- STOC_MOVIM_CELULA:NUL;
- STOC_TERMINOU_MOVIM_CELULA:NUL.

D.2.14.2 Transições

D.2.14.2.1 CANCELA_CARGA_AGV

D.2.14.2.1.1 Predicado

```
(PAL_SENDO_CARREG_AGV->fila==ERRO_PERM_AGV->i)
```

D.2.14.2.1.2 Acção

```
XX->t=xx_gettime();

PAL_AGUARD_SAIDA_ARMAZEM->dt_prox_oper=0.0;
PAL_AGUARD_SAIDA_ARMAZEM->t_mud_est=XX->t;

OP_CD_INTERR->i=PAL_SENDO_CARREG_AGV->fila;

PONTOS_SAIDA_DISPONIVEIS->i=PAL_SENDO_CARREG_AGV->fila;

sprintf(XX->s, "\n%s: operacao de carga cancelada na fila %u
(t=%.3f)\n",
    XX->nome, PAL_SENDO_CARREG_AGV->fila, XX->t);
prn_string(XX->s);
```

D.2.14.2.2 CANCELA_CARGA_AGV_INTERR

D.2.14.2.2.1 Predicado

```
(PAL_CARGA_AGV_INTERR->fila==ERRO_PERM_AGV->i)
```

D.2.14.2.2.2 Acção

```
XX->t=xx_gettime();

sprintf(XX->s, "\n%s: cancelamento da carga na fila %u, apos %.3f s da
ocorr. de um erro (t=%.3f)\n",
    XX->nome, PAL_CARGA_AGV_INTERR->fila,
    XX->t-PAL_CARGA_AGV_INTERR->t_mud_est, XX->t);
prn_string(XX->s);

PAL_AGUARD_SAIDA_ARMAZEM->t_mud_est=XX->t;
PAL_AGUARD_SAIDA_ARMAZEM->dt_prox_oper=0.0;

OP_CD_INTERR->i=PAL_CARGA_AGV_INTERR->fila;

PONTOS_SAIDA_DISPONIVEIS->i=PAL_CARGA_AGV_INTERR->fila;
```

D.2.14.2.3 INICIA_CARGA_AGV

D.2.14.2.3.1 Predicado

```
(INICIO_OPERACAO_CARGA->i==PAL_AGUARD_SAIDA_ARMAZEM->fila)
```

D.2.14.2.3.2 Acção

```
XX->t=xx_gettime();

PAL_SENDO_CARREG_AGV->dt_prox_oper=rnd_normal(XX->vm_tcd, XX->dp_tcd);
PAL_SENDO_CARREG_AGV->t_mud_est=XX->t;

TEMPO_CD_AGV->id=PAL_AGUARD_SAIDA_ARMAZEM->fila;
TEMPO_CD_AGV->t=PAL_SENDO_CARREG_AGV->dt_prox_oper;

/* xx_fdelay(XT_TERMINA_CARGA_AGV, PAL_SENDO_CARREG_AGV-
>dt_prox_oper);*/

sprintf(XX->s, "\n%s: inicio da carga de um AGV na fila %u (t=%.3f)\n",
   XX->nome, PAL_SENDO_CARREG_AGV->fila, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  vai demorar %.3f s\n", PAL_SENDO_CARREG_AGV-
>dt_prox_oper);
prn_string(XX->s);
```

D.2.14.2.4 Acção de 'INICIA_DESCARGA_FILA_SAIDA'

```
double delay=rnd_normal(XX->vm_tcd_stoc, XX->dp_tcd_stoc);

XX->t=xx_gettime();

PAL_SENDO_DESCARR_FILA_SAIDA->t_mud_est=XX->t;
xx_fdelay(XT_STOC_TERMINA_DESC_FILA_SAIDA, delay);
```

```
sprintf(XX->s, "\n%s: o stocador chegou com a palete a fila de saida %u
(t=%.3f)\n",
    XX->nome, PAL_SENDO_MOVIM_FILA_SAIDA->fila, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  vai iniciar a descarga da palete, que demorara %.3f
s\n", delay);
prn_string(XX->s);
```

D.2.14.2.5 Acção de 'INICIA_MOVIM_FILA_SAIDA'

```
double delay=rnd_normal(XX->vm_tstoc_carreg, XX->dp_tstoc_carreg);

XX->t=xx_gettime();

PAL_SENDO_MOVIM_FILA_SAIDA->t_mud_est=XX->t;

xx_fdelay(XT_INICIA_DESCARGA_FILA_SAIDA, delay);

sprintf(XX->s, "\n%s: o stocador acabou de carregar uma palete\n de uma
celula do armazem (t=%.3f)\n",
    XX->nome, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  vai iniciar a movim. da palete ate a fila de saida
%u, que demorara %.3f s\n",
    PAL_SENDO_CARREG_STOC->fila, delay);
prn_string(XX->s);
```

D.2.14.2.6 Acção de 'INICIA_MOVIM_STOC_CELULA'

```
double delay=rnd_normal(XX->vm_tstoc_vazio, XX->dp_tstoc_vazio);

XX->t=xx_gettime();

PAL_ESPERA_STOC_CELULA->t_mud_est=XX->t;

xx_fdelay(XT_STOC_TERMINA_MOVIM_CELULA, delay);

sprintf(XX->s, "\n%s: o stocador vazio iniciou o movimento ate\n uma
celula do armazem (t=%.3f)\n",
    XX->nome, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  vai demorar %.3f s\n", delay);
prn_string(XX->s);
```

D.2.14.2.7 Acção de 'INICIA_SAIDA_PAL_CELULA'

```
double delay=rnd_normal(XX->vm_tcd_stoc, XX->dp_tcd_stoc);

XX->t=xx_gettime();

xx_fdelay(XT_INICIA_MOVIM_FILA_SAIDA, delay);

PAL_SENDO_CARREG_STOC->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: o stocador vai iniciar a carga\n de uma celula do
armazem (t=%.3f)\n",
    XX->nome, XX->t);
prn_string(XX->s);
```

```
sprintf(XX->s, "  vai demorar %.3f s\n", delay);
```

D.2.14.2.8 INTERR_CARGA_AGV

D.2.14.2.8.1 Predicado

```
(PAL_SENDO_CARREG_AGV->fila==ERRO_TEMP_AGV->i)
```

D.2.14.2.8.2 Acção

```
XX->t=xx_gettime();

(PAL_CARGA_AGV_INTERR->dt_prox_oper)-=XX->t-PAL_SENDO_CARREG_AGV-
>t_mud_est;
PAL_CARGA_AGV_INTERR->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: operacao de carga interrompida na fila %u
(t=%.3f)\n",
    XX->nome, PAL_SENDO_CARREG_AGV->fila, XX->t);
prn_string(XX->s);
```

D.2.14.2.9 Predicado de 'LIMPA_INICIO_OPERACAO_CARGA'

```
(INICIO_OPERACAO_CARGA->i<XX->fila_saida_1 ||
 INICIO_OPERACAO_CARGA->i>XX->fila_saida_n
)
```

D.2.14.2.10 Acção de 'NOTIF_OCUPACAO_POS_FILA'

```
FILA_PALETE_INSERIDA->i=PEDIDO_SAIDA->fila;
```

D.2.14.2.11 RETOMA_CARGA_AGV

D.2.14.2.11.1 Predicado

```
(PAL_CARGA_AGV_INTERR->fila==ERRO_TEMP_RESOLV->i)
```

D.2.14.2.11.2 Acção

```
XX->t=xx_gettime();

TEMPO_CD_AGV->id=PAL_CARGA_AGV_INTERR->fila;
TEMPO_CD_AGV->t=PAL_CARGA_AGV_INTERR->dt_prox_oper;

/* xx_fdelay(XT_TERMINA_CARGA_AGV,
    PAL_CARGA_AGV_INTERR->dt_prox_oper);*/

sprintf(XX->s, "\n%s: fim da interrupcao da carga na fila %u, passados
%.3f s (t=%.3f)\n",
    XX->nome, PAL_CARGA_AGV_INTERR->fila,
    XX->t-PAL_CARGA_AGV_INTERR->t_mud_est, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  a operacao de carga vai durar mais %.3f s\n",
    PAL_CARGA_AGV_INTERR->dt_prox_oper);
prn_string(XX->s);

PAL_SENDO_CARREG_AGV->t_mud_est=XX->t;
```

D.2.14.2.12 Acção de 'STOC_TERMINA_DESC_FILA_SAIDA'

```
ulint pos=PAL_SENDO_DESCARR_FILA_SAIDA->fila-XX->fila_saida_1-1;
ulint n=elem_l_ulint(pos, XX->p_num_pal_filas_saida);

XX->t=xx_gettime();

n++;
atribui_ulint_pos(n, pos, XX->p_num_pal_filas_saida);
xx_writesample(obtem_int(pos, XX->l_ve_est_filas_saida), n);

PAL_AGUARD_SAIDA_ARMAZEM->fila=PAL_SENDO_DESCARR_FILA_SAIDA->fila;
PAL_AGUARD_SAIDA_ARMAZEM->destino=PAL_SENDO_DESCARR_FILA_SAIDA-
>destino;
PAL_AGUARD_SAIDA_ARMAZEM->t_mud_est=XX->t;
PAL_AGUARD_SAIDA_ARMAZEM->dt_prox_oper=PAL_SENDO_DESCARR_FILA_SAIDA-
>dt_prox_oper;

PAL_RETIDAS_SAIDA_ARMAZEM->fila=PAL_SENDO_DESCARR_FILA_SAIDA->fila;
PAL_RETIDAS_SAIDA_ARMAZEM->destino=PAL_SENDO_DESCARR_FILA_SAIDA-
>destino;
PAL_RETIDAS_SAIDA_ARMAZEM->t_mud_est=XX->t;
PAL_RETIDAS_SAIDA_ARMAZEM->dt_prox_oper=PAL_SENDO_DESCARR_FILA_SAIDA-
>dt_prox_oper;

/* FILA_PALETE_INSERIDA->i=PAL_SENDO_DESCARR_FILA_SAIDA->fila;*/

sprintf(XX->s, "\n%s: o stocador terminou a descarga da palete na fila
de saida %u (t=%.3f)\n",
    XX->nome, PAL_SENDO_DESCARR_FILA_SAIDA->fila, XX->t);
prn_string(XX->s);
```

D.2.14.2.13 Acção de 'STOC_TERMINA_MOVIM_CELULA'

```
XX->t=xx_gettime();

sprintf(XX->s, "\n%s: o stocador chegou a celula do armazem\n",
   XX->nome);
prn_string(XX->s);
```

D.2.14.2.14 TERMINA_CARGA_AGV

D.2.14.2.14.1 Predicado

```
(PAL_SENDO_CARREG_AGV->fila==FIM_OP_CD->i)
```

D.2.14.2.14.2 Acção

```
ulint pos=PAL_SENDO_CARREG_AGV->fila-XX->fila_saida_1+1;
ulint nins=elem_l_ulint(pos, XX->p_num_pal_filas_saida);
ulint nret=elem_l_ulint(pos, XX->p_num_pal_ret_filas_saida);

XX->t=xx_gettime();

nins--;
atribui_ulint_pos(nins, pos, XX->p_num_pal_filas_saida);
xx_writesample(obtem_int(pos, XX->l_ve_est_filas_saida), nins);
nret++;
atribui_ulint_pos(nret, pos, XX->p_num_pal_ret_filas_saida);
```

```
xx_writesample(obtem_int(pos, XX->l_ve_pal_ret_filas_saida), nret);

FIM_OPERACAO_CARGA->i=PAL_SENDO_CARREG_AGV->fila;
POSICOES_LIVRES_FILAS_SAIDA->i=PAL_SENDO_CARREG_AGV->fila;
PONTOS_SAIDA_DISPONIVEIS->i=PAL_SENDO_CARREG_AGV->fila;
PAL_ACABA_SAIR_ARMAZEM->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: fim da carga de um AGV na fila %u (t=%.3f)\n",
   XX->nome, PAL_SENDO_CARREG_AGV->fila, XX->t);
prn_string(XX->s);

xx_setstate(xx_getstate()-1);
xx_writesample(XX->ve_num_paletes_armaz, xx_getstate());
```

D.3 Classe POSTO_TRABALHO

D.3.1 CMN units incluídas

- ULINT;
- PALETE;
- TEMPORIZACAO;
- CAPAC_FILA.

D.3.2 Posições de entrada

- INICIO_OPERACAO_DESCARGA:ULINT;
- ERRO_TEMP_AGV:ULINT;
- ERRO_PERM_AGV:ULINT;
- ERRO_TEMP_RESOLV:ULINT;
- INICIO_OPERACAO_CARGA:ULINT.

D.3.3 Posições de saída

- FIM_OPERACAO_DESCARGA:ULINT;
- POSICOES_LIVRES_FILAS_ENTR:ULINT;
- FIM_OPERACAO_CARGA:ULINT;
- PALETE_AGUARD_SAIDA:PALETE;
- PONTO_SAIDA_DISPONIVEL:ULINT;
- PAL_ACABA_SAIR_PT:PALETE;
- FILA_PALETE_INSERIDA:ULINT;
- CAPACIDADE_FILA:CAPAC_FILA.

D.3.4 Parâmetros

- capac_filas_entrada:unsigned long int – capacidade das filas de entrada;
- dp_tcd:double – desvio padrão do tempo de carga/descarga de uma palete num AGV;
- fila_entr_1:unsigned long int – identificador da 1ª fila de entrada;
- fila_entr_n:unsigned long int – identificador da n-ésima fila de entrada;
- nome:xx_string – nome identificador da entidade.
- vm_tcd:double – valor médio do tempo de carga/descarga de uma palete num AGV;
- fila_saida:unsigned long int – identificador da fila de saída;
- vm_tdownload:double – valor médio do tempo de *download* de paletes (das filas de entrada para o posto de trabalho e do posto de trabalho para a fila de saída);
- dp_tdownload:double – desvio padrão do tempo de *download* de paletes (das filas de entrada para o posto de trabalho e do posto de trabalho para a fila de saída);
- vm_tprocess:double – valor médio do tempo de processamento;
- dp_tprocess:double – desvio padrão do tempo de processamento;
- capac_fila_saida:unsigned long int – capacidade da fila de saída.

D.3.5 Ficheiros de cabeçalho incluídos

```
#include "listas.h"
#include "mens_log.h"
#include "rand_fun.h"
```

D.3.6 Funções definidas internamente

```
int ex_pal_todas_filas_entr(p)
/* funcao que retorna 1 se todas os elemento da lista
tiverem o campo i > 0; retorna 0 no caso contrario */
  ap_no_l_ulint p;
{
/*  prn_string("\nex_pal_todas filas_entr\n");
  prn_l_ulint(p);*/
  if (p==NULL) return 0;
  while (p!=NULL)
  {
    if (p->i==0) return 0;
    p=p->seguinte;
  }
  return 1;
}


/* funcao para permitir a visualizacao de um token do tipo
ULINT (utilizando o comando "examine Queue") */
void prn_ULINT_POSTO_TRABALHO(a)
 ULINT *a;
{
```

```
  char s[255];

  sprintf(s, "(ulint) i=%u\n", a->i);
  prn_string(s);
}
```

D.3.7 Acções iniciais

```
ap_no_1_int u;
ap_no_1_ulint t;
unsigned long int i;

/* inicializa variaveis utilizadas para processamento de estatisticas */

sprintf(XX->s, "num_paletes_exist_%s", XX->nome);
XX->ve_num_paletes_exist=xx_initmeasure(XX->s);
xx_writesample(XX->ve_num_paletes_exist, 0);

XX->num_pal_ret_filas_entr=0;
sprintf(XX->s, "num_pal_ret_filas_entr_%s", XX->nome);
XX->ve_pal_ret_filas_entr=xx_initmeasure(XX->s);
xx_writesample(XX->ve_pal_ret_filas_entr, 0);

XX->num_pal_fila_saida=0;
sprintf(XX->s, "est_fila_saida_%s", XX->nome);
XX->ve_est_fila_saida=xx_initmeasure(XX->s);
xx_writesample(XX->ve_est_fila_saida, 0);

XX->num_pal_ret_fila_saida=0;
sprintf(XX->s, "num_pal_ret_fila_saida_%s", XX->nome);
XX->ve_pal_ret_fila_saida=xx_initmeasure(XX->s);
xx_writesample(XX->ve_pal_ret_fila_saida, 0);


/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o estado das filas
de entrada e uma lista com o numero de paletes existentes
em cada fila de entrada
*/
XX->l_ve_est_filas_entr=NULL;
u=NULL;
XX->p_num_pal_filas_entr=NULL;
t=NULL;

for (i=XX->fila_entr_1;(i<=XX->fila_entr_n || i==XX->fila_entr_1);i++)
{
  sprintf(XX->s, "estado_fila_%u", i);
  insere_int(xx_initmeasure(XX->s), &XX->l_ve_est_filas_entr, &u);
  xx_writesample(u->i, 0);

  insere_ulint(0, &XX->p_num_pal_filas_entr, &t, 1);
}

/*sprintf(XX->s, "\n%s: paletes nas filas de entrada:", XX->nome);
prn_string(XX->s);
prn_l_ulint(XX->p_num_pal_filas_entr);
*/
```

```
/* Examine Queue em posicoes de tipos nao geridos automatica/ */

 XW_ULINT_print = prn_ULINT_POSTO_TRABALHO;


/* inicializa o estado do objecto */
/* indica o numero de paletes existentes no Posto de
Trabalho, incluindo as filas de entrada e de saida (WIP) */

xx_setstate(0);
```

D.3.8 Variáveis locais

- l_ve_est_filas_entr:ap_no_l_int – lista de identificadores das variáveis estatísticas relacionadas com o estado das filas de entrada;
- ve_pal_ret_filas_entr:int – identificador da variável estatística relacionada com o número de paletes retiradas das filas de entrada (cada vez que é retirada uma palete de uma fila de entrada, também é retirada uma palete de todas as outras filas de entrada);
- p_num_pal_filas_entr:ap_no_l_ulint – apontador para a cabeça da lista que guarda o número de paletes existentes em cada fila de entrada;
- num_pal_ret_filas_entr:ulint – número de paletes retiradas de cada uma das filas de entrada (é igual para todas elas);
- s:xx_string – *buffer* de caracteres utilizado em várias partes do modelo;
- t:double – variável utilizada para guardar o tempo de simulação;
- ve_est_fila_saida:int – identificador da variável estatística relacionada com o estado da fila de saída;
- num_pal_fila_saida:ulint – número de paletes existentes na fila de saída;
- num_pal_ret_fila_saida:ulint – número de paletes retiradas da fila de saída;
- ve_pal_ret_fila_saida:int – identificador da variável estatística "número de paletes retiradas da fila de saída";
- ve_num_paletes_exist:int – identificador da variável estatística "número de paletes existentes no posto de trabalho (WIP)".

D.3.9 Posições partilhadas

- PALETE_AGUARD_PROCESS:PALETE – contém *tokens* relativos a paletes esperando nas filas de entrada para serem processadas;
- POSICOES_LIVRES_FILA_SAIDA:NUL – contém *tokens* relativos a posições livres na fila de saída;
- TEMPO_CD_AGV:TEMPORIZACAO – tempo associado a uma operação de carga/descarga de um AGV;
- OP_CD_INTERR:ULINT – interrompe uma temporização associada a uma operação de carga/descarga de um AGV;

- FIM_OP_CD:ULINT – recebe a confirmação do fim de uma operação de carga/descarga de um AGV.

D.3.10 Entidades da RP da página principal

D.3.10.1 Objectos

D.3.10.1.1 TEMP_CD_AGV:TEMPORIZADOR

É utilizado na temporização de operações de carga/descarga de AGVs.

D.3.10.2 Transições

D.3.10.2.1 Predicado de 'LIMPA_ERRO_PERM_AGV'

```
((ERRO_PERM_AGV->i<XX->fila_entr_1 || ERRO_PERM_AGV->i>XX->fila_entr_n)
&&
 ERRO_PERM_AGV->i!=XX->fila_saida
)
```

D.3.10.2.2 Predicado de 'LIMPA_ERRO_TEMP_AGV'

```
 ((ERRO_TEMP_AGV->i<XX->fila_entr_1 || ERRO_TEMP_AGV->i>XX->fila_entr_n)
&&
 ERRO_TEMP_AGV->i!=XX->fila_saida
)
```

D.3.10.2.3 Predicado de 'LIMPA_ERRO_TEMP_RESOLV'

```
 ((ERRO_TEMP_RESOLV->i<XX->fila_entr_1 || ERRO_TEMP_RESOLV->i>XX-
>fila_entr_n) &&
 ERRO_TEMP_RESOLV->i!=XX->fila_saida
)
```

D.3.11 Entidades da RP da página 'entrada_paletes'

D.3.11.1 Posições

- DESCARREGANDO_AGV:PALETE;
- OPERACAO_DESCARGA_INTERR:PALETE.

D.3.11.2 Transições

D.3.11.2.1 CANCELA_DESCARGA

D.3.11.2.1.1 Predicado

```
(DESCARREGANDO_AGV->fila==ERRO_PERM_AGV->i)
```

D.3.11.2.1.2 Acção

```
ulint pos=DESCARREGANDO_AGV->fila-XX->fila_entr_1+1;
ulint n=elem_1_ulint(pos, XX->p_num_pal_filas_entr);

XX->t=xx_gettime();

n++;
atribui_ulint_pos(n, pos, XX->p_num_pal_filas_entr);
xx_writesample(obtem_int(pos, XX->l_ve_est_filas_entr), n);

PALETE_AGUARD_PROCESS->dt_prox_oper=0.0;
PALETE_AGUARD_PROCESS->t_mud_est=XX->t;

OP_CD_INTERR->i=DESCARREGANDO_AGV->fila;

sprintf(XX->s, "\n%s: operacao de descarga cancelada na fila %u
(t=%.3f)\n",
    XX->nome, DESCARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);

xx_setstate(xx_getstate()+1);
xx_writesample(XX->ve_num_paletes_exist, xx_getstate());
```

D.3.11.2.2 CANCELA_DESCARGA_INTERR

D.3.11.2.2.1 Predicado

```
(OPERACAO_DESCARGA_INTERR->fila==ERRO_PERM_AGV->i)
```

D.3.11.2.2.2 Acção

```
ulint pos=OPERACAO_DESCARGA_INTERR->fila-XX->fila_entr_1+1;
ulint n=elem_1_ulint(pos, XX->p_num_pal_filas_entr);

XX->t=xx_gettime();

n++;
atribui_ulint_pos(n, pos, XX->p_num_pal_filas_entr);
xx_writesample(obtem_int(pos, XX->l_ve_est_filas_entr), n);

sprintf(XX->s, "\n%s: cancelamento da descarga na fila %u, apos %.3f s
da ocorr. de um erro (t=%.3f)\n",
    XX->nome, OPERACAO_DESCARGA_INTERR->fila,
    XX->t-OPERACAO_DESCARGA_INTERR->t_mud_est, XX->t);
prn_string(XX->s);

PALETE_AGUARD_PROCESS->t_mud_est=XX->t;
PALETE_AGUARD_PROCESS->dt_prox_oper=0.0;

OP_CD_INTERR->i=OPERACAO_DESCARGA_INTERR->fila;

xx_setstate(xx_getstate()+1);
xx_writesample(XX->ve_num_paletes_exist, xx_getstate());
```

D.3.11.2.3 INICIA_DESCARGA_AGV

D.3.11.2.3.1 Predicado

```
(INICIO_OPERACAO_DESCARGA->i>=XX->fila_entr_1 &&
INICIO_OPERACAO_DESCARGA->i<=XX->fila_entr_n
)
```

D.3.11.2.3.2 Acção

```
XX->t=xx_gettime();

DESCARREGANDO_AGV->fila=INICIO_OPERACAO_DESCARGA->i;
DESCARREGANDO_AGV->destino=0;
DESCARREGANDO_AGV->recurso=0;
DESCARREGANDO_AGV->dt_prox_oper=rnd_normal(
    XX->vm_tcd, XX->dp_tcd);
DESCARREGANDO_AGV->t_mud_est=XX->t;

TEMPO_CD_AGV->id=INICIO_OPERACAO_DESCARGA->i;
TEMPO_CD_AGV->t=DESCARREGANDO_AGV->dt_prox_oper;

/* xx_fdelay(XT_TERMINA_DESCARGA_AGV,
    DESCARREGANDO_AGV->dt_prox_oper);*/

sprintf(XX->s, "\n%s: inicio da descarga de um AGV na fila %u
(t=%.3f)\n",
    XX->nome, DESCARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  vai demorar %.3f s\n", DESCARREGANDO_AGV-
>dt_prox_oper);
prn_string(XX->s);
```

D.3.11.2.4 INTERROMPE_DESCARGA

D.3.11.2.4.1 Predicado

```
(DESCARREGANDO_AGV->fila==ERRO_TEMP_AGV->i)
```

D.3.11.2.4.2 Acção

```
XX->t=xx_gettime();

(OPERACAO_DESCARGA_INTERR->dt_prox_oper)-=XX->t-DESCARREGANDO_AGV-
>t_mud_est;
OPERACAO_DESCARGA_INTERR->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: operacao de descarga interrompida na fila %u
(t=%.3f)\n",
    XX->nome, DESCARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);
```

D.3.11.2.5 LIMPA_INICIO_OPERACAO_DESCARGA

D.3.11.2.6 RETOMA_DESCARGA

D.3.11.2.6.1 Predicado

```
(OPERACAO_DESCARGA_INTERR->fila==ERRO_TEMP_RESOLV->i)
```

D.3.11.2.6.2 Acção

```
XX->t=xx_gettime();

TEMPO_CD_AGV->id=OPERACAO_DESCARGA_INTERR->fila;
TEMPO_CD_AGV->t=OPERACAO_DESCARGA_INTERR->dt_prox_oper;

/* xx_fdelay(XT_TERMINA_DESCARGA_AGV,
    DESCARREGANDO_AGV->dt_prox_oper);*/

 sprintf(XX->s, "\n%s: fim da interrupcao da descarga na fila %u,
passados %.3f s (t=%.3f)\n",
    XX->nome, DESCARREGANDO_AGV->fila,
    XX->t-OPERACAO_DESCARGA_INTERR->t_mud_est, XX->t);
 prn_string(XX->s);
 sprintf(XX->s, "  a operacao de descarga vai durar mais %.3f s\n",
    DESCARREGANDO_AGV->dt_prox_oper);
 prn_string(XX->s);

 DESCARREGANDO_AGV->t_mud_est=XX->t;
```

D.3.11.2.7 TERMINA_DESCARGA_AGV

D.3.11.2.7.1 Predicado

```
(DESCARREGANDO_AGV->fila==FIM_OP_CD->i)
```

D.3.11.2.7.2 Acção

```
ulint pos=DESCARREGANDO_AGV->fila-XX->fila_entr_1+1;
ulint n=elem_1_ulint(pos, XX->p_num_pal_filas_entr);

XX->t=xx_gettime();

n++;
atribui_ulint_pos(n, pos, XX->p_num_pal_filas_entr);
xx_writesample(obtem_int(pos, XX->1_ve_est_filas_entr), n);

PALETE_AGUARD_PROCESS->dt_prox_oper=0.0;
PALETE_AGUARD_PROCESS->t_mud_est=XX->t;

FIM_OPERACAO_DESCARGA->i=DESCARREGANDO_AGV->fila;

sprintf(XX->s, "\n%s: fim da descarga de um AGV na fila %u (t=%.3f)\n",
   XX->nome, DESCARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);

xx_setstate(xx_getstate()+1);
xx_writesample(XX->ve_num_paletes_exist, xx_getstate());
```

D.3.12 Entidades da RP da página 'inicializa_modelo'

D.3.12.1 Posições

- CONTADOR_POS_FE:ULINT;
- CONTADOR_POS_FS:ULINT;
- CONT_FILA_ENTR:ULINT;
- INICIO_MODELO:NUL.

D.3.12.2 Transições

D.3.12.2.1 CRIA_POSICAO_LIVRE_FE

D.3.12.2.1.1 Predicado

```
(CONT_FILA_ENTR->i<=XX->fila_entr_n)
```

D.3.12.2.1.2 Acção

```
POSICOES_LIVRES_FILAS_ENTR->i=CONT_FILA_ENTR->i;
(CONT_FILA_ENTR->i)++;
```

D.3.12.2.2 CRIA_POSICOES_LIVRES_FS

D.3.12.2.2.1 Predicado

```
(CONTADOR_POS_FS->i<XX->capac_fila_saida)
```

D.3.12.2.2.2 Acção

```
(CONTADOR_POS_FS->i)++;
```

D.3.12.2.3 Acção de 'INICIA_CONTADOR_POS_FS'

```
CONTADOR_POS_FS->i=0;
```

D.3.12.2.4 Acção de 'INICIA_CONT_FE_E_CONT_POS'

```
CONT_FILA_ENTR->i=XX->fila_entr_1;
CONTADOR_POS_FE->i=1;
```

D.3.12.2.5 PREP_GER_OUTRA_POSICAO

D.3.12.2.5.1 Predicado

```
(CONTADOR_POS_FE->i<XX->capac_filas_entrada)
```

D.3.12.2.5.2 Acção

```
CONT_FILA_ENTR->i=XX->fila_entr_1;
(CONTADOR_POS_FE->i)++;
```

D.3.12.2.6 Acção de 'TERMINA_INICIAL_MODELO'

```
PONTO_SAIDA_DISPONIVEL->i=XX->fila_saida;
CAPACIDADE_FILA->fila=XX->fila_saida;
CAPACIDADE_FILA->cap=XX->capac_fila_saida;
```

D.3.13 Entidades da RP da página 'processamento_paletes'

D.3.13.1 Posições

- DOWNLOAD_PALETES:NUL;
- FILA_ENTR:ULINT;
- PALETE_AGUARD_COLOC_FS:PALETE;
- PALETE_SENDO_COLOC_FS:PALETE;
- PROCESSAMENTO:NUL;
- RECURSO_DISPONIVEL:NUL.

D.3.13.2 Transições

D.3.13.2.1 Acção de 'FIM_COLOC_PAL_FS'

```
XX->t=xx_gettime();

(XX->num_pal_fila_saida)++;
xx_writesample(XX->ve_est_fila_saida, XX->num_pal_fila_saida);

sprintf(XX->s, "\n%s: fim da colocacao da palete na fila de saida
(t=%.3f)\n",
    XX->nome, XX->t);
prn_string(XX->s);

PALETE_AGUARD_SAIDA->dt_prox_oper=0.0;
PALETE_AGUARD_SAIDA->t_mud_est=XX->t;

FILA_PALETE_INSERIDA->i=PALETE_SENDO_COLOC_FS->fila;
```

D.3.13.2.2 Acção de 'FIM_DOWNLOAD'

```
sprintf(XX->s, "\n%s: fim do download (t=%.3f)\n",
    XX->nome, xx_gettime());
prn_string(XX->s);

FILA_ENTR->i=XX->fila_entr_1;
```

D.3.13.2.3 Acção de 'FIM_PROCESSAMENTO'

```
XX->t=xx_gettime();

PALETE_AGUARD_COLOC_FS->fila=XX->fila_saida;
PALETE_AGUARD_COLOC_FS->dt_prox_oper=0.0;
PALETE_AGUARD_COLOC_FS->t_mud_est=XX->t;
```

```
 sprintf(XX->s, "\n%s: fim do processamento (t=%.3f)\n", XX->nome, XX-
>t);
 prn_string(XX->s);
 sprintf(XX->s, " a palete vai esperar por uma posicao livre na fila de
saida\n");
 prn_string(XX->s);
```

D.3.13.2.4 INICIA_DOWNLOAD_PALETES

D.3.13.2.4.1 Predicado

```
(ex_pal_todas_filas_entr(XX->p_num_pal_filas_entr))
```

D.3.13.2.4.2 Acção

```
double delay=rnd_normal(XX->vm_tdownload, XX->dp_tdownload);

xx_fdelay(XT_FIM_DOWNLOAD, delay);

sprintf(XX->s, "\n%s: inicio do download de uma palete de cada fila de
entrada (t=%.3f)\n",
    XX->nome, xx_gettime());
prn_string(XX->s);
sprintf(XX->s, "  vai demorar %.3f s\n", delay);
prn_string(XX->s);
```

D.3.13.2.5 Acção de 'INICIA_PROCESS'

```
double delay=rnd_normal(XX->vm_tprocess, XX->dp_tprocess);

xx_fdelay(XT_FIM_PROCESSAMENTO, delay);

sprintf(XX->s, "\n%s: inicio do processamento, que ira demorar %.3f s
(t=%.3f)\n",
    XX->nome, delay, xx_gettime());
prn_string(XX->s);

(XX->num_pal_ret_filas_entr)+=XX->fila_entr_n-XX->fila_entr_1+1;
xx_writesample(XX->ve_pal_ret_filas_entr, XX->num_pal_ret_filas_entr);

xx_setstate(xx_getstate()-XX->fila_entr_n+XX->fila_entr_1);
xx_writesample(XX->ve_num_paletes_exist, xx_getstate());
```

D.3.13.2.6 Acção de 'INICIO_COLOC_PAL_FS'

```
XX->t=xx_gettime();

sprintf(XX->s, "\n%s: a palete esperou %.3f s por uma posicao livre na
fila de saida\n",
    XX->nome, XX->t-PALETE_SENDO_COLOC_FS->t_mud_est);
prn_string(XX->s);

PALETE_SENDO_COLOC_FS->dt_prox_oper=
   rnd_normal(XX->vm_tdownload, XX->dp_tdownload);
PALETE_SENDO_COLOC_FS->t_mud_est=XX->t;

xx_fdelay(XT_FIM_COLOC_PAL_FS, PALETE_SENDO_COLOC_FS->dt_prox_oper);
```

```
 sprintf(XX->s, "  inicio da colocacao de uma palete na fila de saida
(t=%.3f)\n",
    XX->t);
 prn_string(XX->s);
 sprintf(XX->s, "  vai demorar %.3f s\n", PALETE_SENDO_COLOC_FS-
>dt_prox_oper);
 prn_string(XX->s);
```

D.3.13.2.7 RETIRA_PALETE_FILA_ENTR

D.3.13.2.7.1 Predicado

```
(FILA_ENTR->i<=XX->fila_entr_n &&
PALETE_AGUARD_PROCESS->fila==FILA_ENTR->i
)
```

D.3.13.2.7.2 Acção

```
ulint pos=FILA_ENTR->i-XX->fila_entr_1+1;
ulint n=elem_l_ulint(pos, XX->p_num_pal_filas_entr);

atribui_ulint_pos(--n, pos, XX->p_num_pal_filas_entr);
xx_writesample(obtem_int(pos, XX->l_ve_est_filas_entr), n);

sprintf(XX->s, "\n%s: vai ser retirada uma palete da fila %u\n",
   XX->nome, FILA_ENTR->i);
prn_string(XX->s);

POSICOES_LIVRES_FILAS_ENTR->i=FILA_ENTR->i;

(FILA_ENTR->i)++;
```

D.3.14 Entidades da RP da página 'saida_paletes'

D.3.14.1 Posições

- PAL_CARGA_AGV_INTERR:PALETE;
- PAL_SENDO_CARREG_AGV:PALETE.

D.3.14.2 Transições

D.3.14.2.1 CANCELA_CARGA_AGV

D.3.14.2.1.1 Predicado

```
(PAL_SENDO_CARREG_AGV->fila==ERRO_PERM_AGV->i)
```

D.3.14.2.1.2 Acção

```
XX->t=xx_gettime();

PONTO_SAIDA_DISPONIVEL->i=PAL_SENDO_CARREG_AGV->fila;
```

```
OP_CD_INTERR->i=PAL_SENDO_CARREG_AGV->fila;

sprintf(XX->s, "\n%s: operacao de carga cancelada na fila %u
(t=%.3f)\n",
    XX->nome, PAL_SENDO_CARREG_AGV->fila, XX->t);
prn_string(XX->s);
```

D.3.14.2.2 CANCELA_CARGA_AGV_INTERR

D.3.14.2.2.1 Predicado

```
(PAL_CARGA_AGV_INTERR->fila==ERRO_PERM_AGV->i)
```

D.3.14.2.2.2 Acção

```
XX->t=xx_gettime();

sprintf(XX->s, "\n%s: cancelamento da carga na fila %u, apos %.3f s da
ocorr. de um erro (t=%.3f)\n",
    XX->nome, PAL_CARGA_AGV_INTERR->fila,
    XX->t-PAL_CARGA_AGV_INTERR->t_mud_est, XX->t);
prn_string(XX->s);

PONTO_SAIDA_DISPONIVEL->i=PAL_CARGA_AGV_INTERR->fila;

OP_CD_INTERR->i=PAL_CARGA_AGV_INTERR->fila;
```

D.3.14.2.3 INICIA_CARGA_AGV

D.3.14.2.3.1 Predicado

```
(INICIO_OPERACAO_CARGA->i==XX->fila_saida)
```

D.3.14.2.3.2 Acção

```
XX->t=xx_gettime();

PAL_SENDO_CARREG_AGV->fila=INICIO_OPERACAO_CARGA->i;
PAL_SENDO_CARREG_AGV->destino=0;
PAL_SENDO_CARREG_AGV->recurso=0;
PAL_SENDO_CARREG_AGV->dt_prox_oper=rnd_normal(XX->vm_tcd, XX->dp_tcd);
PAL_SENDO_CARREG_AGV->t_mud_est=XX->t;

TEMPO_CD_AGV->id=INICIO_OPERACAO_CARGA->i;
TEMPO_CD_AGV->t=PAL_SENDO_CARREG_AGV->dt_prox_oper;

/* xx_fdelay(XT_TERMINA_CARGA_AGV, PAL_SENDO_CARREG_AGV-
>dt_prox_oper);*/

sprintf(XX->s, "\n%s: inicio da carga de um AGV na fila %u (t=%.3f)\n",
    XX->nome, PAL_SENDO_CARREG_AGV->fila, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  vai demorar %.3f s\n", PAL_SENDO_CARREG_AGV-
>dt_prox_oper);
prn_string(XX->s);
```

D.3.14.2.4 INTERR_CARGA_AGV

D.3.14.2.4.1 Predicado

```
(PAL_SENDO_CARREG_AGV->fila==ERRO_TEMP_AGV->i)
```

D.3.14.2.4.2 Acção

```
XX->t=xx_gettime();

(PAL_CARGA_AGV_INTERR->dt_prox_oper)-=XX->t-PAL_SENDO_CARREG_AGV-
>t_mud_est;
PAL_CARGA_AGV_INTERR->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: operacao de carga interrompida na fila %u
(t=%.3f)\n",
    XX->nome, PAL_SENDO_CARREG_AGV->fila, XX->t);
prn_string(XX->s);
```

D.3.14.2.5 LIMPA_INICIO_OPERACAO_CARGA

D.3.14.2.6 RETOMA_CARGA_AGV

D.3.14.2.6.1 Predicado

```
(PAL_CARGA_AGV_INTERR->fila==ERRO_TEMP_RESOLV->i)
```

D.3.14.2.6.2 Acção

```
XX->t=xx_gettime();

TEMPO_CD_AGV->id=PAL_CARGA_AGV_INTERR->fila;
TEMPO_CD_AGV->t=PAL_CARGA_AGV_INTERR->dt_prox_oper;

/* xx_fdelay(XT_TERMINA_CARGA_AGV,
    PAL_CARGA_AGV_INTERR->dt_prox_oper);*/

sprintf(XX->s, "\n%s: fim da interrupcao da carga na fila %u, passados
%.3f s (t=%.3f)\n",
    XX->nome, PAL_CARGA_AGV_INTERR->fila,
    XX->t-PAL_CARGA_AGV_INTERR->t_mud_est, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  a operacao de carga vai durar mais %.3f s\n",
    PAL_CARGA_AGV_INTERR->dt_prox_oper);
prn_string(XX->s);

PAL_SENDO_CARREG_AGV->t_mud_est=XX->t;
```

D.3.14.2.7 TERMINA_CARGA_AGV

D.3.14.2.7.1 Predicado

```
(PAL_SENDO_CARREG_AGV->fila==FIM_OP_CD->i)
```

D.3.14.2.7.2 Acção

```
XX->t=xx_gettime();
```

```
(XX->num_pal_fila_saida)--;
xx_writesample(XX->ve_est_fila_saida, XX->num_pal_fila_saida);
(XX->num_pal_ret_fila_saida)++;
xx_writesample(XX->ve_pal_ret_fila_saida, XX->num_pal_ret_fila_saida);

FIM_OPERACAO_CARGA->i=PAL_SENDO_CARREG_AGV->fila;
PONTO_SAIDA_DISPONIVEL->i=PAL_SENDO_CARREG_AGV->fila;
PAL_ACABA_SAIR_PT->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: fim da carga de um AGV na fila %u (t=%.3f)\n",
   XX->nome, PAL_SENDO_CARREG_AGV->fila, XX->t);
prn_string(XX->s);

xx_setstate(xx_getstate()-1);
xx_writesample(XX->ve_num_paletes_exist, xx_getstate());
```

D.4 Classe CENTRO_EXPEDICAO

D.4.1 CMN units incluídas

- ULINT;
- PALETE;
- TEMPORIZACAO.

D.4.2 Posições de entrada

- INICIO_OPERACAO_DESCARGA:ULINT);
- ERRO_TEMP_AGV:ULINT;
- ERRO_PERM_AGV:ULINT;
- ERRO_TEMP_RESOLV:ULINT.

D.4.3 Posições de saída

- FIM_OPERACAO_DESCARGA:ULINT;
- POSICOES_LIVRES_FILAS:ULINT.

D.4.4 Parâmetros

- capacidade_filas:unsigned long int – capacidade das filas de entrada;
- dp_tdescarga:double – desvio padrão do tempo de descarga de uma palete de um AGV;
- vm_tdescarga:double – valor médio do tempo de descarga de uma palete de um AGV;
- fila1:unsigned long int – identificador da 1ª fila de entrada;

- filan:unsigned long int – identificador da n-ésima fila de entrada;
- nome:xx_string – nome identificador da entidade;
- num_recursos:unsigned long int – número de recursos existentes para realizar a expedição de paletes;
- tam_lote_paletes:unsigned long int – tamanho do lote de paletes correspondente a cada carregamento;
- tm_entre_carreg:double – valor médio do tempo que medeia entre a realização de dois carregamentos de paletes consecutivos;
- tm_exped_pal:double – tempo médio de expedição de uma palete;
- dp_exped_pal:double – desvio padrão do tempo de expedição de uma palete;
- vm_tcarreg:double – valor médio do tempo de um carregamento;
- dp_tcarreg:double – desvio padrão do tempo de um carregamento.

D.4.5 Ficheiros de cabeçalho incluídos

```
#include "listas.h"
#include "mens_log.h"
#include "rand_fun.h"
```

D.4.6 Funções definidas internamente

```
/* funcao para permitir a visualizacao de um token do tipo
ULINT (utilizando o comando "examine Queue") */
void prn_ULINT_CENTRO_EXPEDICAO(a)
 ULINT *a;
{
  char s[255];

  sprintf(s, "(ulint) i=%u\n", a->i);
  prn_string(s);
}
```

D.4.7 Acções iniciais

```
ap_no_l_int u;
ap_no_l_ulint t;
unsigned long int i;

/* inicializa variaveis utilizadas para processamento de estatisticas */

sprintf(XX->s, "num_paletes_exist_%s", XX->nome);
XX->ve_num_paletes_exist=xx_initmeasure(XX->s);
xx_writesample(XX->ve_num_paletes_exist, 0);



/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o estado das filas e
uma lista com o numero de paletes existentes em cada fila */
XX->l_ve_estado_filas=NULL;
u=NULL;
```

```
XX->p_num_paletes_filas=NULL;
t=NULL;
for (i=XX->fila1;(i<=XX->filan || i==XX->fila1);i++)
{
  sprintf(XX->s, "estado_fila_%u", i);
  insere_int(xx_initmeasure(XX->s), &XX->l_ve_estado_filas, &u);
  xx_writesample(u->i, 0);
  insere_ulint(0, &XX->p_num_paletes_filas, &t, 1);
}

/* cria uma lista com os identificadores retornados pelas
declaracoes de medidas relacionadas com o numero de
paletes retiradas das filas e uma lista com o numero de
paletes retiradas de cada fila */
XX->l_ve_paletes_retiradas_filas=NULL;
u=NULL;
XX->p_num_paletes_retiradas_filas=NULL;
t=NULL;
for (i=XX->fila1;(i<=XX->filan || i==XX->fila1);i++)
{
  sprintf(XX->s, "paletes_retiradas_fila_%u", i);
  insere_int(xx_initmeasure(XX->s), &XX->l_ve_paletes_retiradas_filas,
&u);
  xx_writesample(u->i, 0);
  insere_ulint(0, &XX->p_num_paletes_retiradas_filas, &t, 1);
}


/* Examine Queue em posicoes de tipos nao geridos automatica/ */

 XW_ULINT_print = prn_ULINT_CENTRO_EXPEDICAO;


/* inicializa o estado do objecto */
/* (indica o WIP do Centro de Recepcao */

xx_setstate(0);
```

D.4.8 Variáveis locais

- l_ve_estado_filas:ap_no_l_int – lista de identificadores das variáveis estatísticas relacionadas com o estado das filas;
- l_ve_paletes_retiradas_filas:ap_no_l_int – lista de identificadores das variáveis estatísticas relacionadas com o número de paletes retiradas das filas;
- p_num_paletes_filas:ap_no_l_ulint – apontador para a cabeça da lista que guarda o número de paletes existentes em cada fila;
- p_num_paletes_retiradas_filas:ap_no_l_ulint – apontador para a cabeça da lista que guarda o número de paletes retiradas de cada fila;
- s:xx_string – *buffer* utilizado em várias partes do modelo;
- t:double – variável utilizada para guardar o tempo de simulação;
- ve_num_paletes_exist:int – identificador da variável estatística "número de paletes existentes".

D.4.9 Posições partilhadas

- RECURSOS_LIVRES:ULINT;
- ESPERA_NOVO_CARREG:NUL.

D.4.10 Entidades da RP da página principal

D.4.10.1 Objectos

- TEMP_DESCARGA_AGV:TEMPORIZADOR – temporização de operações de descarga de AGVs;
- TEMP_EXPEDICAO_PALETE:TEMPORIZADOR – temporização de operações de expedição de paletes.

D.4.10.2 Posições

- DESCARREGANDO_AGV:PALETE;
- EXPEDICAO_PERMITIDA:NUL;
- FIM_DESC_AGV:ULINT;
- FIM_TEMP_EXPED_PAL:ULINT;
- NOVO_CARREG_LOTE:NUL;
- NUM_PALETES_AGUARD_SAIDA:ULINT – o token da marcação inicial é inicializado através da instrução NUM_PALETES_AGUARD_SAIDA->i=0;
- OPERACAO_DESCARGA_INTERR:PALETE;
- OP_DESCARGA_INTERR:PALETE;
- PALETE_AGUARD_EXPEDICAO:PALETE;
- PALETE_SENDO_EXPEDIDA:PALETE;
- REALIZANDO_CARREG:NUL;
- TEMPO_DESC_AGV:TEMPORIZACAO;
- TEMPO_EXPED_PAL:TEMPORIZACAO.

D.4.10.3 Transições

D.4.10.3.1 CANCELA_DESCARGA

D.4.10.3.1.1 Predicado

```
(DESCARREGANDO_AGV->fila==ERRO_PERM_AGV->i)
```

D.4.10.3.1.2 Acção

```
ulint pos=DESCARREGANDO_AGV->fila-XX->fila1+1;
ulint n=elem_1_ulint(pos, XX->p_num_paletes_filas);
```

```
XX->t=xx_gettime();

n++;
atribui_ulint_pos(n, pos, XX->p_num_paletes_filas);
xx_writesample(obtem_int(pos, XX->l_ve_estado_filas), n);

PALETE_AGUARD_EXPEDICAO->dt_prox_oper=0.0;
PALETE_AGUARD_EXPEDICAO->t_mud_est=XX->t;

OP_DESCARGA_INTERR->i=DESCARREGANDO_AGV->fila;

sprintf(XX->s, "\n%s: operacao de descarga cancelada na fila %u
(t=%.3f)\n",
    XX->nome, DESCARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);

xx_setstate(xx_getstate()+1);
xx_writesample(XX->ve_num_paletes_exist, xx_getstate());
```

D.4.10.3.2 CANCELA_DESCARGA_INTERR

D.4.10.3.2.1 Predicado

```
(OPERACAO_DESCARGA_INTERR->fila==ERRO_PERM_AGV->i)
```

D.4.10.3.2.2 Acção

```
ulint pos=OPERACAO_DESCARGA_INTERR->fila-XX->fila1+1;
ulint n=elem_l_ulint(pos, XX->p_num_paletes_filas);

XX->t=xx_gettime();

n++;
atribui_ulint_pos(n, pos, XX->p_num_paletes_filas);
xx_writesample(obtem_int(pos, XX->l_ve_estado_filas), n);

sprintf(XX->s, "\n%s: cancelamento da descarga na fila %u, apos %.3f s
da ocorr. de um erro (t=%.3f)\n",
    XX->nome, OPERACAO_DESCARGA_INTERR->fila,
    XX->t-OPERACAO_DESCARGA_INTERR->t_mud_est, XX->t);
prn_string(XX->s);

PALETE_AGUARD_EXPEDICAO->t_mud_est=XX->t;
PALETE_AGUARD_EXPEDICAO->dt_prox_oper=0.0;

OP_DESCARGA_INTERR->i=OPERACAO_DESCARGA_INTERR->fila;


xx_setstate(xx_getstate()+1);
xx_writesample(XX->ve_num_paletes_exist, xx_getstate());
```

D.4.10.3.3 FIM_ESPERA_CARREG

D.4.10.3.4 INICIA_CARREG_LOTE

D.4.10.3.4.1 Predicado

```
(NUM_PALETES_AGUARD_SAIDA->i>=XX->tam_lote_paletes)
```

D.4.10.3.4.2 Acção

```
double delay=rnd_normal(XX->vm_tcarreg, XX->dp_tcarreg);

(NUM_PALETES_AGUARD_SAIDA->i)-=XX->tam_lote_paletes;

xx_fdelay(XT_TERMINA_CARREG_LOTE, delay);

XX->t=xx_gettime();

sprintf(XX->s, "\n%s: inicio do carregamento de um lote de paletes
(t=%.3f)\n",
    XX->nome, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  vai demorar %.3f s\n", delay);
prn_string(XX->s);
```

D.4.10.3.5 INICIA_DESCARGA_AGV

D.4.10.3.5.1 Predicado

```
(INICIO_OPERACAO_DESCARGA->i>=XX->fila1 &&
INICIO_OPERACAO_DESCARGA->i<=XX->filan
)
```

D.4.10.3.5.2 Acção

```
XX->t=xx_gettime();

DESCARREGANDO_AGV->fila=INICIO_OPERACAO_DESCARGA->i;
DESCARREGANDO_AGV->destino=0;
DESCARREGANDO_AGV->recurso=0;
DESCARREGANDO_AGV->dt_prox_oper=rnd_normal(
    XX->vm_tdescarga, XX->dp_tdescarga);
DESCARREGANDO_AGV->t_mud_est=XX->t;

TEMPO_DESC_AGV->id=INICIO_OPERACAO_DESCARGA->i;
TEMPO_DESC_AGV->t=DESCARREGANDO_AGV->dt_prox_oper;

/* xx_fdelay(XT_TERMINA_DESCARGA_AGV,
    DESCARREGANDO_AGV->dt_prox_oper);*/

sprintf(XX->s, "\n%s: inicio da descarga de um AGV na fila %u
(t=%.3f)\n",
    XX->nome, DESCARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  vai demorar %.3f s\n", DESCARREGANDO_AGV-
>dt_prox_oper);
```

```
prn_string(XX->s);
```

D.4.10.3.6 Acção de 'INICIA_EXPEDICAO_PALETE'

```
double delay=rnd_normal(XX->tm_exped_pal, XX->dp_exped_pal);
ulint pos=PALETE_AGUARD_EXPEDICAO->fila-XX->fila1+1;
ulint n_ins=elem_l_ulint(pos, XX->p_num_paletes_filas);
ulint n_ret=elem_l_ulint(pos, XX->p_num_paletes_retiradas_filas);

XX->t=xx_gettime();

POSICOES_LIVRES_FILAS->i=PALETE_AGUARD_EXPEDICAO->fila;

PALETE_SENDO_EXPEDIDA->recurso=RECURSOS_LIVRES->i;
PALETE_SENDO_EXPEDIDA->t_mud_est=XX->t;

n_ins--;
atribui_ulint_pos(n_ins, pos, XX->p_num_paletes_filas);
xx_writesample(obtem_int(pos, XX->l_ve_estado_filas), n_ins);
n_ret++;
atribui_ulint_pos(n_ret, pos, XX->p_num_paletes_retiradas_filas);
xx_writesample(obtem_int(pos, XX->l_ve_paletes_retiradas_filas),
n_ret);

TEMPO_EXPED_PAL->id=RECURSOS_LIVRES->i;
TEMPO_EXPED_PAL->t=delay;

/* xx_fdelay(XT_TERMINA_EXPEDICAO_PALETE, delay);*/

sprintf(XX->s, "\n%s: inicio da expedicao de uma palete da fila %u
(t=%.3f)\n",
    XX->nome, PALETE_AGUARD_EXPEDICAO->fila, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  vai demorar %.3f s\n", delay);
prn_string(XX->s);
```

D.4.10.3.7 INTERROMPE_DESCARGA

D.4.10.3.7.1 Predicado

```
(DESCARREGANDO_AGV->fila==ERRO_TEMP_AGV->i)
```

D.4.10.3.7.2 Acção

```
XX->t=xx_gettime();

(OPERACAO_DESCARGA_INTERR->dt_prox_oper)-=XX->t-DESCARREGANDO_AGV-
>t_mud_est;
OPERACAO_DESCARGA_INTERR->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: operacao de descarga interrompida na fila %u
(t=%.3f)\n",
    XX->nome, DESCARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);
```

D.4.10.3.8 Predicado de 'LIMPA_ERRO_PERM_AGV0'

```
(ERRO_PERM_AGV->i<XX->fila1 || ERRO_PERM_AGV->i>XX->filan)
```

D.4.10.3.9 Predicado de 'LIMPA_ERRO_TEMP_AGV'

```
(ERRO_TEMP_AGV->i<XX->fila1 || ERRO_TEMP_AGV->i>XX->filan)
```

D.4.10.3.10 Predicado de 'LIMPA_ERRO_TEMP_RESOLV'

```
(ERRO_TEMP_RESOLV->i<XX->fila1 || ERRO_TEMP_RESOLV->i>XX->filan)
```

D.4.10.3.11 LIMPA_INICIO_OPERACAO_DESCARGA

D.4.10.3.12 RETOMA_DESCARGA

D.4.10.3.12.1 Predicado

```
(OPERACAO_DESCARGA_INTERR->fila==ERRO_TEMP_RESOLV->i)
```

D.4.10.3.12.2 Acção

```
XX->t=xx_gettime();

TEMPO_DESC_AGV->id=OPERACAO_DESCARGA_INTERR->fila;
TEMPO_DESC_AGV->t=OPERACAO_DESCARGA_INTERR->dt_prox_oper;

/* xx_fdelay(XT_TERMINA_DESCARGA_AGV,
    DESCARREGANDO_AGV->dt_prox_oper);*/

 sprintf(XX->s, "\n%s: fim da interrupcao da descarga na fila %u,
passados %.3f s (t=%.3f)\n",
    XX->nome, DESCARREGANDO_AGV->fila,
    XX->t-OPERACAO_DESCARGA_INTERR->t_mud_est, XX->t);
 prn_string(XX->s);
 sprintf(XX->s, "  a operacao de descarga vai durar mais %.3f s\n",
    DESCARREGANDO_AGV->dt_prox_oper);
 prn_string(XX->s);

 DESCARREGANDO_AGV->t_mud_est=XX->t;
```

D.4.10.3.13 Acção de 'TERMINA_CARREG_LOTE'

```
double delay=rnd_exp_neg(XX->tm_entre_carreg);

xx_fdelay(XT_FIM_ESPERA_CARREG, delay);

XX->t=xx_gettime();

sprintf(XX->s, "\n%s: fim do carregamento de um lote (t=%.3f)\n",
    XX->nome, XX->t);
prn_string(XX->s);
sprintf(XX->s, "  o proximo carreg. podera ter lugar dentro de %.3f
s\n", delay);
prn_string(XX->s);

xx_setstate(xx_getstate()-XX->tam_lote_paletes);
xx_writesample(XX->ve_num_paletes_exist, xx_getstate());
```

D.4.10.3.14 TERMINA_DESCARGA_AGV

D.4.10.3.14.1 Predicado

```
(DESCARREGANDO_AGV->fila==FIM_DESC_AGV->i)
```

D.4.10.3.14.2 Acção

```
ulint pos=DESCARREGANDO_AGV->fila-XX->fila1+1;
ulint n=elem_1_ulint(pos, XX->p_num_paletes_filas);

XX->t=xx_gettime();

n++;
atribui_ulint_pos(n, pos, XX->p_num_paletes_filas);
xx_writesample(obtem_int(pos, XX->l_ve_estado_filas), n);

PALETE_AGUARD_EXPEDICAO->dt_prox_oper=0.0;
PALETE_AGUARD_EXPEDICAO->t_mud_est=XX->t;

FIM_OPERACAO_DESCARGA->i=DESCARREGANDO_AGV->fila;

sprintf(XX->s, "\n%s: fim da descarga de um AGV na fila %u (t=%.3f)\n",
   XX->nome, DESCARREGANDO_AGV->fila, XX->t);
prn_string(XX->s);

xx_setstate(xx_getstate()+1);
xx_writesample(XX->ve_num_paletes_exist, xx_getstate());
```

D.4.10.3.15 TERMINA_EXPEDICAO_PALETE

D.4.10.3.15.1 Predicado

```
(PALETE_SENDO_EXPEDIDA->recurso==FIM_TEMP_EXPED_PAL->i)
```

D.4.10.3.15.2 Acção

```
RECURSOS_LIVRES->i=PALETE_SENDO_EXPEDIDA->recurso;

(NUM_PALETES_AGUARD_SAIDA->i)++;

sprintf(XX->s, "\n%s: fim da expedicao de uma palete da fila %u
(t=%.3f)\n",
   XX->nome, PALETE_SENDO_EXPEDIDA->fila, xx_gettime());
prn_string(XX->s);
```

D.4.11 Entidades da RP da página 'inicializa_modelo'

D.4.11.1 Posições

- CONTADOR_FILA:ULINT;
- CONTADOR_POS:ULINT;
- CONTADOR_RECURSOS:ULINT;
- INICIO_MODELO:NUL.

D.4.11.2 Transições

D.4.11.2.1 Acção de 'ARRANCA_MODELO'

```
double delay=rnd_exp_neg(XX->tm_entre_carreg);

xx_fdelay(XT_FIM_ESPERA_CARREG, delay);

sprintf(XX->s, "\n%s: o proximo carreg. podera ter lugar dentro de %.3f
s\n",
    XX->nome, delay);
prn_string(XX->s);
```

D.4.11.2.2 CRIA_POSICAO_LIVRE

D.4.11.2.2.1 Predicado

```
(CONTADOR_FILA->i<=XX->filan)
```

D.4.11.2.2.2 Acção

```
POSICOES_LIVRES_FILAS->i=CONTADOR_FILA->i;
(CONTADOR_FILA->i)++;
```

D.4.11.2.3 CRIA_RECURSO_LIVRE

D.4.11.2.3.1 Predicado

```
(CONTADOR_RECURSOS->i<=XX->num_recursos)
```

D.4.11.2.3.2 Acção

```
RECURSOS_LIVRES->i=CONTADOR_RECURSOS->i;
(CONTADOR_RECURSOS->i)++;
```

D.4.11.2.4 Acção de 'INICIA_CONT_FILA_E_CONT_POS'

```
CONTADOR_FILA->i=XX->fila1;
CONTADOR_POS->i=1;
```

D.4.11.2.5 Acção de 'INICIA_CRIACAO_RECURSOS'

```
CONTADOR_RECURSOS->i=1;
```

D.4.11.2.6 PREP_GER_OUTRA_POSICAO

D.4.11.2.6.1 Predicado

```
(CONTADOR_POS->i<XX->capacidade_filas)
```

D.4.11.2.6.2 Acção

```
CONTADOR_FILA->i=XX->fila1;
(CONTADOR_POS->i)++;
```



D.5 Classe SISTEMA_MANUFACTURA

D.5.1 CMN units incluídas

- ULINT;
- PALETE;
- ORDEM;
- CAPAC_FILA.

D.5.2 Posições de entrada

- INICIO_OPERACAO_CARGA:ULINT;
- INICIO_OPERACAO_DESCARGA:ULINT;
- ERRO_TEMP_AGV:ULINT;
- ERRO_TEMP_RESOLV:ULINT;
- ERRO_PERM_AGV:ULINT;
- PONTO_SAIDA_DISPONIVEL:ULINT.

D.5.3 Posições de saída

- FIM_OPERACAO_CARGA:ULINT;
- FIM_OPERACAO_DESCARGA:ULINT;
- ORDEM_MOVIM:ORDEM;
- FILA_PALETE_INSERIDA:ULINT;
- CAPACIDADE_FILA:CAPAC_FILA.

D.5.4 Parâmetros

- p_CRMP_AAMP_CM1:double – probabilidade de ser escolhido o AAMP como destino de uma palete do CRMP, quando for necessário optar entre o AAMP e o CM1;
- nome:xx_string – identificador da entidade representada pelo objecto;
- p_AAMP_CM1_CM2:double – probabilidade de ser escolhido o CM1 como destino de uma palete do AAMP, quando for necessário optar entre o CM1 e o CM2;
- t_entre_ped_AAMP:double – tempo médio entre 2 pedidos consecutivos de saída de palete do AAMP;
- p_CM1_PM1_PM2:double – probabilidade de ser escolhido o PM1 como destino de uma palete do CM1, quando for necessário optar entre o PM1 e o PM2;

- p_CM2_PM3_PM4:double – probabilidade de ser escolhido o PM3 como destino de uma palete do CM2, quando for necessário optar entre o PM3 e o PM4;
- p_PM1_PM3_CM3:double – probabilidade de ser escolhido o PM3 como destino de uma palete do PM1, quando for necessário optar entre o PM3 e o CM3;
- p_PM2_PM4_CM3:double – probabilidade de ser escolhido o PM4 como destino de uma palete do PM2, quando for necessário optar entre o PM4 e o CM3;
- p_PM3_PM5_PM6:double – probabilidade de ser escolhido o PM5 como destino de uma palete do PM3, quando for necessário optar entre o PM5 e o PM6;
- p_PM4_PM5_PM6:double – probabilidade de ser escolhido o PM5 como destino de uma palete do PM4, quando for necessário optar entre o PM5 e o PM6;
- p_PM5_CM4_CM:double – probabilidade de ser escolhido o CM4 como destino de uma palete do PM5, quando for necessário optar entre o CM4 e a CM;
- p_PM6_CM4_CM:double – probabilidade de ser escolhido o CM4 como destino de uma palete do PM6, quando for necessário optar entre o CM4 e a CM;
- p_CM3_PM5_PM6:double – probabilidade de ser escolhido o PM5 como destino de uma palete do CM3, quando for necessário optar entre o PM5 e o PM6;
- p_CM_CQ_AAPA:double – probabilidade de ser escolhido o CQ como destino de uma palete da CM, quando for necessário optar entre o CQ e o AAPA;
- p_CQ_AAPA_CEPA:double – probabilidade de ser escolhido o AAPA como destino de uma palete do CQ, quando for necessário optar entre o AAPA e o CEPA;
- t_entre_ped_AAPA:double – tempo médio entre 2 pedidos consecutivos de saída de palete do AAPA;
- p_AAPA_CQ_CEPA:double – probabilidade de ser escolhido o CQ como destino de uma palete do AAPA, quando for necessário optar entre o CQ e o CEPA.

D.5.5 Ficheiros de cabeçalho incluídos

```
#include "rand_fun.h"
```

D.5.6 Funções definidas internamente

```
/* funcao para permitir a visualizacao de um token do tipo
ULINT (utilizando o comando "examine Queue") */
void prn_ULINT_SIST_MANUF(a)
 ULINT *a;
{
  char s[255];

  sprintf(s, "(ulint) i=%u\n", a->i);
  prn_string(s);
}
```

D.5.7 Acções iniciais

```
/* Examine Queue em posicoes de tipos nao geridos automatica/ */
```

```
XW_ULINT_print = prn_ULINT_SIST_MANUF;


/* configura os parametros dos sub-objectos */
 if (xx_readparameters("sismanuf.ini")==-1)
  printf("\nErro na leitura dos parametros do Sistema de fabrico\n");

 xx_fdelay(XT_PERMITE_NOVO_PEDIDO_AAMP,
    rnd_exp_neg(XX->t_entre_ped_AAMP));
 xx_fdelay(XT_PERMITE_NOVO_PEDIDO_AAPA,
    rnd_exp_neg(XX->t_entre_ped_AAPA));
```

D.5.8 Variáveis locais

- s:xx_string – *buffer* utilizado em várias partes do modelo;
- t:double – variável utilizada para guardar o tempo de simulacao.

D.5.9 Objectos partilhados

- RECEPCAO_MAT_PRIMAS:CENTRO_RECEPCAO – Centro de Recepção de Matérias-Primas;
- ARMAZEM_MAT_PRIMAS:ARMAZEM_AUTOMATICO – Armazém Automático de Matérias-Primas;
- CENTRO_MAQ_1:POSTO_TRABALHO – Centro de Maquinação 1 com 1 ponto de entrada (e 1 ponto de saída);
- CENTRO_MAQ_2:POSTO_TRABALHO – Centro de Maquinação 2 com 1 ponto de entrada (e 1 ponto de saída);
- POSTO_TRAB_MAN_1:POSTO_TRABALHO – Posto de Trabalho Manual 1, com 1 ponto de entrada (e 1 ponto de saída);
- POSTO_TRAB_MAN_2:POSTO_TRABALHO – Posto de Trabalho Manual 2, com 1 ponto de entrada (e 1 ponto de saída);
- POSTO_TRAB_MAN_3:POSTO_TRABALHO – Posto de Trabalho Manual 3, com 1 ponto de entrada (e 1 ponto de saída);
- POSTO_TRAB_MAN_4:POSTO_TRABALHO – Posto de Trabalho Manual 4, com 1 ponto de entrada (e 1 ponto de saída);
- POSTO_TRAB_MAN_5:POSTO_TRABALHO – Posto de Trabalho Manual 5, com 1 ponto de entrada (e 1 ponto de saída);
- POSTO_TRAB_MAN_6:POSTO_TRABALHO – Posto de Trabalho Manual 6, com 1 ponto de entrada (e 1 ponto de saída);
- CENTRO_MAQ_3:POSTO_TRABALHO – Centro de Maquinação 3 com 1 ponto de entrada (e 1 ponto de saída);
- CENTRO_MAQ_4:POSTO_TRABALHO – Centro de Maquinacao 4 com 2 pontos de entrada (e 1 ponto de saída);
- CELULA_MONTAGEM:POSTO_TRABALHO – Célula de Montagem com 3 pontos de entrada (e 1 ponto de saída);

- CONTR_QUALIDADE:POSTO_TRABALHO – Centro de Controlo de Qualidade com 1 ponto de entrada (e 1 ponto de saída);
- ARMAZEM_PROD_ACABADO:ARMAZEM_AUTOMATICO – Armazém Automático de Produto Acabado;
- EXPEDICAO_PROD_ACABADO:CENTRO_EXPEDICAO – Centro de Expedição de Produto Acabado com 2 pontos de entrada.

D.5.10 Posições partilhadas

- FILAS_SAIDA_CRMP:PALETE;
- POS_LIVRES_FE_AAMP:ULINT;
- POS_LIVRES_FE_CM1:ULINT;
- POS_LIVRES_FE_CM2:ULINT;
- POS_LIVRE_A_REPOR_CM1:ULINT;
- PAL_ARMAZ_AAMP:PALETE;
- POS_LIVRES_FS_AAMP:ULINT;
- PEDIDO_SAIDA_AAMP:PALETE;
- PAL_ACABA_SAIR_AAMP:PALETE;
- PAL_RETIDAS_SAIDA_AAMP:PALETE;
- FILA_SAIDA_CM1:PALETE;
- POS_LIVRES_FE_PM1:ULINT;
- POS_LIVRES_FE_PM2:ULINT;
- FILA_SAIDA_CM2:PALETE;
- POS_LIVRES_FE_PM3:ULINT;
- POS_LIVRES_FE_PM4:ULINT;
- POS_LIVRE_A_REPOR_PM3:ULINT;
- POS_LIVRE_A_REPOR_PM4:ULINT;
- POS_LIVRES_FE_CM3:ULINT;
- FILA_SAIDA_PM1:PALETE;
- POS_LIVRE_A_REPOR_CM3:ULINT;
- FILA_SAIDA_PM2:PALETE;
- POS_LIVRES_FE_PM5:ULINT;
- POS_LIVRES_FE_PM6:ULINT;
- FILA_SAIDA_PM3:PALETE;
- POS_LIVRE_A_REPOR_PM5:ULINT;
- POS_LIVRE_A_REPOR_PM6:ULINT;

- FILA_SAIDA_PM4:PALETE;
- POS_LIVRES_FE_CM4:ULINT;
- POS_LIVRES_FE_CM:ULINT;
- POS_LIVRE_A_REPOR_CM4:ULINT;
- POS_LIVRE_A_REPOR_CM:ULINT;
- FILA_SAIDA_PM5:PALETE;
- FILA_SAIDA_PM6:PALETE;
- FILA_SAIDA_CM3:PALETE;
- FILA_SAIDA_CM4:PALETE;
- POS_LIVRE_A_REPOR_CQ:ULINT;
- POS_LIVRES_FE_CQ:ULINT;
- POS_LIVRES_FE_AAPA:ULINT;
- POS_LIVRE_A_REPOR_AAPA:ULINT;
- FILA_SAIDA_CM:PALETE;
- POS_LIVRES_FE_CEPA:ULINT;
- POS_LIVRE_A_REPOR_CEPA:ULINT;
- FILA_SAIDA_CQ:PALETE;
- PAL_ARMAZ_AAPA:PALETE;
- POS_LIVRES_FS_AAPA:ULINT;
- PEDIDO_SAIDA_AAPA:PALETE;
- PAL_ACABA_SAIR_AAPA:PALETE;
- PAL_RETIDAS_SAIDA_AAPA:PALETE;
- PAL_ACABA_SAIR_CRMP:PALETE;
- PAL_ACABA_SAIR_CM1:PALETE;
- PAL_ACABA_SAIR_CM2:PALETE;
- PAL_ACABA_SAIR_PM1:PALETE;
- PAL_ACABA_SAIR_PM2:PALETE;
- PAL_ACABA_SAIR_PM3:PALETE;
- PAL_ACABA_SAIR_PM4:PALETE;
- PAL_ACABA_SAIR_PM5:PALETE;
- PAL_ACABA_SAIR_PM6:PALETE;
- PAL_ACABA_SAIR_CM3:PALETE;
- PAL_ACABA_SAIR_CM4:PALETE;
- PAL_ACABA_SAIR_CM:PALETE;
- PAL_ACABA_SAIR_CQ:PALETE.

D.5.11 Entidades da RP da página 'encam_pal_AAMP_CM1_ou_CM2'

D.5.11.1 Posições

- DESTINO_PROX_PAL_AAMP:ULINT;
- ESPERA_NOVO_PEDIDO_AAMP:NUL;
- INICIA_DEC_ENC_PAL_AAMP:NUL;
- NOVO_PEDIDO_AAMP:PALETE;
- ORDEM_PENDENTE_AAMP:ORDEM;
- PAL_NOVO_PEDIDO_AAMP:PALETE;
- POS_LIVRE_A_REPOR_CM2:ULINT.

D.5.11.2 Transições

D.5.11.2.1 Acção de 'DECIDE_AAMP_CM1'

```
DESTINO_PROX_PAL_AAMP->i=POS_LIVRES_FE_CM1->i;
```

D.5.11.2.2 Acção de 'DECIDE_AAMP_CM1_OU_CM2'

```
if (random()<XX->p_AAMP_CM1_CM2)
{
  DESTINO_PROX_PAL_AAMP->i=POS_LIVRES_FE_CM1->i;
  POS_LIVRE_A_REPOR_CM1->i=0;
  POS_LIVRE_A_REPOR_CM2->i=POS_LIVRES_FE_CM2->i;
}
else
{
  DESTINO_PROX_PAL_AAMP->i=POS_LIVRES_FE_CM2->i;
  POS_LIVRE_A_REPOR_CM2->i=0;
  POS_LIVRE_A_REPOR_CM1->i=POS_LIVRES_FE_CM1->i;
}
```

D.5.11.2.3 Acção de 'DECIDE_AAMP_CM2'

```
DESTINO_PROX_PAL_AAMP->i=POS_LIVRES_FE_CM2->i;
```

D.5.11.2.4 Acção de 'DECIDE_DESTINO_PAL_AAMP'

```
NOVO_PEDIDO_AAMP->destino=DESTINO_PROX_PAL_AAMP->i;
```

D.5.11.2.5 ENVIA_ORDEM_PAL_AAMP

D.5.11.2.5.1 Predicado

```
(ORDEM_PENDENTE_AAMP->origem==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.11.2.5.2 Acção

```
sprintf(XX->s, "\n%s: envio da ordem de movimento %u->%u (t=%.3f)\n",
```

```
    XX->nome, ORDEM_PENDENTE_AAMP->origem,
    ORDEM_PENDENTE_AAMP->destino, xx_gettime());
prn_string(XX->s);
```

D.5.11.2.6 ENVIA_ORDEM_PAL_RET_AAMP

D.5.11.2.6.1 Predicado

```
(PAL_RETIDAS_SAIDA_AAMP->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.11.2.6.2 Acção

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_AAMP->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_AAMP->destino;

sprintf(XX->s, "\n%s: envio da ordem de movimento %u->%u (t=%.3f)\n",
    XX->nome, PAL_RETIDAS_SAIDA_AAMP->fila,
    PAL_RETIDAS_SAIDA_AAMP->destino, xx_gettime());
prn_string(XX->s);
```

D.5.11.2.7 Acção de 'FAZ_PEDIDO_AAMP'

```
double t=xx_gettime();

PEDIDO_SAIDA_AAMP->fila=POS_LIVRES_FS_AAMP->i;
PEDIDO_SAIDA_AAMP->t_mud_est=t;

ORDEM_PENDENTE_AAMP->origem=NOVO_PEDIDO_AAMP->fila;
ORDEM_PENDENTE_AAMP->destino=NOVO_PEDIDO_AAMP->destino;

sprintf(XX->s, "\n%s: pedido de saida de uma palete para a fila %u
(t=%.3f)\n",
    XX->nome, PEDIDO_SAIDA_AAMP->fila, t);
prn_string(XX->s);
```

D.5.11.2.8 Predicado de 'LIMPA_PAL_RETIDAS_SAIDA_AAMP'

```
(PAL_ACABA_SAIR_AAMP->fila==PAL_RETIDAS_SAIDA_AAMP->fila)
```

D.5.11.2.9 LIMPA_POS_LIVRE_A_REPOR_CM2

D.5.11.2.10 Acção de 'PERMITE_NOVO_PEDIDO_AAMP'

```
PAL_NOVO_PEDIDO_AAMP->t_mud_est=xx_gettime();

xx_fdelay(XT_PERMITE_NOVO_PEDIDO_AAMP,
    rnd_exp_neg(XX->t_entre_ped_AAMP));
```

D.5.11.2.11 Predicado de 'REPOE_POS_LIVRE_CM2'

```
(POS_LIVRE_A_REPOR_CM2->i>0)
```

D.5.12 Entidades da RP da página 'encam_pal_AAPA_CQ_CEPA'

D.5.12.1 Posições

- DESTINO_PROX_PAL_AAPA:ULINT;
- ESPERA_NOVO_PEDIDO_AAPA:NUL;
- INICIA_DEC_ENC_PAL_AAPA:NUL;
- NOVO_PEDIDO_AAPA:PALETE;
- ORDEM_PENDENTE_AAPA:ORDEM;
- PAL_NOVO_PEDIDO_AAPA:PALETE.

D.5.12.2 Transições

D.5.12.2.1 Acção de 'DECIDE_AAPA_CEPA'

```
DESTINO_PROX_PAL_AAPA->i=POS_LIVRES_FE_CEPA->i;
```

D.5.12.2.2 Acção de 'DECIDE_AAPA_CQ'

```
DESTINO_PROX_PAL_AAPA->i=POS_LIVRES_FE_CQ->i;
```

D.5.12.2.3 Acção de 'DECIDE_AAPA_CQ_OU_CEPA'

```
if (random()<XX->p_AAPA_CQ_CEPA)
{
  DESTINO_PROX_PAL_AAPA->i=POS_LIVRES_FE_CQ->i;
  POS_LIVRE_A_REPOR_CQ->i=0;
  POS_LIVRE_A_REPOR_CEPA->i=POS_LIVRES_FE_CEPA->i;
}
else
{
  DESTINO_PROX_PAL_AAPA->i=POS_LIVRES_FE_CEPA->i;
  POS_LIVRE_A_REPOR_CEPA->i=0;
  POS_LIVRE_A_REPOR_CQ->i=POS_LIVRES_FE_CQ->i;
}
```

D.5.12.2.4 Acção de 'DECIDE_DESTINO_PAL_AAPA'

```
NOVO_PEDIDO_AAPA->destino=DESTINO_PROX_PAL_AAPA->i;
```

D.5.12.2.5 ENVIA_ORDEM_PAL_AAPA

D.5.12.2.5.1 Predicado

```
(ORDEM_PENDENTE_AAPA->origem==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.12.2.5.2 Acção

```
sprintf(XX->s, "\n%s: envio da ordem de movimento %u->%u (t=%.3f)\n",
    XX->nome, ORDEM_PENDENTE_AAPA->origem,
    ORDEM_PENDENTE_AAPA->destino, xx_gettime());
```

```
prn_string(XX->s);
```

D.5.12.2.6 ENVIA_ORDEM_PAL_RET_AAPA

D.5.12.2.6.1 Predicado

```
(PAL_RETIDAS_SAIDA_AAPA->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.12.2.6.2 Acção

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_AAPA->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_AAPA->destino;

sprintf(XX->s, "\n%s: envio da ordem de movimento %u->%u (t=%.3f)\n",
    XX->nome, PAL_RETIDAS_SAIDA_AAPA->fila,
    PAL_RETIDAS_SAIDA_AAPA->destino, xx_gettime());
prn_string(XX->s);
```

D.5.12.2.7 Acção de 'FAZ_PEDIDO_AAPA'

```
double t=xx_gettime();

PEDIDO_SAIDA_AAPA->fila=POS_LIVRES_FS_AAPA->i;
PEDIDO_SAIDA_AAPA->t_mud_est=t;

ORDEM_PENDENTE_AAPA->origem=NOVO_PEDIDO_AAPA->fila;
ORDEM_PENDENTE_AAPA->destino=NOVO_PEDIDO_AAPA->destino;

sprintf(XX->s, "\n%s: pedido de saida de uma palete para a fila %u
(t=%.3f)\n",
    XX->nome, PEDIDO_SAIDA_AAPA->fila, t);
prn_string(XX->s);
```

D.5.12.2.8 Predicado de 'LIMPA_PAL_RETIDAS_SAIDA_AAPA'

```
(PAL_ACABA_SAIR_AAPA->fila==PAL_RETIDAS_SAIDA_AAPA->fila)
```

D.5.12.2.9 Acção de 'PERMITE_NOVO_PEDIDO_AAPA'

```
PAL_NOVO_PEDIDO_AAPA->t_mud_est=xx_gettime();

xx_fdelay(XT_PERMITE_NOVO_PEDIDO_AAPA,
    rnd_exp_neg(XX->t_entre_ped_AAPA));
```

D.5.13 Entidades da RP da página 'encam_pal_CM1_PM1_PM2'

D.5.13.1 Posições

- DESTINO_PROX_PAL_CM1:ULINT;
- INICIA_DEC_ENC_PAL_CM1:NUL;
- PAL_RETIDAS_SAIDA_CM1:PALETE;
- POS_LIVRE_A_REPOR_PM1:ULINT;
- POS_LIVRE_A_REPOR_PM2:ULINT.

D.5.13.2 Transições

D.5.13.2.1 Acção de 'DECIDE_CM1_PM1'

```
DESTINO_PROX_PAL_CM1->i=POS_LIVRES_FE_PM1->i;
```

D.5.13.2.2 Acção de 'DECIDE_CM1_PM1_OU_PM2'

```
if (random()<XX->p_CM1_PM1_PM2)
{
  DESTINO_PROX_PAL_CM1->i=POS_LIVRES_FE_PM1->i;
  POS_LIVRE_A_REPOR_PM1->i=0;
  POS_LIVRE_A_REPOR_PM2->i=POS_LIVRES_FE_PM2->i;
}
else
{
  DESTINO_PROX_PAL_CM1->i=POS_LIVRES_FE_PM2->i;
  POS_LIVRE_A_REPOR_PM2->i=0;
  POS_LIVRE_A_REPOR_PM1->i=POS_LIVRES_FE_PM1->i;
}
```

D.5.13.2.3 Acção de 'DECIDE_CM1_PM2'

```
DESTINO_PROX_PAL_CM1->i=POS_LIVRES_FE_PM2->i;
```

D.5.13.2.4 ENVIA_ORDEM_PAL_RET_SAIDA_CM1

D.5.13.2.4.1 Predicado

```
(PAL_RETIDAS_SAIDA_CM1->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.13.2.4.2 Acção

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_CM1->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_CM1->destino;

sprintf(XX->s, "\n%s: envio da ordem %u->%u, para uma palete retida no
CM1 (t=%.3f)\n",
    XX->nome, ORDEM_MOVIM->origem, ORDEM_MOVIM->destino, xx_gettime());
prn_string(XX->s);
```

D.5.13.2.5 ENVIA_ORDEM_SAIDA_CM1

D.5.13.2.5.1 Predicado

```
(FILA_SAIDA_CM1->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.13.2.5.2 Acção

```
XX->t=xx_gettime();

ORDEM_MOVIM->origem=FILA_SAIDA_CM1->fila;
ORDEM_MOVIM->destino=DESTINO_PROX_PAL_CM1->i;

PAL_RETIDAS_SAIDA_CM1->destino=DESTINO_PROX_PAL_CM1->i;
PAL_RETIDAS_SAIDA_CM1->t_mud_est=XX->t;
```

```
sprintf(XX->s, "\n%s: envio da ordem %u->%u (t=%.3f)\n",
   XX->nome, FILA_SAIDA_CM1->fila,
   DESTINO_PROX_PAL_CM1->i, XX->t);
prn_string(XX->s);
```

D.5.13.2.6 Predicado de 'LIMPA_PAL_RETIDA_SAIDA_CM1'

```
(PAL_RETIDAS_SAIDA_CM1->fila==PAL_ACABA_SAIR_CM1->fila)
```

D.5.13.2.7 LIMPA_POS_LIVRE_A_REPOR_PM1

D.5.13.2.8 LIMPA_POS_LIVRE_A_REPOR_PM2

D.5.13.2.9 Predicado de 'REPOE_POS_LIVRE_PM1'

```
(POS_LIVRE_A_REPOR_PM1->i>0)
```

D.5.13.2.10 Predicado de 'REPOE_POS_LIVRE_PM2'

```
(POS_LIVRE_A_REPOR_PM2->i>0)
```

D.5.14 Entidades da RP da página 'encam_pal_CM2_PM3_PM4'

D.5.14.1 Posições

- DESTINO_PROX_PAL_CM2:ULINT;
- INICIA_DEC_ENC_PAL_CM2:NUL;
- PAL_RETIDAS_SAIDA_CM2:PALETE.

D.5.14.2 Transições

D.5.14.2.1 Acção de 'DECIDE_CM2_PM3'

```
DESTINO_PROX_PAL_CM2->i=POS_LIVRES_FE_PM3->i;
```

D.5.14.2.2 Acção de 'DECIDE_CM2_PM3_OU_PM4'

```
if (random()<XX->p_CM2_PM3_PM4)
{
  DESTINO_PROX_PAL_CM2->i=POS_LIVRES_FE_PM3->i;
  POS_LIVRE_A_REPOR_PM3->i=0;
  POS_LIVRE_A_REPOR_PM4->i=POS_LIVRES_FE_PM4->i;
}
else
{
  DESTINO_PROX_PAL_CM2->i=POS_LIVRES_FE_PM4->i;
  POS_LIVRE_A_REPOR_PM4->i=0;
  POS_LIVRE_A_REPOR_PM3->i=POS_LIVRES_FE_PM3->i;
}
```

D.5.14.2.3 Acção de 'DECIDE_CM2_PM4'

```
DESTINO_PROX_PAL_CM2->i=POS_LIVRES_FE_PM4->i;
```

D.5.14.2.4 ENVIA_ORDEM_PAL_RET_SAIDA_CM2

D.5.14.2.4.1 Predicado

```
(PAL_RETIDAS_SAIDA_CM2->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.14.2.4.2 Acção

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_CM2->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_CM2->destino;

sprintf(XX->s, "\n%s: envio da ordem %u->%u, para uma palete retida no
CM2 (t=%.3f)\n",
    XX->nome, ORDEM_MOVIM->origem, ORDEM_MOVIM->destino, xx_gettime());
prn_string(XX->s);
```

D.5.14.2.5 ENVIA_ORDEM_SAIDA_CM2

D.5.14.2.5.1 Predicado

```
(FILA_SAIDA_CM2->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.14.2.5.2 Acção

```
XX->t=xx_gettime();

ORDEM_MOVIM->origem=FILA_SAIDA_CM2->fila;
ORDEM_MOVIM->destino=DESTINO_PROX_PAL_CM2->i;

PAL_RETIDAS_SAIDA_CM2->destino=DESTINO_PROX_PAL_CM2->i;
PAL_RETIDAS_SAIDA_CM2->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: envio da ordem %u->%u (t=%.3f)\n",
    XX->nome, FILA_SAIDA_CM2->fila,
    DESTINO_PROX_PAL_CM2->i, XX->t);
prn_string(XX->s);
```

D.5.14.2.6 Predicado de 'LIMPA_PAL_RETIDA_SAIDA_CM2'

```
(PAL_RETIDAS_SAIDA_CM2->fila==PAL_ACABA_SAIR_CM2->fila)
```

D.5.14.2.7 LIMPA_POS_LIVRE_A_REPOR_PM3

D.5.14.2.8 LIMPA_POS_LIVRE_A_REPOR_PM4

D.5.14.2.9 Predicado de 'REPOE_POS_LIVRE_PM3'

```
(POS_LIVRE_A_REPOR_PM3->i>0)
```

D.5.14.2.10 Predicado de 'REPOE_POS_LIVRE_PM4'

```
(POS_LIVRE_A_REPOR_PM4->i>0)
```

D.5.15 Entidades da RP da página 'encam_pal_CM4_CM'

D.5.15.1 Posições

- DESTINO_PROX_PAL_CM4:ULINT;
- INICIA_DEC_ENC_PAL_CM4:NUL;
- PAL_RETIDAS_SAIDA_CM4:PALETE.

D.5.15.2 Transições

D.5.15.2.1 Acção de 'DECIDE_CM4_CM'

```
DESTINO_PROX_PAL_CM4->i=POS_LIVRES_FE_CM->i;
```

D.5.15.2.2 ENVIA_ORDEM_PAL_RET_SAIDA_CM4

D.5.15.2.2.1 Predicado

```
(PAL_RETIDAS_SAIDA_CM4->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.15.2.2.2 Acção

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_CM4->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_CM4->destino;

sprintf(XX->s, "\n%s: envio da ordem %u->%u, para uma palete retida no
CM4 (t=%.3f)\n",
    XX->nome, ORDEM_MOVIM->origem, ORDEM_MOVIM->destino, xx_gettime());
prn_string(XX->s);
```

D.5.15.2.3 ENVIA_ORDEM_SAIDA_CM4

D.5.15.2.3.1 Predicado

```
(FILA_SAIDA_CM4->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.15.2.3.2 Acção

```
XX->t=xx_gettime();

ORDEM_MOVIM->origem=FILA_SAIDA_CM4->fila;
ORDEM_MOVIM->destino=DESTINO_PROX_PAL_CM4->i;

PAL_RETIDAS_SAIDA_CM4->destino=DESTINO_PROX_PAL_CM4->i;
PAL_RETIDAS_SAIDA_CM4->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: envio da ordem %u->%u (t=%.3f)\n",
   XX->nome, FILA_SAIDA_CM4->fila,
   DESTINO_PROX_PAL_CM4->i, XX->t);
```

```
prn_string(XX->s);
```

D.5.15.2.4 Predicado de 'LIMPA_PAL_RETIDA_SAIDA_CM4'

```
(PAL_RETIDAS_SAIDA_CM4->fila==PAL_ACABA_SAIR_CM4->fila)
```

D.5.16 Entidades da RP da página 'encam_pal_CM_CQ_AAPA'

D.5.16.1 Posições

- DESTINO_PROX_PAL_CM:ULINT;
- INICIA_DEC_ENC_PAL_CQ_AAPA:NUL;
- PAL_RETIDAS_SAIDA_CM:PALETE.

D.5.16.2 Transições

D.5.16.2.1 Acção de 'DECIDE_CM_AAPA'

```
DESTINO_PROX_PAL_CM->i=POS_LIVRES_FE_AAPA->i;
```

D.5.16.2.2 Acção de 'DECIDE_CM_CQ'

```
DESTINO_PROX_PAL_CM->i=POS_LIVRES_FE_CQ->i;
```

D.5.16.2.3 Acção de 'DECIDE_CM_CQ_OU_AAPA'

```
if (random()<XX->p_CM_CQ_AAPA)
{
  DESTINO_PROX_PAL_CM->i=POS_LIVRES_FE_CQ->i;
  POS_LIVRE_A_REPOR_CQ->i=0;
  POS_LIVRE_A_REPOR_AAPA->i=POS_LIVRES_FE_AAPA->i;
}
else
{
  DESTINO_PROX_PAL_CM->i=POS_LIVRES_FE_AAPA->i;
  POS_LIVRE_A_REPOR_AAPA->i=0;
  POS_LIVRE_A_REPOR_CQ->i=POS_LIVRES_FE_CQ->i;
}
```

D.5.16.2.4 ENVIA_ORDEM_PAL_RET_SAIDA_CM

D.5.16.2.4.1 Predicado

```
(PAL_RETIDAS_SAIDA_CM->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.16.2.4.2 Acção

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_CM->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_CM->destino;

sprintf(XX->s, "\n%s: envio da ordem %u->%u, para uma palete retida na
CM (t=%.3f)\n",
    XX->nome, ORDEM_MOVIM->origem, ORDEM_MOVIM->destino, xx_gettime());
```

```
prn_string(XX->s);
```

D.5.16.2.5 ENVIA_ORDEM_SAIDA_CM

D.5.16.2.5.1 Predicado

```
(FILA_SAIDA_CM->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.16.2.5.2 Acção

```
XX->t=xx_gettime();

ORDEM_MOVIM->origem=FILA_SAIDA_CM->fila;
ORDEM_MOVIM->destino=DESTINO_PROX_PAL_CM->i;

PAL_RETIDAS_SAIDA_CM->destino=DESTINO_PROX_PAL_CM->i;
PAL_RETIDAS_SAIDA_CM->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: envio da ordem %u->%u (t=%.3f)\n",
    XX->nome, FILA_SAIDA_CM->fila,
    DESTINO_PROX_PAL_CM->i, XX->t);
prn_string(XX->s);
```

D.5.16.2.6 Predicado de 'LIMPA_PAL_RETIDA_SAIDA_CM'

```
(PAL_RETIDAS_SAIDA_CM->fila==PAL_ACABA_SAIR_CM->fila)
```

D.5.16.2.7 LIMPA_POS_LIVRE_A_REPOR_AAPA

D.5.16.2.8 LIMPA_POS_LIVRE_A_REPOR_CQ

D.5.16.2.9 Predicado de 'REPOE_POS_LIVRE_AAPA'

```
(POS_LIVRE_A_REPOR_AAPA->i>0)
```

D.5.16.2.10 Predicado de 'REPOE_POS_LIVRE_CQ'

```
(POS_LIVRE_A_REPOR_CQ->i>0)
```

D.5.17 Entidades de RP da página 'encam_pal_CQ_AAPA_CEPA'

D.5.17.1 Posições

- DESTINO_PROX_PAL_CQ:ULINT;
- INICIA_DEC_ENC_PAL_CQ:NUL;
- PAL_RETIDAS_SAIDA_CQ:PALETE.

D.5.17.2 Transições

D.5.17.2.1 Acção de DECIDE_CQ_AAPA

```
DESTINO_PROX_PAL_CQ->i=POS_LIVRES_FE_AAPA->i;
```

D.5.17.2.2 Acção de 'DECIDE_CQ_AAPA_OU_CEPA'

```
if (random()<XX->p_CQ_AAPA_CEPA)
{
  DESTINO_PROX_PAL_CQ->i=POS_LIVRES_FE_AAPA->i;
  POS_LIVRE_A_REPOR_AAPA->i=0;
  POS_LIVRE_A_REPOR_CEPA->i=POS_LIVRES_FE_CEPA->i;
}
else
{
  DESTINO_PROX_PAL_CQ->i=POS_LIVRES_FE_CEPA->i;
  POS_LIVRE_A_REPOR_CEPA->i=0;
  POS_LIVRE_A_REPOR_AAPA->i=POS_LIVRES_FE_AAPA->i;
}
```

D.5.17.2.3 Acção de 'DECIDE_CQ_CEPA'

```
DESTINO_PROX_PAL_CQ->i=POS_LIVRES_FE_CEPA->i;
```

D.5.17.2.4 ENVIA_ORDEM_PAL_RET_SAIDA_CQ

D.5.17.2.4.1 Predicado

```
(PAL_RETIDAS_SAIDA_CQ->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.17.2.4.2 Acção

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_CQ->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_CQ->destino;

sprintf(XX->s, "\n%s: envio da ordem %u->%u, para uma palete retida no
CQ (t=%.3f)\n",
    XX->nome, ORDEM_MOVIM->origem, ORDEM_MOVIM->destino, xx_gettime());
prn_string(XX->s);
```

D.5.17.2.5 ENVIA_ORDEM_SAIDA_CQ

D.5.17.2.5.1 Predicado

```
(FILA_SAIDA_CQ->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.17.2.5.2 Acção

```
XX->t=xx_gettime();

ORDEM_MOVIM->origem=FILA_SAIDA_CQ->fila;
ORDEM_MOVIM->destino=DESTINO_PROX_PAL_CQ->i;

PAL_RETIDAS_SAIDA_CQ->destino=DESTINO_PROX_PAL_CQ->i;
PAL_RETIDAS_SAIDA_CQ->t_mud_est=XX->t;
```

```
sprintf(XX->s, "\n%s: envio da ordem %u->%u (t=%.3f)\n",
   XX->nome, FILA_SAIDA_CQ->fila,
   DESTINO_PROX_PAL_CQ->i, XX->t);
prn_string(XX->s);
```

D.5.17.2.6 Predicado de 'LIMPA_PAL_RETIDA_SAIDA_CQ'

```
(PAL_RETIDAS_SAIDA_CQ->fila==PAL_ACABA_SAIR_CQ->fila)
```

D.5.17.2.7 LIMPA_POS_LIVRE_A_REPOR_CEPA

D.5.17.2.8 Predicado de 'REPOE_POS_LIVRE_CEPA'

```
(POS_LIVRE_A_REPOR_CEPA->i>0)
```

D.5.18 Entidades da RP da página 'encam_pal_CRMP_AAMP_ou_CM1'

D.5.18.1 Posições

- DESTINO_PROX_PAL_CRMP:ULINT;
- INICIA_DEC_ENC_PAL_CRMP:NUL;
- PAL_RETIDAS_SAIDA_CRMP:PALETE;
- POS_LIVRE_A_REPOR_AAMP:ULINT.

D.5.18.2 Transições

D.5.18.2.1 Acção de 'DECIDE_CRMP_AAMP'

```
DESTINO_PROX_PAL_CRMP->i=POS_LIVRES_FE_AAMP->i;
```

D.5.18.2.2 Acção de 'DECIDE_CRMP_AAMP_OU_CM2'

```
if (random()<XX->p_CRMP_AAMP_CM1)
{
  DESTINO_PROX_PAL_CRMP->i=POS_LIVRES_FE_AAMP->i;
  POS_LIVRE_A_REPOR_AAMP->i=0;
  POS_LIVRE_A_REPOR_CM1->i=POS_LIVRES_FE_CM1->i;
}
else
{
  DESTINO_PROX_PAL_CRMP->i=POS_LIVRES_FE_CM1->i;
  POS_LIVRE_A_REPOR_CM1->i=0;
  POS_LIVRE_A_REPOR_AAMP->i=POS_LIVRES_FE_AAMP->i;
}
```

D.5.18.2.3 Acção de 'DECIDE_CRMP_CM1'

```
DESTINO_PROX_PAL_CRMP->i=POS_LIVRES_FE_CM1->i;
```

D.5.18.2.4 ENVIA_ORDEM_PAL_RET_SAIDA_CRMP

D.5.18.2.4.1 Predicado

```
(PAL_RETIDAS_SAIDA_CRMP->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.18.2.4.2 Acção

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_CRMP->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_CRMP->destino;

sprintf(XX->s, "\n%s: envio da ordem %u->%u, para uma palete retida no
CRMP (t=%.3f)\n",
    XX->nome, ORDEM_MOVIM->origem, ORDEM_MOVIM->destino, xx_gettime());
prn_string(XX->s);
```

D.5.18.2.5 ENVIA_ORDEM_SAIDA_CRMP

D.5.18.2.5.1 Predicado

```
(FILAS_SAIDA_CRMP->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.18.2.5.2 Acção

```
XX->t=xx_gettime();

ORDEM_MOVIM->origem=FILAS_SAIDA_CRMP->fila;
ORDEM_MOVIM->destino=DESTINO_PROX_PAL_CRMP->i;

PAL_RETIDAS_SAIDA_CRMP->destino=DESTINO_PROX_PAL_CRMP->i;
PAL_RETIDAS_SAIDA_CRMP->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: envio da ordem %u->%u (t=%.3f)\n",
    XX->nome, FILAS_SAIDA_CRMP->fila,
    DESTINO_PROX_PAL_CRMP->i, XX->t);
prn_string(XX->s);
```

D.5.18.2.6 Predicado de 'LIMPA_PAL_RETIDA_SAIDA_CRMP'

```
(PAL_RETIDAS_SAIDA_CRMP->fila==PAL_ACABA_SAIR_CRMP->fila)
```

D.5.18.2.7 LIMPA_POS_LIVRE_A_REPOR_AAMP

D.5.18.2.8 LIMPA_POS_LIVRE_A_REPOR_CM1

D.5.18.2.9 Predicado de 'REPOE_POS_LIVRE_AAMP'

```
(POS_LIVRE_A_REPOR_AAMP->i>0)
```

D.5.18.2.10 Predicado de 'REPOE_POS_LIVRE_CM1'

```
(POS_LIVRE_A_REPOR_CM1->i>0)
```

D.5.19 Entidades da RP da página 'encam_pal_PM1_PM3_CM3'

D.5.19.1 Posições

- DESTINO_PROX_PAL_PM1:ULINT;
- INICIA_DEC_ENC_PAL_PM1:NUL;
- PAL_RETIDAS_SAIDA_PM1:PALETE.

D.5.19.2 Transições

D.5.19.2.1 Acção de 'DECIDE_PM1_CM3'

```
DESTINO_PROX_PAL_PM1->i=POS_LIVRES_FE_CM3->i;
```

D.5.19.2.2 Acção de 'DECIDE_PM1_PM3'

```
DESTINO_PROX_PAL_PM1->i=POS_LIVRES_FE_PM3->i;
```

D.5.19.2.3 Acção de 'DECIDE_PM1_PM3_OU_CM3'

```
if (random()<XX->p_PM1_PM3_CM3)
{
  DESTINO_PROX_PAL_PM1->i=POS_LIVRES_FE_PM3->i;
  POS_LIVRE_A_REPOR_PM3->i=0;
  POS_LIVRE_A_REPOR_CM3->i=POS_LIVRES_FE_CM3->i;
}
else
{
  DESTINO_PROX_PAL_PM1->i=POS_LIVRES_FE_CM3->i;
  POS_LIVRE_A_REPOR_CM3->i=0;
  POS_LIVRE_A_REPOR_PM3->i=POS_LIVRES_FE_PM3->i;
}
```

D.5.19.2.4 ENVIA_ORDEM_PAL_RET_SAIDA_PM1

D.5.19.2.4.1 Predicado

```
(PAL_RETIDAS_SAIDA_PM1->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.19.2.4.2 Acção

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_PM1->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_PM1->destino;

sprintf(XX->s, "\n%s: envio da ordem %u->%u, para uma palete retida no
PM1 (t=%.3f)\n",
    XX->nome, ORDEM_MOVIM->origem, ORDEM_MOVIM->destino, xx_gettime());
prn_string(XX->s);
```

D.5.19.2.5 ENVIA_ORDEM_SAIDA_PM1

D.5.19.2.5.1 Predicado

```
(FILA_SAIDA_PM1->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.19.2.5.2 Acção

```
XX->t=xx_gettime();

ORDEM_MOVIM->origem=FILA_SAIDA_PM1->fila;
ORDEM_MOVIM->destino=DESTINO_PROX_PAL_PM1->i;

PAL_RETIDAS_SAIDA_PM1->destino=DESTINO_PROX_PAL_PM1->i;
PAL_RETIDAS_SAIDA_PM1->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: envio da ordem %u->%u (t=%.3f)\n",
   XX->nome, FILA_SAIDA_PM1->fila,
   DESTINO_PROX_PAL_PM1->i, XX->t);
prn_string(XX->s);
```

D.5.19.2.6 Predicado de 'LIMPA_PAL_RETIDA_SAIDA_PM1'

```
(PAL_RETIDAS_SAIDA_PM1->fila==PAL_ACABA_SAIR_PM1->fila)
```

D.5.19.2.7 LIMPA_POS_LIVRE_A_REPOR_CM3

D.5.19.2.8 Predicado de 'REPOE_POS_LIVRE_CM3'

```
(POS_LIVRE_A_REPOR_CM3->i>0)
```

D.5.20 Entidades da RP da página 'encam_pal_PM2_PM4_CM3'

D.5.20.1 Posições

- DESTINO_PROX_PAL_PM2:ULINT;
- INICIA_DEC_ENCAM_PAL_PM2:NUL;
- PAL_RETIDAS_SAIDA_PM2:PALETE.

D.5.20.2 Transições

D.5.20.2.1 Acção de 'DECIDE_PM2_CM3'

```
DESTINO_PROX_PAL_PM2->i=POS_LIVRES_FE_CM3->i;
```

D.5.20.2.2 Acção de 'DECIDE_PM2_PM4'

```
DESTINO_PROX_PAL_PM2->i=POS_LIVRES_FE_PM4->i;
```

D.5.20.2.3 Acção de 'DECIDE_PM2_PM4_OU_CM3'

```
if (random()<XX->p_PM2_PM4_CM3)
```

```
{
  DESTINO_PROX_PAL_PM2->i=POS_LIVRES_FE_PM4->i;
  POS_LIVRE_A_REPOR_PM4->i=0;
  POS_LIVRE_A_REPOR_CM3->i=POS_LIVRES_FE_CM3->i;
}
else
{
  DESTINO_PROX_PAL_PM2->i=POS_LIVRES_FE_CM3->i;
  POS_LIVRE_A_REPOR_CM3->i=0;
  POS_LIVRE_A_REPOR_PM4->i=POS_LIVRES_FE_PM4->i;
}
```

D.5.20.2.4 ENVIA_ORDEM_PAL_RET_SAIDA_PM2

D.5.20.2.4.1 Predicado

```
(PAL_RETIDAS_SAIDA_PM2->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.20.2.4.2 Acção

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_PM2->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_PM2->destino;

sprintf(XX->s, "\n%s: envio da ordem %u->%u, para uma palete retida no
PM2 (t=%.3f)\n",
    XX->nome, ORDEM_MOVIM->origem, ORDEM_MOVIM->destino, xx_gettime());
prn_string(XX->s);
```

D.5.20.2.5 ENVIA_ORDEM_SAIDA_PM2

D.5.20.2.5.1 Predicado

```
(FILA_SAIDA_PM2->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.20.2.5.2 Acção

```
XX->t=xx_gettime();

ORDEM_MOVIM->origem=FILA_SAIDA_PM2->fila;
ORDEM_MOVIM->destino=DESTINO_PROX_PAL_PM2->i;

PAL_RETIDAS_SAIDA_PM2->destino=DESTINO_PROX_PAL_PM2->i;
PAL_RETIDAS_SAIDA_PM2->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: envio da ordem %u->%u (t=%.3f)\n",
    XX->nome, FILA_SAIDA_PM2->fila,
    DESTINO_PROX_PAL_PM2->i, XX->t);
prn_string(XX->s);
```

D.5.20.2.6 Predicado de 'LIMPA_PAL_RETIDA_SAIDA_PM2'

```
(PAL_RETIDAS_SAIDA_PM2->fila==PAL_ACABA_SAIR_PM2->fila)
```

D.5.21 Entidades da RP da página 'encam_pal_PM3_PM5_PM6'

D.5.21.1 Posições

- DESTINO_PROX_PAL_PM3:ULINT;
- INICIA_DEC_ENC_PAL_PM3:NUL;
- PAL_RETIDAS_SAIDA_PM3:PALETE.

D.5.21.2 Transições

D.5.21.2.1 Acção de 'DECIDE_PM3_PM5'

```
DESTINO_PROX_PAL_PM3->i=POS_LIVRES_FE_PM5->i;
```

D.5.21.2.2 Acção de 'DECIDE_PM3_PM5_OU_PM6'

```
if (random()<XX->p_PM3_PM5_PM6)
{
  DESTINO_PROX_PAL_PM3->i=POS_LIVRES_FE_PM5->i;
  POS_LIVRE_A_REPOR_PM5->i=0;
  POS_LIVRE_A_REPOR_PM6->i=POS_LIVRES_FE_PM6->i;
}
else
{
  DESTINO_PROX_PAL_PM3->i=POS_LIVRES_FE_PM6->i;
  POS_LIVRE_A_REPOR_PM6->i=0;
  POS_LIVRE_A_REPOR_PM5->i=POS_LIVRES_FE_PM5->i;
}
```

D.5.21.2.3Acção de 'DECIDE_PM3_PM6'

```
DESTINO_PROX_PAL_PM3->i=POS_LIVRES_FE_PM6->i;
```

D.5.21.2.4 ENVIA_ORDEM_PAL_RET_SAIDA_PM3

D.5.21.2.4.1 Predicado

```
(PAL_RETIDAS_SAIDA_PM3->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.21.2.4.2 Acção

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_PM3->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_PM3->destino;

sprintf(XX->s, "\n%s: envio da ordem %u->%u, para uma palete retida no
PM3 (t=%.3f)\n",
    XX->nome, ORDEM_MOVIM->origem, ORDEM_MOVIM->destino, xx_gettime());
prn_string(XX->s);
```

D.5.21.2.5 ENVIA_ORDEM_SAIDA_PM3

D.5.21.2.4.1 Predicado

```
(FILA_SAIDA_PM3->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.21.2.4.2 Acção

```
XX->t=xx_gettime();

ORDEM_MOVIM->origem=FILA_SAIDA_PM3->fila;
ORDEM_MOVIM->destino=DESTINO_PROX_PAL_PM3->i;

PAL_RETIDAS_SAIDA_PM3->destino=DESTINO_PROX_PAL_PM3->i;
PAL_RETIDAS_SAIDA_PM3->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: envio da ordem %u->%u (t=%.3f)\n",
   XX->nome, FILA_SAIDA_PM3->fila,
   DESTINO_PROX_PAL_PM3->i, XX->t);
prn_string(XX->s);
```

D.5.21.2.6 Predicado de 'LIMPA_PAL_RETIDAS_SAIDA_PM3'

```
(PAL_RETIDAS_SAIDA_PM3->fila==PAL_ACABA_SAIR_PM3->fila)
```

D.5.21.2.7 LIMPA_POS_LIVRE_A_REPOR_PM5

D.5.21.2.8 LIMPA_POS_LIVRE_A_REPOR_PM6

D.5.21.2.9 Predicado de 'REPOE_POS_LIVRE_PM5'

```
(POS_LIVRE_A_REPOR_PM5->i>0)
```

D.5.21.2.10 Predicado de 'REPOE_POS_LIVRE_PM6'

```
(POS_LIVRE_A_REPOR_PM6->i>0)
```

D.5.22 Entidades da RP da página 'encam_pal_PM4_PM5_PM6'

D.5.22.1 Posições

- DESTINO_PROX_PAL_PM4:ULINT;
- INICIA_DEC_ENC_PAL_PM4:NUL;
- PAL_RETIDAS_SAIDA_PM4:PALETE.

D.5.22.2 Transições

D.5.22.2.1 Acção de 'DECIDE_PM4_PM5'

```
DESTINO_PROX_PAL_PM4->i=POS_LIVRES_FE_PM5->i;
```

D.5.22.2.2 Acção de 'DECIDE_PM4_PM5_OU_PM6'

```
if (random()<XX->p_PM4_PM5_PM6)
{
  DESTINO_PROX_PAL_PM4->i=POS_LIVRES_FE_PM5->i;
  POS_LIVRE_A_REPOR_PM5->i=0;
  POS_LIVRE_A_REPOR_PM6->i=POS_LIVRES_FE_PM6->i;
}
else
{
  DESTINO_PROX_PAL_PM4->i=POS_LIVRES_FE_PM6->i;
  POS_LIVRE_A_REPOR_PM6->i=0;
  POS_LIVRE_A_REPOR_PM5->i=POS_LIVRES_FE_PM5->i;
}
```

D.5.22.2.3 Acção de 'DECIDE_PM4_PM6'

```
DESTINO_PROX_PAL_PM4->i=POS_LIVRES_FE_PM6->i;
```

D.5.22.2.4 ENVIA_ORDEM_PAL_RET_SAIDA_PM4

D.5.22.2.4.1 Predicado

```
(PAL_RETIDAS_SAIDA_PM4->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.22.2.4.2 Acção

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_PM4->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_PM4->destino;

sprintf(XX->s, "\n%s: envio da ordem %u->%u, para uma palete retida no
PM4 (t=%.3f)\n",
    XX->nome, ORDEM_MOVIM->origem, ORDEM_MOVIM->destino, xx_gettime());
prn_string(XX->s);
```

D.5.22.2.5 ENVIA_ORDEM_SAIDA_PM4

D.5.22.2.5.1 Predicado

```
(FILA_SAIDA_PM4->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.22.2.5.2 Acção

```
XX->t=xx_gettime();

ORDEM_MOVIM->origem=FILA_SAIDA_PM4->fila;
ORDEM_MOVIM->destino=DESTINO_PROX_PAL_PM4->i;

PAL_RETIDAS_SAIDA_PM4->destino=DESTINO_PROX_PAL_PM4->i;
PAL_RETIDAS_SAIDA_PM4->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: envio da ordem %u->%u (t=%.3f)\n",
    XX->nome, FILA_SAIDA_PM4->fila,
    DESTINO_PROX_PAL_PM4->i, XX->t);
prn_string(XX->s);
```

D.5.22.2.6 Predicado de 'LIMPA_PAL_RETIDA_SAIDA_PM4'

```
(PAL_RETIDAS_SAIDA_PM4->fila==PAL_ACABA_SAIR_PM4->fila)
```

D.5.23 Entidades da RP da página 'encam_pal_PM5_CM4_CM'

D.5.23.1 Posições

- DESTINO_PROX_PAL_PM5:ULINT;
- INICIA_DEC_ENC_PAL_PM5:NUL;
- PAL_RETIDAS_SAIDA_PM5:PALETE.

D.5.23.2 Transições

D.5.23.2.1 Acção de 'DECIDE_PM5_CM'

```
DESTINO_PROX_PAL_PM5->i=POS_LIVRES_FE_CM->i;
```

D.5.23.2.2 Acção de 'DECIDE_PM5_CM4'

```
DESTINO_PROX_PAL_PM5->i=POS_LIVRES_FE_CM4->i;
```

D.5.23.2.3 Acção de 'DECIDE_PM5_CM4_OU_CM'

```
if (random()<XX->p_PM5_CM4_CM)
{
  DESTINO_PROX_PAL_PM5->i=POS_LIVRES_FE_CM4->i;
  POS_LIVRE_A_REPOR_CM4->i=0;
  POS_LIVRE_A_REPOR_CM->i=POS_LIVRES_FE_CM->i;
}
else
{
  DESTINO_PROX_PAL_PM5->i=POS_LIVRES_FE_CM->i;
  POS_LIVRE_A_REPOR_CM->i=0;
  POS_LIVRE_A_REPOR_CM4->i=POS_LIVRES_FE_CM4->i;
}
```

D.5.23.2.4 ENVIA_ORDEM_PAL_RET_SAIDA_PM5

D.5.23.2.4.1 Predicado

```
(PAL_RETIDAS_SAIDA_PM5->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.23.2.4.2 Acção

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_PM5->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_PM5->destino;

sprintf(XX->s, "\n%s: envio da ordem %u->%u, para uma palete retida no
PM5 (t=%.3f)\n",
    XX->nome, ORDEM_MOVIM->origem, ORDEM_MOVIM->destino, xx_gettime());
prn_string(XX->s);
```

D.5.23.2.5 Transition ENVIA_ORDEM_SAIDA_PM5

D.5.23.2.5.1 Predicado

```
(FILA_SAIDA_PM5->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.23.2.5.2 Acção

```
XX->t=xx_gettime();

ORDEM_MOVIM->origem=FILA_SAIDA_PM5->fila;
ORDEM_MOVIM->destino=DESTINO_PROX_PAL_PM5->i;

PAL_RETIDAS_SAIDA_PM5->~estino=DESTINO_PROX_PAL_PM5-
PAL_RETIDAS_SAIDA_PM5->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: envio da ordem %u->%u (t=%.3f)\n",
   XX->nome, FILA_SAIDA_PM5->fila,
   DESTINO_PROX_PAL_PM5->i, XX->t);
prn_string(XX->s);
```

D.5.23.2.6 Predicado de 'LIMPA_PAL_RETIDA_SAIDA_PM5'

```
(PAL_RETIDAS_SAIDA_PM5->fila==PAL_ACABA_SAIR_PM5->fila)
```

D.5.23.2.7 Transition LIMPA_POS_LIVRE_A_REPOR_CM

D.5.23.2.8 LIMPA_POS_LIVRE_A_REPOR_CM4

D.5.23.2.9 Predicado de 'REPOE_POS_LIVRE_CM'

```
(POS_LIVRE_A_REPOR_CM->i>0)
```

D.5.23.2.10 Predicado de 'REPOE_POS_LIVRE_CM4'

```
(POS_LIVRE_A_REPOR_CM4->i>0)
```

D.5.24 Entidades da RP da página 'encam_pal_PM6_CM4_CM'

D.5.24.1 Posições

- DESTINO_PROX_PAL_PM6:ULINT;
- INICIA_DEC_ENC_PAL_PM6:NUL;
- PAL_RETIDAS_SAIDA_PM6:PALETE.

D.5.24.2 Transições

D.5.24.2.1nAcção de 'DECIDE_PM6_CM'

```
DESTINO_PROX_PAL_PM6->i=POS_LIVRES_FE_CM->i;
```

D.5.24.2.2 Acção de 'DECIDE_PM6_CM4'

```
DESTINO_PROX_PAL_PM6->i=POS_LIVRES_FE_CM4->i;
```

D.5.24.2.3 Acção de 'DECIDE_PM6_CM4_OU_CM'

```
if (random()<XX->p_PM6_CM4_CM)
{
  DESTINO_PROX_PAL_PM6->i=POS_LIVRES_FE_CM4->i;
  POS_LIVRE_A_REPOR_CM4->i=0;
  POS_LIVRE_A_REPOR_CM->i=POS_LIVRES_FE_CM->i;
}
else
{
  DESTINO_PROX_PAL_PM6->i=POS_LIVRES_FE_CM->i;
  POS_LIVRE_A_REPOR_CM->i=0;
  POS_LIVRE_A_REPOR_CM4->i=POS_LIVRES_FE_CM4->i;
}
```

D.5.24.2.4 ENVIA_ORDEM_PAL_RET_SAIDA_PM6

D.5.24.2.4.1 Predicado

```
(PAL_RETIDAS_SAIDA_PM6->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.24.2.4.2 Acção

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_PM6->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_PM6->destino;

sprintf(XX->s, "\n%s: envio da ordem %u->%u, para uma palete retida no
PM6 (t=%.3f)\n",
    XX->nome, ORDEM_MOVIM->origem, ORDEM_MOVIM->destino, xx_gettime());
prn_string(XX->s);
```

D.5.24.2.5 ENVIA_ORDEM_SAIDA_PM6

D.5.24.2.5.1 Predicado

```
(FILA_SAIDA_PM6->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.24.2.5.2 Acção

```
XX->t=xx_gettime();

ORDEM_MOVIM->origem=FILA_SAIDA_PM6->fila;
ORDEM_MOVIM->destino=DESTINO_PROX_PAL_PM6->i;

PAL_RETIDAS_SAIDA_PM6->destino=DESTINO_PROX_PAL_PM6->i;
PAL_RETIDAS_SAIDA_PM6->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: envio da ordem %u->%u (t=%.3f)\n",
   XX->nome, FILA_SAIDA_PM6->fila,
   DESTINO_PROX_PAL_PM6->i, XX->t);
prn_string(XX->s);
```

D.5.24.2.6 Predicado de 'LIMPA_PAL_RETIDA_SAIDA_PM6'

```
(PAL_RETIDAS_SAIDA_PM6->fila==PAL_ACABA_SAIR_PM6->fila)
```

D.5.25 Entidades da RP da página 'escam_pal_CM3_PM5_PM6'

D.5.25.1 Posições

- DESTINO_PROX_PAL_CM3:ULINT;
- INICIA_DEC_ENC_PAL_CM3:NUL;
- PAL_RETIDAS_SAIDA_CM3:PALETE.

D.5.25.2 Transições

D.5.25.2.1 Acção de 'DECIDE_CM3_PM5'

```
DESTINO_PROX_PAL_CM3->i=POS_LIVRES_FE_PM5->i;
```

D.5.25.2.2 Acção de 'DECIDE_CM3_PM5_OU_PM6'

```
if (random()<XX->p_CM3_PM5_PM6)
{
  DESTINO_PROX_PAL_CM3->i=POS_LIVRES_FE_PM5->i;
  POS_LIVRE_A_REPOR_PM5->i=0;
  POS_LIVRE_A_REPOR_PM6->i=POS_LIVRES_FE_PM6->i;
}
else
{
  DESTINO_PROX_PAL_CM3->i=POS_LIVRES_FE_PM6->i;
  POS_LIVRE_A_REPOR_PM6->i=0;
  POS_LIVRE_A_REPOR_PM5->i=POS_LIVRES_FE_PM5->i;
}
```

D.5.25.2.3 Acção de 'DECIDE_CM3_PM6'

```
DESTINO_PROX_PAL_CM3->i=POS_LIVRES_FE_PM6->i;
```

D.5.25.2.4 ENVIA_ORDEM_PAL_RET_SAIDA_CM3

D.5.25.2.4.1 Predicado

```
(PAL_RETIDAS_SAIDA_CM3->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.25.2.4.2

```
ORDEM_MOVIM->origem=PAL_RETIDAS_SAIDA_CM3->fila;
ORDEM_MOVIM->destino=PAL_RETIDAS_SAIDA_CM3->destino;

sprintf(XX->s, "\n%s: envio da ordem %u->%u, para uma paletm retida no
CM3 (t=%.3f)\n",
    XX->nome, ORDEM_MOVIM->origem, ORDEM_MOVIM->destino, xx_gettime());
prn_string(XX->s);
```

D.5.25.2.5 ENVIA_ORDEM_SAIDA_CM3

D.5.25.2.5.1 Predicado

```
(FILA_SAIDA_CM3->fila==PONTO_SAIDA_DISPONIVEL->i)
```

D.5.25.2.5.2 Acção

```
XX->t=xx_gettime();

ORDEM_MOVIM->origem=FILA_SAIDA_CM3->fila;
ORDEM_MOVIM->destino=DESTINO_PROX_PAL_CM3->i;

PAL_RETIDAS_SAIDA_CM3->destino=DESTINO_PROX_PAL_CM3->i;
PAL_RETIDAS_SAIDA_CM3->t_mud_est=XX->t;

sprintf(XX->s, "\n%s: envio da ordem %u->%u (t=%.3f)\n",
   XX->nome, FILA_SAIDA_CM3->fila,
   DESTINO_PROX_PAL_CM3->i, XX->t);
prn_string(XX->s);
```

D.5.25.2.6 Predicado de 'LIMPA_PAL_RETIDA_SAIDA_CM3'

```
(PAL_RETIDAS_SAIDA_CM3->fila==PAL_ACABA_SAIR_CM3->fila)
```

Anexo E – Classe Artifex SISTEMA_DE_AGVS

E.1 Parâmetros

- nome_fich_log:xx_string – nome do ficheiro onde poderão ser registadas todas as mensagens imprimidas pela função 'prn_string()' (todas as mensagens imprimidas na janela 'Simulator Input/Output').

E.2 Ficheiros de cabeçalho incluídos

```
#include "mens_log.h"
```

E.3 Acções iniciais

```
/* configura os parametros dos sub-objectos */
 if (xx_readparameters("sistagvs.ini")==-1)
  printf("\nErro na leitura dos parametros do SistAGVs\n");

/* decide se abre o ficheiro de log */
 printf("\n\nSISTEMA de Gestao de AGV's\n");
 printf(" Criar Ficheiro de Log ? (!=0 ==> sim)  Opcao:");
 scanf("%d", &XX->fich_log);
 if (XX->fich_log) abre_fich_log(XX->nome_fich_log, "w");
```

E.4 Acções finais

```
/* fecha o ficheiro de log */
 if (XX->fich_log) fecha_fich_log();
```

E.5 Variáveis locais

- fich_log:int – variável que, quando inicializada com um valor ≠0, permite a criação de um ficheiro de log das mensagens imprimidas na janela 'Simulator Input/Output'.

E.6 Objectos

- SG:SGM – objecto que modeliza o comportamento do SGM;
- SGA:SGAGV – objecto que modeliza o comportamento do SGAGV;
- SP:SPAGV – objecto que modeliza o comportamento do SPAGV;

- FABRICA:SISTEMA_FABRICO – objecto que modeliza o comportamento da fábrica servida pelo SAGVs.

Anexo F – Ficheiros de configuração do SAGVs

F.1 Parâmetros do SAGVs ('sistagvs.ini')

```
SP: nome_fich_layout        =       layout.txt
SP: nome_fich_mesas         =       mesas.txt
SP: nome_fich_nos_intr      =       nintrtar.txt
SP: numero_agvs             =       5
SP: p_erro_manual           =       0.1
SP: p_introd_or_pos         =       1.0
SP: poss_acesso             =       -1
SP: t_med_erro              =       10800.0
SP: t_med_introducao        =       60.0
SP: t_med_manual            =       172800.0
SP: t_med_perm_erro         =       300.0
SP: vel_curva               =       75
SP: vel_frente              =       100
SP: vel_tras                =       50
SGA:nome_fich_layout        =       layout.txt
SGA:nome_fich_mesas         =       mesas.txt
SGA:num_max_af              =       5
SGA:num_max_mov_avan        =       1
SGA:numero_agvs             =       5
SGA:poss_acesso             =       -1
SGA:reaf_dinam              =       1
SGA:regra_ADCT              =       0
SGA:regra_ADV               =       0
SGA:vel_curva               =       75
SGA:vel_frente              =       100
SGA:vel_tras                =       50
SG: dp_carga                =       2.0
SG: nome_fich_nos_tarefas   =       nintrtar.txt
SG: numero_agvs             =       5
SG: p_alocado               =       0.0
SG: p_cancel_1_tar          =       0.7
SG: p_cancel_tar_exec       =       0.2
SG: p_esc_sgm               =       1.0
SG: p_tar_prior             =       0.0
SG: priorid_tar_max         =       10
SG: t_med_cancel_tar        =       259200.0
SG: t_med_dis_en            =       1.0
SG: t_med_en_dis            =       604800.0
SG: t_med_mud_est_aloc      =       31104000.0
SG: vm_carga                =       10.0
FABRICA:nome                =       FABRICA
FABRICA:p_AAMP_CM1_CM2      =       0.66
FABRICA:p_AAPA_CQ_CEPA      =       0.0
FABRICA:p_CM1_PM1_PM2       =       0.5
FABRICA:p_CM2_PM3_PM4       =       0.5
FABRICA:p_CM3_PM5_PM6       =       0.5
FABRICA:p_CM_CQ_AAPA        =       0.75
FABRICA:p_CQ_AAPA_CEPA      =       0.1
```

```
FABRICA:p_CRMP_AAMP_CM1   =      0.9
FABRICA:p_PM1_PM3_CM3     =      0.5
FABRICA:p_PM2_PM4_CM3     =      0.5
FABRICA:p_PM3_PM5_PM6     =      0.5
FABRICA:p_PM4_PM5_PM6     =      0.5
FABRICA:p_PM5_CM4_CM      =      0.5
FABRICA:p_PM6_CM4_CM      =      0.5
FABRICA:t_entre_ped_AAMP  =      180.0
FABRICA:t_entre_ped_AAPA  =      60.0
```

F.2 Parâmetros do Sistema de Fabrico ('sismanuf.ini')

```
RECEPCAO_MAT_PRIMAS:      capacidade_filas      =      4
RECEPCAO_MAT_PRIMAS:      dp_tcarga             =      0.51
RECEPCAO_MAT_PRIMAS:      dp_trecep_pal         =      7.7
RECEPCAO_MAT_PRIMAS:      fila1                 =      1
RECEPCAO_MAT_PRIMAS:      filan                 =      2
RECEPCAO_MAT_PRIMAS:      nome                  =      CRMP
RECEPCAO_MAT_PRIMAS:      num_recursos          =      2
RECEPCAO_MAT_PRIMAS:      tam_lote_paletes      =      300
RECEPCAO_MAT_PRIMAS:      tm_entre_lotes        =      86400.0
RECEPCAO_MAT_PRIMAS:      tm_recep_pal          =      40.0
RECEPCAO_MAT_PRIMAS:      vm_tcarga             =      20.0
ARMAZEM_MAT_PRIMAS:       capac_armazem         =      100
ARMAZEM_MAT_PRIMAS:       capac_filas_entrada   =      4
ARMAZEM_MAT_PRIMAS:       capac_filas_saida     =      4
ARMAZEM_MAT_PRIMAS:       dp_tcd                =      0.51
ARMAZEM_MAT_PRIMAS:       dp_tcd_stoc           =      0.51
ARMAZEM_MAT_PRIMAS:       dp_tstoc_carreg       =      10.2
ARMAZEM_MAT_PRIMAS:       dp_tstoc_vazio        =      10.2
ARMAZEM_MAT_PRIMAS:       fila_entr_1           =      3
ARMAZEM_MAT_PRIMAS:       fila_entr_n           =      4
ARMAZEM_MAT_PRIMAS:       fila_saida_1          =      5
ARMAZEM_MAT_PRIMAS:       fila_saida_n          =      6
ARMAZEM_MAT_PRIMAS:       nome                  =      AAMP
ARMAZEM_MAT_PRIMAS:       vm_tcd                =      20.0
ARMAZEM_MAT_PRIMAS:       vm_tcd_stoc           =      15.0
ARMAZEM_MAT_PRIMAS:       vm_tstoc_carreg       =      40.0
ARMAZEM_MAT_PRIMAS:       vm_tstoc_vazio        =      35.0
CENTRO_MAQ_1:             capac_fila_saida      =      2
CENTRO_MAQ_1:             capac_filas_entrada   =      1
CENTRO_MAQ_1:             dp_tcd                =      0.51
CENTRO_MAQ_1:             dp_tdownload          =      5.1
CENTRO_MAQ_1:             dp_tprocess           =      30.6
CENTRO_MAQ_1:             fila_entr_1           =      26
CENTRO_MAQ_1:             fila_entr_n           =      26
CENTRO_MAQ_1:             fila_saida            =      25
CENTRO_MAQ_1:             nome                  =      CM1
CENTRO_MAQ_1:             vm_tcd                =      20.0
CENTRO_MAQ_1:             vm_tdownload          =      30.0
CENTRO_MAQ_1:             vm_tprocess           =      240.0
CENTRO_MAQ_2:             capac_fila_saida      =      4
CENTRO_MAQ_2:             capac_filas_entrada   =      2
CENTRO_MAQ_2:             dp_tcd                =      0.51
CENTRO_MAQ_2:             dp_tdownload          =      5.1
CENTRO_MAQ_2:             dp_tprocess           =      15.3
CENTRO_MAQ_2:             fila_entr_1           =      34
CENTRO_MAQ_2:             fila_entr_n           =      34
```

```
CENTRO_MAQ_2:          fila_saida              =      33
CENTRO_MAQ_2:          nome                    =      CM2
CENTRO_MAQ_2:          vm_tcd                  =      20.0
CENTRO_MAQ_2:          vm_tdownload            =      25.0
CENTRO_MAQ_2:          vm_tprocess             =      300.0
CENTRO_MAQ_3:          capac_fila_saida        =      4
CENTRO_MAQ_3:          capac_filas_entrada     =      4
CENTRO_MAQ_3:          dp_tcd                  =      0.51
CENTRO_MAQ_3:          dp_tdownload            =      2.6
CENTRO_MAQ_3:          dp_tprocess             =      15.3
CENTRO_MAQ_3:          fila_entr_1             =      10
CENTRO_MAQ_3:          fila_entr_n             =      10
CENTRO_MAQ_3:          fila_saida              =      11
CENTRO_MAQ_3:          nome                    =      CM3
CENTRO_MAQ_3:          vm_tcd                  =      20.0
CENTRO_MAQ_3:          vm_tdownload            =      20.0
CENTRO_MAQ_3:          vm_tprocess             =      270.0
CENTRO_MAQ_4:          capac_fila_saida        =      4
CENTRO_MAQ_4:          capac_filas_entrada     =      2
CENTRO_MAQ_4:          dp_tcd                  =      0.51
CENTRO_MAQ_4:          dp_tdownload            =      5.1
CENTRO_MAQ_4:          dp_tprocess             =      15.3
CENTRO_MAQ_4:          fila_entr_1             =      7
CENTRO_MAQ_4:          fila_entr_n             =      8
CENTRO_MAQ_4:          fila_saida              =      9
CENTRO_MAQ_4:          nome                    =      CM4
CENTRO_MAQ_4:          vm_tcd                  =      20.0
CENTRO_MAQ_4:          vm_tdownload            =      30.0
CENTRO_MAQ_4:          vm_tprocess             =      360.0
POSTO_TRAB_MAN_1:  capac_fila_saida            =      4
POSTO_TRAB_MAN_1:  capac_filas_entrada         =      2
POSTO_TRAB_MAN_1:  dp_tcd                      =      0.51
POSTO_TRAB_MAN_1:  dp_tdownload                =      0.51
POSTO_TRAB_MAN_1:  dp_tprocess                 =      61.2
POSTO_TRAB_MAN_1:  fila_entr_1                 =      35
POSTO_TRAB_MAN_1:  fila_entr_n                 =      35
POSTO_TRAB_MAN_1:  fila_saida                  =      36
POSTO_TRAB_MAN_1:  nome                        =      PM1
POSTO_TRAB_MAN_1:  vm_tcd                      =      20.0
POSTO_TRAB_MAN_1:  vm_tdownload                =      5.0
POSTO_TRAB_MAN_1:  vm_tprocess                 =      300.0
POSTO_TRAB_MAN_2:  capac_fila_saida            =      4
POSTO_TRAB_MAN_2:  capac_filas_entrada         =      2
POSTO_TRAB_MAN_2:  dp_tcd                      =      0.51
POSTO_TRAB_MAN_2:  dp_tdownload                =      0.51
POSTO_TRAB_MAN_2:  dp_tprocess                 =      61.2
POSTO_TRAB_MAN_2:  fila_entr_1                 =      37
POSTO_TRAB_MAN_2:  fila_entr_n                 =      37
POSTO_TRAB_MAN_2:  fila_saida                  =      38
POSTO_TRAB_MAN_2:  nome                        =      PM2
POSTO_TRAB_MAN_2:  vm_tcd                      =      20.0
POSTO_TRAB_MAN_2:  vm_tdownload                =      5.0
POSTO_TRAB_MAN_2:  vm_tprocess                 =      300.0
POSTO_TRAB_MAN_3:  capac_fila_saida            =      4
POSTO_TRAB_MAN_3:  capac_filas_entrada         =      4
POSTO_TRAB_MAN_3:  dp_tcd                      =      0.51
POSTO_TRAB_MAN_3:  dp_tdownload                =      0.51
POSTO_TRAB_MAN_3:  dp_tprocess                 =      45.9
POSTO_TRAB_MAN_3:  fila_entr_1                 =      39
```

```
POSTO_TRAB_MAN_3:  fila_entr_n              =      39
POSTO_TRAB_MAN_3:  fila_saida               =      40
POSTO_TRAB_MAN_3:  nome                     =      PM3
POSTO_TRAB_MAN_3:  vm_tcd                   =      20.0
POSTO_TRAB_MAN_3:  vm_tdownload             =      5.0
POSTO_TRAB_MAN_3:  vm_tprocess              =      240.0
POSTO_TRAB_MAN_4:  capac_fila_saida         =      4
POSTO_TRAB_MAN_4:  capac_filas_entrada      =      4
POSTO_TRAB_MAN_4:  dp_tcd                   =      0.51
POSTO_TRAB_MAN_4:  dp_tdownload             =      0.51
POSTO_TRAB_MAN_4:  dp_tprocess              =      45.9
POSTO_TRAB_MAN_4:  fila_entr_1              =      27
POSTO_TRAB_MAN_4:  fila_entr_n              =      27
POSTO_TRAB_MAN_4:  fila_saida               =      28
POSTO_TRAB_MAN_4:  nome                     =      PM4
POSTO_TRAB_MAN_4:  vm_tcd                   =      20.0
POSTO_TRAB_MAN_4:  vm_tdownload             =      5.0
POSTO_TRAB_MAN_4:  vm_tprocess              =      240.0
POSTO_TRAB_MAN_5:  capac_fila_saida         =      4
POSTO_TRAB_MAN_5:  capac_filas_entrada      =      4
POSTO_TRAB_MAN_5:  dp_tcd                   =      0.51
POSTO_TRAB_MAN_5:  dp_tdownload             =      0.51
POSTO_TRAB_MAN_5:  dp_tprocess              =      61.2
POSTO_TRAB_MAN_5:  fila_entr_1              =      29
POSTO_TRAB_MAN_5:  fila_entr_n              =      29
POSTO_TRAB_MAN_5:  fila_saida               =      30
POSTO_TRAB_MAN_5:  nome                     =      PM5
POSTO_TRAB_MAN_5:  vm_tcd                   =      20.0
POSTO_TRAB_MAN_5:  vm_tdownload             =      5.0
POSTO_TRAB_MAN_5:  vm_tprocess              =      360.0
POSTO_TRAB_MAN_6:  capac_fila_saida         =      4
POSTO_TRAB_MAN_6:  capac_filas_entrada      =      4
POSTO_TRAB_MAN_6:  dp_tcd                   =      0.51
POSTO_TRAB_MAN_6:  dp_tdownload             =      0.51
POSTO_TRAB_MAN_6:  dp_tprocess              =      61.2
POSTO_TRAB_MAN_6:  fila_entr_1              =      31
POSTO_TRAB_MAN_6:  fila_entr_n              =      31
POSTO_TRAB_MAN_6:  fila_saida               =      32
POSTO_TRAB_MAN_6:  nome                     =      PM6
POSTO_TRAB_MAN_6:  vm_tcd                   =      20.0
POSTO_TRAB_MAN_6:  vm_tdownload             =      5.0
POSTO_TRAB_MAN_6:  vm_tprocess              =      360.0
CELULA_MONTAGEM:   capac_fila_saida         =      2
CELULA_MONTAGEM:   capac_filas_entrada      =      2
CELULA_MONTAGEM:   dp_tcd                   =      0.51
CELULA_MONTAGEM:   dp_tdownload             =      2.6
CELULA_MONTAGEM:   dp_tprocess              =      30.6
CELULA_MONTAGEM:   fila_entr_1              =      12
CELULA_MONTAGEM:   fila_entr_n              =      14
CELULA_MONTAGEM:   fila_saida               =      16
CELULA_MONTAGEM:   nome                     =      CM
CELULA_MONTAGEM:   vm_tcd                   =      20.0
CELULA_MONTAGEM:   vm_tdownload             =      30.0
CELULA_MONTAGEM:   vm_tprocess              =      300.0
CONTR_QUALIDADE:   capac_fila_saida         =      2
CONTR_QUALIDADE:   capac_filas_entrada      =      2
CONTR_QUALIDADE:   dp_tcd                   =      0.51
CONTR_QUALIDADE:   dp_tdownload             =      2.6
CONTR_QUALIDADE:   dp_tprocess              =      10.2
```

```
CONTR_QUALIDADE:   fila_entr_1                    =      17
CONTR_QUALIDADE:   fila_entr_n                    =      17
CONTR_QUALIDADE:   fila_saida                     =      18
CONTR_QUALIDADE:   nome                           =      CQ
CONTR_QUALIDADE:   vm_tcd                         =      20.0
CONTR_QUALIDADE:   vm_tdownload                   =      10.0
CONTR_QUALIDADE:   vm_tprocess                    =      120.0
ARMAZEM_PROD_ACABADO:     capac_armazem           =      100
ARMAZEM_PROD_ACABADO:     capac_filas_entrada     =      2
ARMAZEM_PROD_ACABADO:     capac_filas_saida       =      4
ARMAZEM_PROD_ACABADO:     dp_tcd                  =      0.51
ARMAZEM_PROD_ACABADO:     dp_tcd_stoc             =      0.51
ARMAZEM_PROD_ACABADO:     dp_tstoc_carreg         =      10.2
ARMAZEM_PROD_ACABADO:     dp_tstoc_vazio          =      10.2
ARMAZEM_PROD_ACABADO:     fila_entr_1             =      19
ARMAZEM_PROD_ACABADO:     fila_entr_n             =      20
ARMAZEM_PROD_ACABADO:     fila_saida_1            =      21
ARMAZEM_PROD_ACABADO:     fila_saida_n            =      22
ARMAZEM_PROD_ACABADO:     nome                    =      AAPA
ARMAZEM_PROD_ACABADO:     vm_tcd                  =      20.0
ARMAZEM_PROD_ACABADO:     vm_tcd_stoc             =      15.0
ARMAZEM_PROD_ACABADO:     vm_tstoc_carreg         =      40.0
ARMAZEM_PROD_ACABADO:     vm_tstoc_vazio          =      35.0
EXPEDICAO_PROD_ACABADO: capacidade_filas          =      2
EXPEDICAO_PROD_ACABADO: dp_exped_pal              =      7.7
EXPEDICAO_PROD_ACABADO: dp_tcarreg                =      459.2
EXPEDICAO_PROD_ACABADO: dp_tdescarga              =      0.51
EXPEDICAO_PROD_ACABADO: fila1                     =      23
EXPEDICAO_PROD_ACABADO: filan                     =      24
EXPEDICAO_PROD_ACABADO: nome                      =      CEPA
EXPEDICAO_PROD_ACABADO: num_recursos              =      2
EXPEDICAO_PROD_ACABADO: tam_lote_paletes          =      50
EXPEDICAO_PROD_ACABADO: tm_entre_carreg           =      86400.0
EXPEDICAO_PROD_ACABADO: tm_exped_pal              =      40.0
EXPEDICAO_PROD_ACABADO: vm_tcarreg                =      1800.0
EXPEDICAO_PROD_ACABADO: vm_tdescarga              =      20.0
```

F.3 Configuração do layout ('layout.txt')

```
/*  Ficheiro de Layout

Tipos de Nos:
     TRANSITO        0
     CARGA_DESCARGA  1
     CARGA           2
     DESCARGA        3

   Tipos de Ramos:
     TERMINAL        0
     PASSAGEM        1
     TERM_PASSAG     2
*/

N:1,0,0,0,0,2,0,0,0,0,0
R:2,6600,0,0,0,0,2,0,0,0,0
B:(1,1),(2,1)
R:101,4200,2000,0,10,0,2,0,0,0,
0
B:(101,1),(201,1)

N:2,0,0,0,0,2,0,0,0,0,0
R:3,2400,0,0,10,0,2,0,0,0,0
B:(3,1)

N:3,0,0,0,0,3,0,0,0,0,0
R:2,0,0,2400,0,0,0,0,0,0,0
B:(2,1)
R:4,5100,0,0,10,0,2,0,0,0,0
B:(4,1)
R:102,3000,2000,0,0,0,1,0,0,0,0
B:(102,1),(202,1)

N:4,0,0,0,0,3,0,0,0,0,0
R:5,5100,0,0,10,0,2,0,0,0,0
B:(5,1)
R:103,3000,2000,0,10,0,1,0,0,0,
0
```

```
B:(103,1),(203,1)

N:5,0,0,0,0,2,0,0,0,0,0
R:4,0,0,5100,0,0,0,0,0,0,0
B:(4,1)
R:6,5100,0,0,10,0,2,0,0,0,0
B:(6,1)
R:104,3000,2000,0,10,0,1,0,0,0,
0
B:(104,1),(204,1)

N:6,0,0,0,0,2,0,0,0,0,0
R:7,6000,0,0,10,0,2,0,0,0,0
B:(7,1),(8,1)
R:105,3000,2000,0,10,0,1,0,0,0,
0
B:(105,1),(205,1)

N:7,0,0,0,0,3,0,0,0,0,0
O:8
R:6,0,0,6000,0,0,0,0,0,0,0
B:(6,1)
R:8,1400,0,0,10,0,2,0,0,0,0
B:(8,1)

N:8,0,0,0,0,3,0,0,0,0,0
O:7
R:7,0,0,1400,0,0,0,0,0,0,0
B:(7,1)
R:9,2000,0,0,10,0,2,0,0,0,0
B:(9,1)

N:9,0,0,0,0,2,0,0,0,0,0
R:8,0,0,2000,0,0,0,0,0,0,0
B:(8,1),(7,1)
R:10,4200,0,0,10,0,2,0,0,0,0
B:(10,1)
R:112,3200,2000,0,0,0,1,0,0,0,0
B:(112,1),(206,1)

N:10,0,0,0,0,3,0,0,0,0,0
R:9,0,0,4200,0,0,0,0,0,0,0
B:(9,1)
R:11,2400,0,0,10,0,2,0,0,0,0
B:(11,1)
R:110,4500,2000,0,10,0,1,0,0,0,
0
B:(110,1),(207,1)

N:11,0,0,0,0,2,0,0,0,0,0
R:10,0,0,2400,0,0,0,0,0,0,0
B:(10,1)
R:109,4500,2000,0,10,0,1,0,0,0,
0
B:(109,1)

N:12,0,0,0,0,3,0,0,0,0,0
R:111,3600,0,0,0,0,1,0,0,0,0
B:(111,1)

N:13,0,0,0,0,3,0,0,0,0,0
R:14,4500,0,0,10,0,2,0,0,0,0
B:(14,1)
R:126,3000,2000,0,10,0,1,0,0,0,
0
B:(126,1),(216,1)

N:14,0,0,0,0,3,0,0,0,0,0
R:15,4200,2000,0,0,0,1,0,0,0,0
B:(15,1)
R:128,6000,2000,0,10,0,1,0,0,0,
0
B:(128,1),(217,1)
R:130,7500,2000,0,10,0,1,0,0,0,
0
B:(130,1),(217,1)

N:15,0,0,0,0,0,0,0,0,0,0
R:16,2000,0,0,10,0,2,0,0,0,0
B:(16,1)

N:16,0,0,0,0,2,0,0,0,0,0
R:129,4500,2000,0,0,0,1,0,0,0,0
B:(129,1)

N:17,0,0,0,0,3,0,0,0,0,0
R:18,4000,0,0,10,0,2,0,0,0,0
B:(18,1)
R:130,3900,3000,0,0,0,1,0,0,0,0
B:(130,1),(220,1)

N:18,0,0,0,0,2,0,0,0,0,0
R:17,0,0,4000,0,0,0,0,0,0,0
B:(17,1)
R:19,5400,0,0,10,0,2,0,0,0,0
B:(19,1)
R:131,2000,3000,0,10,0,1,0,0,0,
0
B:(131,1),(220,1)

N:19,0,0,0,0,3,0,0,0,0,0
R:18,0,0,5400,0,0,0,0,0,0,0
B:(18,1)
R:20,6000,0,0,10,0,2,0,0,0,0
B:(20,1)
R:132,2000,2000,0,10,0,1,0,0,0,
0
B:(132,1),(221,1)

N:20,0,0,0,0,3,0,0,0,0,0
R:21,6000,0,0,10,0,2,0,0,0,0
B:(21,1)
R:133,2000,2000,0,10,0,1,0,0,0,
0
B:(133,1),(222,1)

N:21,0,0,0,0,2,0,0,0,0,0
R:20,0,0,6000,0,0,0,0,0,0,0
```

```
B:(20,1)
R:22,6000,0,0,10,0,2,0,0,0,0
B:(22,1)
R:134,2000,2000,0,10,0,1,0,0,0,
0
B:(134,1),(223,1)

N:22,0,0,0,0,2,0,0,0,0,0
R:23,6000,0,0,10,0,2,0,0,0,0
B:(23,1)
R:135,2000,2000,0,10,0,1,0,0,0,
0
B:(135,1),(224,1)

N:23,0,0,0,0,3,0,0,0,0,0
R:22,0,0,6000,0,0,0,0,0,0,0
B:(22,1)
R:24,4500,0,0,10,0,2,0,0,0,0
B:(24,1)
R:136,2000,2000,0,10,0,1,0,0,0,
0
B:(136,1),(225,1)
R:139,4500,1000,0,10,0,1,0,0,0,
0
B:(139,1),(225,1)

N:24,0,0,0,0,3,0,0,0,0,0
R:137,2000,0,0,10,0,1,0,0,0,0
B:(137,1)

N:25,0,0,0,0,2,0,0,0,0,0
R:26,4500,0,0,10,0,2,0,0,0,0
B:(26,1)
R:154,2000,2000,0,10,0,1,0,0,0,
0
B:(154,1),(227,1)

N:26,0,0,0,0,3,0,0,0,0,0
R:25,0,0,4500,0,0,0,0,0,0,0
B:(25,1)
R:121,2000,1000,0,10,0,1,0,0,0,
0
B:(121,1),(215,1)
R:124,4500,0,0,10,0,1,0,0,0,0
B:(124,1),(215,1),(201,1)

N:27,0,0,0,0,3,0,0,0,0,0
R:28,3000,0,0,10,0,2,0,0,0,0
B:(28,1)

N:28,0,0,0,0,2,0,0,0,0,0
R:27,0,0,3000,0,0,0,0,0,0,0
B:(27,1)
R:29,3000,0,0,10,0,2,0,0,0,0
B:(29,1)
R:142,2000,2000,0,0,0,1,0,0,0,0
B:(142,1),(229,1)

N:29,0,0,0,0,3,0,0,0,0,0
R:28,0,0,3000,0,0,0,0,0,0,0
B:(28,1)
R:30,3000,0,0,10,0,2,0,0,0,0
B:(30,1)

N:30,0,0,0,0,2,0,0,0,0,0
R:29,0,0,3000,0,0,0,0,0,0,0
B:(29,1)
R:31,3000,0,0,10,0,2,0,0,0,0
B:(31,1)
R:143,2000,2000,0,0,0,1,0,0,0,0
B:(143,1),(230,1)

N:31,0,0,0,0,3,0,0,0,0,0
R:30,0,0,3000,0,0,0,0,0,0,0
B:(30,1)
R:32,3000,0,0,10,0,2,0,0,0,0
B:(32,1)

N:32,0,0,0,0,2,0,0,0,0,0
R:31,0,0,3000,0,0,0,0,0,0,0
B:(31,1)
R:33,3000,0,0,10,0,2,0,0,0,0
B:(33,1)
R:144,2000,2000,0,0,0,1,0,0,0,0
B:(144,1),(231,1)

N:33,0,0,0,0,2,0,0,0,0,0
R:145,6000,0,0,10,0,1,0,0,0,0
B:(145,1),(218,1),(219,1)
R:147,3000,2000,0,10,0,1,0,0,0,
0
B:(147,1),(219,1)

N:34,0,0,0,0,3,0,0,0,0,0
R:35,3000,0,0,10,0,2,0,0,0,0
B:(35,1)
R:117,2000,2000,0,10,0,1,0,0,0,
0
B:(117,1),(211,1)

N:35,0,0,0,0,3,0,0,0,0,0
R:36,3000,0,0,10,0,2,0,0,0,0
B:(36,1)

N:36,0,0,0,0,2,0,0,0,0,0
R:35,0,0,3000,0,0,0,0,0,0,0
B:(35,1)
R:37,3000,0,0,10,0,2,0,0,0,0
B:(37,1)
R:118,2000,2000,0,0,0,1,0,0,0,0
B:(118,1),(212,1)

N:37,0,0,0,0,3,0,0,0,0,0
R:36,0,0,3000,0,0,0,0,0,0,0
B:(36,1)
R:38,3000,0,0,10,0,2,0,0,0,0
B:(38,1)
```

```
N:38,0,0,0,0,2,0,0,0,0,0
R:37,0,0,3000,0,0,0,0,0,0,0
B:(37,1)
R:39,3000,0,0,10,0,2,0,0,0,0
B:(39,1)
R:119,2000,2000,0,0,0,1,0,0,0,0
B:(119,1),(213,1)

N:39,0,0,0,0,3,0,0,0,0,0
R:38,0,0,3000,0,0,0,0,0,0,0
B:(38,1)
R:40,3000,0,0,10,0,2,0,0,0,0
B:(40,1)

N:40,0,0,0,0,2,0,0,0,0,0
R:39,0,0,3000,0,0,0,0,0,0,0
B:(39,1)
R:120,4500,0,0,0,0,1,0,0,0,0
B:(120,1),(214,1),(215,1)
R:150,3900,1000,0,0,0,1,0,0,0,0
B:(150,1),(214,1)

N:101,0,0,0,0,0,0,0,0,0,0
R:4,6000,2000,0,0,0,2,0,0,0,0
B:(4,1),(202,1)
R:102,7500,0,0,0,0,1,0,0,0,0
B:(102,1)
R:120,2000,3000,0,0,0,1,0,0,0,0
B:(120,1),(214,1)
R:150,6000,1000,0,0,0,1,0,0,0,0
B:(150,1),(214,1)

N:102,0,0,0,0,0,0,0,0,0,0
R:5,2000,2000,0,0,0,2,0,0,0,0
B:(5,1),(203,1)
R:103,5100,0,0,0,0,1,0,0,0,0
B:(103,1)

N:103,0,0,0,0,0,0,0,0,0,0
R:6,2000,2000,0,0,0,2,0,0,0,0
B:(6,1),(204,1)
R:104,5100,0,0,0,0,1,0,0,0,0
B:(104,1)

N:104,0,0,0,0,0,0,0,0,0,0
R:7,3600,2000,0,0,0,2,0,0,0,0
B:(7,1),(8,1),(205,1)
R:105,6000,0,0,0,0,1,0,0,0,0
B:(105,1)
R:117,3000,2000,0,0,0,1,0,0,0,0
B:(117,1)

N:105,0,0,0,0,0,0,0,0,0,0
R:106,3000,0,0,0,0,1,0,0,0,0
B:(106,1)

N:106,0,0,0,0,0,0,0,0,0,0
R:10,3600,2000,0,0,0,2,0,0,0,0
B:(10,1),(206,1)
R:107,6000,0,0,0,0,1,0,0,0,0
B:(107,1),(206,1)
R:112,1000,1000,0,0,0,1,0,0,0,0
B:(112,1),(206,1)

N:107,0,0,0,0,0,0,0,0,0,0
R:11,3000,2000,0,0,0,2,0,0,0,0
B:(11,1),(207,1)
R:108,3600,0,0,0,0,1,0,0,0,0
B:(108,1),(207,1)

N:108,0,0,0,0,0,0,0,0,0,0
R:109,3000,1000,0,0,0,1,0,0,0,0
B:(109,1)

N:109,0,0,0,0,0,0,0,0,0,0
R:12,3000,1000,0,0,0,2,0,0,0,0
B:(12,1)

N:110,0,0,0,0,0,0,0,0,0,0
R:111,3000,1000,0,0,0,1,0,0,0,0
B:(111,1)

N:111,0,0,0,0,0,0,0,0,0,0
R:13,3000,1000,0,0,0,2,0,0,0,0
B:(13,1),(208,1)
R:113,3000,0,0,0,0,1,0,0,0,0
B:(113,1),(208,1),(209,1)

N:112,0,0,0,0,0,0,0,0,0,0
R:13,4500,0,0,0,0,2,0,0,0,0
B:(13,1),(208,1)
R:113,3000,1000,0,0,0,1,0,0,0,0
B:(113,1),(208,1),(209,1)

N:113,0,0,0,0,0,0,0,0,0,0
O:208,209
R:114,3000,0,0,0,0,1,0,0,0,0
B:(114,1),(209,1),(210,1)
R:115,3000,2000,0,0,0,1,0,0,0,0
B:(115,1),(209,1)
R:125,3000,1000,0,0,0,1,0,0,0,0
B:(125,1),(209,1)

N:114,0,0,0,0,0,0,0,0,0,0
O:209,210
R:34,5400,0,0,0,0,2,0,0,0,0
B:(34,1),(210,1)

N:115,0,0,0,0,0,0,0,0,0,0
R:116,3600,0,0,0,0,1,0,0,0,0
B:(116,1)

N:116,0,0,0,0,0,0,0,0,0,0
R:35,3000,2000,0,0,0,2,0,0,0,0
B:(35,1),(211,1)
R:117,6000,0,0,0,0,1,0,0,0,0
B:(117,1)
```

```
N:117,0,0,0,0,0,0,0,0,0,0
R:37,3000,2000,0,0,0,2,0,0,0,0
B:(37,1),(212,1)
R:104,3000,2000,0,0,0,1,0,0,0,0
B:(104,1)
R:118,6000,0,0,0,0,1,0,0,0,0
B:(118,1)

N:118,0,0,0,0,0,0,0,0,0,0
R:39,3000,2000,0,0,0,2,0,0,0,0
B:(39,1),(213,1)
R:119,6000,0,0,0,0,1,0,0,0,0
B:(119,1)

N:119,0,0,0,0,0,0,0,0,0,0
R:120,4500,2000,0,0,0,1,0,0,0,0
B:(120,1),(214,1)
R:150,6000,1000,0,0,0,1,0,0,0,0
B:(150,1),(214,1)

N:120,0,0,0,0,0,0,0,0,0,0
O:214
R:121,6000,0,0,0,0,1,0,0,0,0
B:(121,1),(215,1)
R:124,3000,1000,0,0,0,1,0,0,0,0
B:(124,1),(215,1)

N:121,0,0,0,0,0,0,0,0,0,0
R:122,3000,1000,0,0,0,1,0,0,0,0
B:(122,1)

N:122,0,0,0,0,0,0,0,0,0,0
R:1,4500,1000,0,0,0,2,0,0,0,0
B:(1,1)
R:123,3000,1000,0,0,0,1,0,0,0,0
B:(123,1)

N:123,0,0,0,0,0,0,0,0,0,0
R:2,3000,2000,0,0,0,2,0,0,0,0
B:(2,1),(201,1)
R:101,6000,0,0,0,0,1,0,0,0,0
B:(101,1),(201,1)

N:124,0,0,0,0,0,0,0,0,0,0
R:2,3000,2000,0,0,0,2,0,0,0,0
B:(2,1),(201,1)
R:101,3000,1000,0,0,0,1,0,0,0,0
B:(101,1),(201,1)

N:125,0,0,0,0,0,0,1,0,0,0
R:14,3000,2000,0,0,0,2,0,0,0,0
B:(14,1),(216,1)
R:126,4500,0,0,0,0,1,0,0,0,0
B:(126,1)

N:126,0,0,0,0,0,0,0,0,0,0
R:127,3000,1000,0,0,0,1,0,0,0,0
B:(127,1),(217,1),(218,1)
R:146,4500,0,0,0,0,1,0,0,0,0
B:(146,1),(218,1)

N:127,0,0,0,0,0,0,0,0,0,0
O:217,218
R:15,3000,0,0,0,0,1,0,0,0,0
B:(15,1),(217,1)
R:128,4500,3000,0,0,0,1,0,0,0,0
B:(128,1),(217,1)
R:130,4500,2000,0,0,0,1,0,0,0,0
B:(130,1),(217,1)

N:128,0,0,0,0,0,0,0,0,0,0
R:129,3000,1000,0,0,0,1,0,0,0,0
B:(129,1)

N:129,0,0,0,0,0,0,0,0,0,0
R:17,3000,1000,0,0,0,2,0,0,0,0
B:(17,1)

N:130,0,0,0,0,0,0,0,0,0,0
O:220
R:131,3000,0,0,0,0,1,0,0,0,0
B:(131,1),(220,1)

N:131,0,0,0,0,0,0,0,0,0,0
O:220
R:20,3000,2000,0,0,0,2,0,0,0,0
B:(20,1),(221,1)
R:132,6900,0,0,0,0,1,0,0,0,0
B:(132,1)
R:145,6000,2000,0,0,0,1,0,0,0,0
B:(145,1),(218,1),(219,1)
R:147,4500,1000,0,0,0,1,0,0,0,0
B:(147,1),(219,1)

N:132,0,0,0,0,0,0,0,0,0,0
R:21,3000,2000,0,0,0,2,0,0,0,0
B:(21,1),(222,1)
R:133,7500,0,0,0,0,1,0,0,0,0
B:(133,1)

N:133,0,0,0,0,0,0,0,0,0,0
R:22,3000,2000,0,0,0,2,0,0,0,0
B:(22,1),(223,1)
R:134,7500,0,0,0,0,1,0,0,0,0
B:(134,1)
R:142,3000,2000,0,0,0,1,0,0,0,0
B:(142,1)

N:134,0,0,0,0,0,0,0,0,0,0
R:23,3000,2000,0,0,0,2,0,0,0,0
B:(23,1),(224,1)
R:135,7500,0,0,0,0,1,0,0,0,0
B:(135,1)

N:135,0,0,0,0,0,0,0,0,0,0
R:24,3000,2000,0,0,0,2,0,0,0,0
B:(24,1),(225,1)
R:136,7500,0,0,0,0,1,0,0,0,0
```

```
B:(136,1),(225,1)
R:139,3000,1000,0,0,0,1,0,0,0,0
B:(139,1),(225,1)

N:136,0,0,0,0,0,0,0,0,0,0
R:138,4500,2000,0,0,0,1,0,0,0,0
B:(138,1),(232,1)
R:155,7500,2000,0,0,0,1,0,0,0,0
B:(155,1),(232,1)

N:137,0,0,0,0,0,0,0,0,0,0
R:138,7500,2000,0,0,0,1,0,0,0,0
B:(138,1),(232,1)
R:155,9000,2000,0,0,0,1,0,0,0,0
B:(155,1),(232,1)

N:138,0,0,0,0,0,0,0,0,0,0
R:140,6000,0,0,0,0,1,0,0,0,0
B:(140,1),(139,1)

N:139,0,0,0,0,0,0,0,0,0,0
R:25,4500,0,0,0,0,1,0,0,0,0
B:(25,1),(226,1)
R:153,3000,2000,0,0,0,1,0,0,0,0
B:(153,1),(156,1),(226,1)
R:156,3000,2000,0,0,0,1,0,0,0,0
B:(156,1),(153,1),(226,1)

N:140,0,0,0,0,0,0,0,0,0,0
R:141,6000,0,0,0,0,1,0,0,0,0
B:(141,1)

N:141,0,0,0,0,0,0,0,0,0,0
R:29,3000,2000,0,0,0,2,0,0,0,0
B:(29,1),(229,1)
R:142,6000,0,0,0,0,1,0,0,0,0
B:(142,1)

N:142,0,0,0,0,0,0,0,0,0,0
R:31,3000,2000,0,0,0,2,0,0,0,0
B:(31,1),(230,1)
R:133,3000,2000,0,0,0,1,0,0,0,0
B:(133,1)
R:143,6000,0,0,0,0,1,0,0,0,0
B:(143,1)

N:143,0,0,0,0,0,0,0,0,0,0
R:33,3000,2000,0,0,0,2,0,0,0,0
B:(33,1),(231,1)
R:144,6000,0,0,0,0,1,0,0,0,0
B:(144,1)

N:144,0,0,0,0,0,0,0,0,0,0
R:145,3000,2000,0,0,0,1,0,0,0,0
B:(145,1),(218,1),(219,1)
R:147,4500,1000,0,0,0,1,0,0,0,0
B:(147,1),(219,1)

N:145,0,0,0,0,0,0,0,0,0,0
O:218,219
R:127,3000,0,0,0,0,1,0,0,0,0
B:(127,1),(217,1),(218,1)
R:146,3000,1000,0,0,0,1,0,0,0,0
B:(146,1)

N:146,0,0,0,0,0,0,0,0,0,0
R:19,4500,1000,0,0,0,2,0,0,0,0
B:(19,1),(220,1)
R:131,3000,1000,0,0,0,1,0,0,0,0
B:(131,1),(220,1)

N:147,0,0,0,0,0,0,0,0,0,0
R:148,3000,0,0,0,0,1,0,0,0,0
B:(148,1)

N:148,0,0,0,0,0,0,0,0,0,0
R:149,3000,0,0,0,0,1,0,0,0,0
B:(149,1)

N:149,0,0,0,0,0,0,0,0,0,0
R:34,4500,1000,0,0,0,2,0,0,0,0
B:(34,1),(210,1)
R:116,4500,1000,0,0,0,1,0,0,0,0
B:(116,1),(210,1)

N:150,0,0,0,0,0,0,0,0,0,0
R:151,3000,0,0,0,0,1,0,0,0,0
B:(151,1)

N:151,0,0,0,0,0,0,0,0,0,0
R:152,3000,0,0,0,0,1,0,0,0,0
B:(152,1)

N:152,0,0,0,0,0,0,0,0,0,0
R:27,4500,1000,0,0,0,2,0,0,0,0
B:(27,1),(228,1)
R:141,6000,1000,0,0,0,1,0,0,0,0
B:(141,1),(228,1)

N:153,0,0,0,0,0,0,0,0,0,0
R:26,3000,2000,0,0,0,2,0,0,0,0
B:(26,1),(227,1)
R:154,6000,0,0,0,0,1,0,0,0,0
B:(154,1)

N:154,0,0,0,0,0,0,0,0,0,0
R:121,6000,1000,0,0,0,1,0,0,0,0
B:(121,1),(215,1)
R:124,3000,2000,0,0,0,1,0,0,0,0
B:(124,1),(215,1)

N:155,0,0,0,0,0,0,0,0,0,0
R:25,3000,1000,0,0,0,2,0,0,0,0
B:(25,1),(226,1)
R:153,6000,1000,0,0,0,1,0,0,0,0
B:(153,1),(156,1),(226,1)
R:156,4500,0,0,0,0,1,0,0,0,0
```

```
B:(156,1),(226,1),(153,1)

N:156,0,0,0,0,0,0,0,0,0,0
R:27,6000,0,0,0,0,2,0,0,0,0
B:(27,1),(228,1)


# Nos ficticios utilizados como
semaforos nos cruzamentos

N:201,0,0,0,0,0,0,0,0,0,0
N:202,0,0,0,0,0,0,0,0,0,0
N:203,0,0,0,0,0,0,0,0,0,0
N:204,0,0,0,0,0,0,0,0,0,0
N:205,0,0,0,0,0,0,0,0,0,0
N:206,0,0,0,0,0,0,0,0,0,0
N:207,0,0,0,0,0,0,0,0,0,0
N:208,0,0,0,0,0,0,0,0,0,0
N:209,0,0,0,0,0,0,0,0,0,0
N:210,0,0,0,0,0,0,0,0,0,0
N:211,0,0,0,0,0,0,0,0,0,0
N:212,0,0,0,0,0,0,0,0,0,0
N:213,0,0,0,0,0,0,0,0,0,0
N:214,0,0,0,0,0,0,0,0,0,0
N:215,0,0,0,0,0,0,0,0,0,0
N:216,0,0,0,0,0,0,0,0,0,0
N:217,0,0,0,0,0,0,0,0,0,0
N:218,0,0,0,0,0,0,0,0,0,0
N:219,0,0,0,0,0,0,0,0,0,0
N:220,0,0,0,0,0,0,0,0,0,0
N:221,0,0,0,0,0,0,0,0,0,0
N:222,0,0,0,0,0,0,0,0,0,0
N:223,0,0,0,0,0,0,0,0,0,0
N:224,0,0,0,0,0,0,0,0,0,0
N:225,0,0,0,0,0,0,0,0,0,0
N:226,0,0,0,0,0,0,0,0,0,0
N:227,0,0,0,0,0,0,0,0,0,0
N:228,0,0,0,0,0,0,0,0,0,0
N:229,0,0,0,0,0,0,0,0,0,0
N:230,0,0,0,0,0,0,0,0,0,0
N:231,0,0,0,0,0,0,0,0,0,0
N:232,0,0,0,0,0,0,0,0,0,0
```

F.4 Configuração dos pontos de carga/descarga ('mesas.txt')

```
/* Ficheiro de Configuracao dos Pontos de Carga/Descarga */
/*

 M:<id_no>,<pos_rel>

 <pos_rel> vale -1 ou 1, consoante a posicao da mesa relativamente
 ao referencial arbitrado para os sentidos fixo no no´ <id_no>
*/

M:1,-1
M:2,-1
M:3,-1
M:4,-1
M:5,-1
M:6,-1
M:7,-1
M:8,-1
M:9,-1
M:10,-1
M:11,-1
M:12,-1
M:13,-1
M:14,-1
M:16,-1
M:17,-1
M:18,-1
M:19,-1
M:20,-1
M:21,-1
M:22,-1
M:23,-1
M:24,-1
M:25,-1
M:26,-1
```

```
M:27,-1
M:28,-1
M:29,-1
M:30,-1
M:31,-1
M:32,-1
M:33,-1
M:34,-1
M:35,-1
M:36,-1
M:37,-1
M:38,-1
M:39,-1
M:40,-1
```

F.5 Configuração dos nós para introdução e posicionamento de AGVs ('nintrtar.txt')

```
/* Ficheiro de Configuracao dos Nos de Introducao/Tarefas */
/*

 N:<id_no>,<p_introd>,<p_posic>

 <p_...> toma valores entre 0.0 e 1.0
 o somatorio de cada parametro para todos os nos tem de ser igual a 1.0
*/

N:101,0.0625,0.04
N:102,0.0625,0.04
N:103,0.0625,0.04
N:104,0.0625,0.04
N:105,0.0625,0.04
N:106,0,0.04
N:115,0.0625,0.04
N:116,0.0625,0.04
N:117,0.0625,0.04
N:118,0.0625,0.04
N:119,0.0625,0.04
N:132,0.0625,0.04
N:133,0.0625,0.04
N:134,0.0625,0.04
N:141,0.06250,0.04
N:142,0.0625,0.04
N:143,0.0625,0.04
N:144,0,0.04
N:147,0,0.04
N:148,0,0.04
N:149,0,0.04
N:150,0,0.04
N:151,0,0.04
N:152,0,0.04
N:153,0,0.04
```

Anexo G – Excertos do ficheiro de log de uma simulação do SAGVs

```
Ficheiro aberto em 98/08/12, Quarta-feira, 17:47:08


SGM: leitura do ficheiro 'nintrtar.txt' c/ probabilidades dos nos...
linha 10 (no 101)
linha 11 (no 102)
linha 12 (no 103)
linha 13 (no 104)
linha 14 (no 105)
linha 15 (no 106)
linha 16 (no 115)
linha 17 (no 116)
linha 18 (no 117)

(...)

SGM: verifica somatorio das probab. dos nos de tarefas...
Verificacao concluida.
SGM: AGV 1 no estado DISABLED criado
SGM: AGV 2 no estado DISABLED criado
SGM: AGV 3 no estado DISABLED criado
SGM: AGV 4 no estado DISABLED criado
SGM: AGV 5 no estado DISABLED criado

SGAGV: Leitura do Ficheiro de Layout 'layout.txt'
 leitura na linha 20 do no 1
  leitura dos ramos do no 1
   leitura na linha 21 do ramo 1->2...
    leitura dos bloqueios do ramo 1->2 (linha 22)...
    concluida.
   ramo 1->2 lido com sucesso
   leitura na linha 23 do ramo 1->101...
    leitura dos bloqueios do ramo 1->101 (linha 24)...
    concluida.
   ramo 1->101 lido com sucesso
  ramos do no 1 lidos com sucesso
 no 1 lido com sucesso
 leitura na linha 26 do no 2
  leitura dos ramos do no 2
   leitura na linha 27 do ramo 2->3...
    leitura dos bloqueios do ramo 2->3 (linha 28)...
    concluida.
   ramo 2->3 lido com sucesso
  ramos do no 2 lidos com sucesso
 no 2 lido com sucesso

(...)

SGAGV: leitura do ficheiro 'mesas.txt' com a posicao relativa das mesas...
linha 10 (mesa 1)
linha 11 (mesa 2)
linha 12 (mesa 3)
linha 13 (mesa 4)
```

```
linha 14 (mesa 5)
linha 15 (mesa 6)
linha 16 (mesa 7)
linha 17 (mesa 8)
linha 18 (mesa 9)

(...)

SPAGV: leitura do ficheiro 'nintrtar.txt' c/ nos e probab. de introducao de
AGV's...
linha 10 (no 101)
linha 12 (no 102)
linha 14 (no 103)
linha 16 (no 104)
linha 18 (no 105)
linha 20 (no 106)
linha 22 (no 115)

(...)

SPAGV: verifica somatorio das probab. dos nos de introducao...
Verificacao concluida (soma=1.000).

SPAGV: AGV 1 no estado DISABLED criado
SPAGV: AGV 2 no estado DISABLED criado
SPAGV: AGV 3 no estado DISABLED criado
SPAGV: AGV 4 no estado DISABLED criado
SPAGV: AGV 5 no estado DISABLED criado

CEPA: o proximo carreg. podera ter lugar dentro de 71428.066 s

CRMP: chegada de um novo lote de 300 paletes (t=0.000)
  proximo lote chegara em t=10008.784

CRMP: inicio da recepcao de uma palete (t=0.000)
  chegara a zona das filas de saida em t=38.951

CRMP: inicio da recepcao de uma palete (t=0.000)
  chegara a zona das filas de saida em t=35.315

SGAGV: fila de carga 5 tem capacidade igual a 4

SGAGV: fila de carga 6 tem capacidade igual a 4

SGAGV: fila de carga 21 tem capacidade igual a 4

SGAGV: fila de carga 22 tem capacidade igual a 4

SGAGV: fila de carga 16 tem capacidade igual a 2

(...)

SGM: notifica o SGAGV da colocacao no estado ENABLED do AGV 2, enviando:
     0.192:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ENABLE_AGV,2,NAO_ALOCA_AGV,-,-)

SGAGV: recepcao da mensagem ENABLE_AGV:
     0.192:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ENABLE_AGV,2,NAO_ALOCA_AGV,-,-)

SGAGV: notificacao da passagem do AGV 2 DISABLED ao estado ENABLED,
 atraves do envio ao SGM da mensagem:
     0.192:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,CONFIRMACAO,SUCESSO,-,-,-)

SGAGV: notificacao do SPAGV da mudanca do estado do AGV 2
 de DISABLED para ENABLED, atraves do envio da mensagem:
     0.192:(id_agv,tipo_mens,p1,p2)=(2,INF_EST_AGV,ENABLED,-)
```

```
SPAGV: AGV 2 DISABLED passa a estar ENABLED apos a recepcao de:
     0.192:(id_agv,tipo_mens,p1,p2)=(2,INF_EST_AGV,ENABLED,-)

SGM: AGV 2 passa ao estado ENABLED apos a recepcao de:
     0.192:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,CONFIRMACAO,SUCESSO,-,-,-)

SGM: notifica o SGAGV da colocacao no estado ENABLED do AGV 5, enviando:
     0.346:(id_mens,tipo_mens,p1,p2,p3,p4)=(1,ENABLE_AGV,5,NAO_ALOCA_AGV,-,-)

SGAGV: recepcao da mensagem ENABLE_AGV:
     0.346:(id_mens,tipo_mens,p1,p2,p3,p4)=(1,ENABLE_AGV,5,NAO_ALOCA_AGV,-,-)

SGAGV: notificacao da passagem do AGV 5 DISABLED ao estado ENABLED,
 atraves do envio ao SGM da mensagem:
     0.346:(id_mens,tipo_mens,p1,p2,p3,p4)=(1,CONFIRMACAO,SUCESSO,-,-,-)

SGAGV: notificacao do SPAGV da mudanca do estado do AGV 5
 de DISABLED para ENABLED, atraves do envio da mensagem:
     0.346:(id_agv,tipo_mens,p1,p2)=(5,INF_EST_AGV,ENABLED,-)

SPAGV: AGV 5 DISABLED passa a estar ENABLED apos a recepcao de:
     0.346:(id_agv,tipo_mens,p1,p2)=(5,INF_EST_AGV,ENABLED,-)

SGM: AGV 5 passa ao estado ENABLED apos a recepcao de:
     0.346:(id_mens,tipo_mens,p1,p2,p3,p4)=(1,CONFIRMACAO,SUCESSO,-,-,-)

SGM: notifica o SGAGV da colocacao no estado ENABLED do AGV 1, enviando:
     0.877:(id_mens,tipo_mens,p1,p2,p3,p4)=(2,ENABLE_AGV,1,NAO_ALOCA_AGV,-,-)

SGAGV: recepcao da mensagem ENABLE_AGV:
     0.877:(id_mens,tipo_mens,p1,p2,p3,p4)=(2,ENABLE_AGV,1,NAO_ALOCA_AGV,-,-)

SGAGV: notificacao da passagem do AGV 1 DISABLED ao estado ENABLED,
 atraves do envio ao SGM da mensagem:
     0.877:(id_mens,tipo_mens,p1,p2,p3,p4)=(2,CONFIRMACAO,SUCESSO,-,-,-)

SGAGV: notificacao do SPAGV da mudanca do estado do AGV 1
 de DISABLED para ENABLED, atraves do envio da mensagem:
     0.877:(id_agv,tipo_mens,p1,p2)=(1,INF_EST_AGV,ENABLED,-)

SPAGV: AGV 1 DISABLED passa a estar ENABLED apos a recepcao de:
     0.877:(id_agv,tipo_mens,p1,p2)=(1,INF_EST_AGV,ENABLED,-)

SGM: AGV 1 passa ao estado ENABLED apos a recepcao de:
     0.877:(id_mens,tipo_mens,p1,p2,p3,p4)=(2,CONFIRMACAO,SUCESSO,-,-,-)

SGM: notifica o SGAGV da colocacao no estado ENABLED do AGV 4, enviando:
     1.172:(id_mens,tipo_mens,p1,p2,p3,p4)=(3,ENABLE_AGV,4,NAO_ALOCA_AGV,-,-)

SGAGV: recepcao da mensagem ENABLE_AGV:
     1.172:(id_mens,tipo_mens,p1,p2,p3,p4)=(3,ENABLE_AGV,4,NAO_ALOCA_AGV,-,-)

SGAGV: notificacao da passagem do AGV 4 DISABLED ao estado ENABLED,
 atraves do envio ao SGM da mensagem:
     1.172:(id_mens,tipo_mens,p1,p2,p3,p4)=(3,CONFIRMACAO,SUCESSO,-,-,-)

SGAGV: notificacao do SPAGV da mudanca do estado do AGV 4
 de DISABLED para ENABLED, atraves do envio da mensagem:
     1.172:(id_agv,tipo_mens,p1,p2)=(4,INF_EST_AGV,ENABLED,-)

SPAGV: AGV 4 DISABLED passa a estar ENABLED apos a recepcao de:
     1.172:(id_agv,tipo_mens,p1,p2)=(4,INF_EST_AGV,ENABLED,-)

SGM: AGV 4 passa ao estado ENABLED apos a recepcao de:
```

```
      1.172:(id_mens,tipo_mens,p1,p2,p3,p4)=(3,CONFIRMACAO,SUCESSO,-,-,-)

 SGM: notifica o SGAGV da colocacao no estado ENABLED do AGV 3, enviando:
      3.263:(id_mens,tipo_mens,p1,p2,p3,p4)=(4,ENABLE_AGV,3,NAO_ALOCA_AGV,-,-)

 SGAGV: recepcao da mensagem ENABLE_AGV:
      3.263:(id_mens,tipo_mens,p1,p2,p3,p4)=(4,ENABLE_AGV,3,NAO_ALOCA_AGV,-,-)

 SGAGV: notificacao da passagem do AGV 3 DISABLED ao estado ENABLED,
  atraves do envio ao SGM da mensagem:
      3.263:(id_mens,tipo_mens,p1,p2,p3,p4)=(4,CONFIRMACAO,SUCESSO,-,-,-)

 SGAGV: notificacao do SPAGV da mudanca do estado do AGV 3
  de DISABLED para ENABLED, atraves do envio da mensagem:
      3.263:(id_agv,tipo_mens,p1,p2)=(3,INF_EST_AGV,ENABLED,-)

 SPAGV: AGV 3 DISABLED passa a estar ENABLED apos a recepcao de:
      3.263:(id_agv,tipo_mens,p1,p2)=(3,INF_EST_AGV,ENABLED,-)

 SGM: AGV 3 passa ao estado ENABLED apos a recepcao de:
      3.263:(id_mens,tipo_mens,p1,p2,p3,p4)=(4,CONFIRMACAO,SUCESSO,-,-,-)

 SPAGV: Notifica introd. AGV 1, enviando a mensagem:
      7.990:(id_agv,tipo_mens,p1,p2)=(1,INTRODUCAO,134,1)

 SGAGV: recepcao da mensagem do SPAGV:
      7.990:(id_agv,tipo_mens,p1,p2)=(1,INTRODUCAO,134,1)

 SGAGV: notificacao do SPAGV sobre o sucesso da introducao do AGV 1,
  atraves do envio da mensagem:
      7.990:(id_agv,tipo_mens,p1,p2)=(1,ESTADO_INTR,AGV_INTROD,-)

 SGAGV: notificacao do SGM sobre a introducao do AGV 1,
  atraves do envio da mensagem:

 7.990:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,1,PARADO,NAO_ALOCADO,DESCARREGA
 DO)

 SGM: notificacao da passagem do AGV 1 ao modo automatico,
  atraves da recepcao da mensagem:

 7.990:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,1,PARADO,NAO_ALOCADO,DESCARREGA
 DO)

 SPAGV: AGV 1 ENABLED passa a estar PARADO apos a recepcao de:
      7.990:(id_agv,tipo_mens,p1,p2)=(1,ESTADO_INTR,AGV_INTROD,-)

 SPAGV: Notifica introd. AGV 3, enviando a mensagem:
     23.416:(id_agv,tipo_mens,p1,p2)=(3,INTRODUCAO,141,1)

 SGAGV: recepcao da mensagem do SPAGV:
     23.416:(id_agv,tipo_mens,p1,p2)=(3,INTRODUCAO,141,1)

 SGAGV: notificacao do SPAGV sobre o sucesso da introducao do AGV 3,
  atraves do envio da mensagem:
     23.416:(id_agv,tipo_mens,p1,p2)=(3,ESTADO_INTR,AGV_INTROD,-)

 SGAGV: notificacao do SGM sobre a introducao do AGV 3,
  atraves do envio da mensagem:

 23.416:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,3,PARADO,NAO_ALOCADO,DESCARREG
 ADO)

 SGM: notificacao da passagem do AGV 3 ao modo automatico,
  atraves da recepcao da mensagem:
```

```
23.416:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,3,PARADO,NAO_ALOCADO,DESCARREG
ADO)

SPAGV: AGV 3 ENABLED passa a estar PARADO apos a recepcao de:
    23.416:(id_agv,tipo_mens,p1,p2)=(3,ESTADO_INTR,AGV_INTROD,-)

CRMP: nova palete pronta a ser colocada numa das filas de saida (t=35.315)

CRMP: insercao de uma palete na fila 1 (t=35.315)

CRMP: inicio da recepcao de uma palete (t=35.315)
  chegara a zona das filas de saida em t=67.811

FABRICA: envio da ordem 1->3 (t=35.315)

SGAGV: insercao de uma palete na fila 1 (Lf=1, Rf=1)

SGM: lancamento de uma nova tarefa normal atraves do envio de:
    35.315:(id_mens,tipo_mens,p1,p2,p3,p4)=(5,TAREFA_NORMAL,QUALQUER,1,3,0)

SGAGV: recepcao da mensagem relativa a uma nova tarefa:
    35.315:(id_mens,tipo_mens,p1,p2,p3,p4)=(5,TAREFA_NORMAL,QUALQUER,1,3,0)

SGAGV: nova Tarefa 5 no estado FILA inserida na Lista de Tarefas.

SGAGV: o Escalonador vai ser chamado.

escalonador do tipo NV/TMCV:
  numero de afectacoes=2
  forma lista de tarefas e lista de agvs intervenientes...
   Lista de Tarefas a Escalonar: {5}
   Lista de AGV's a Escalonar: {1,3}

  forma matriz de eficiencia...
   matriz de eficiencia:
        409 4294967295
        601 4294967295

  executa o algoritmo hungaro...

   matriz de eficiencia apos uma iteracao do Alg. Hungaro:
          0 4294966886
          0 4294966694

    numero de solucoes testadas=3

   menor numero de tracos que cobrem os zeros da matriz (passo 2):
    linhas={}; colunas={1}; num_tracos=1


   matriz de eficiencia apos uma iteracao do Alg. Hungaro:
          0         192
          0           0

    numero de solucoes testadas=5

   menor numero de tracos que cobrem os zeros da matriz (passo 2):
    linhas={2}; colunas={1}; num_tracos=2

  lista de afectacoes: (1, 1), (2, 2)
  actualiza a lista de AGVs e a fila de Tarefas...
Fim do escalonador VMP/TMCV

Lista de Tarefas:
```

```
Tarefa 5, tipo=NORMAL, prior=0, estado=PASSO1, est_antes_desp=FILA
 origem=1, destino=3, AGV=1
 ctraj1=409, traj1=135(75), 139(43), 153(57), 154(60), 121(73), 122(43), 1(58)
 ctraj2=190, traj2=2(66), 3(124)
 (t_lancam, t_mud_estado, dt_fila, dt_p1, dt_p2, dt_p3, dt_p4, num_af)=
  (35.315, 35.315, 0.000, 0.000, 0.000, 0.000, 0.000, 1)

Lista de AGV's
AGV 1, estado=(PARADO,NAO_ALOCADO,DESCARREGADO), est_mov_ant=ENABLED
 posicao=134, ult_posicao=134
 orient=1, tarefa_agv=5, num_mov_avan=0, t_mud_estado=7.990, taxa_util=100.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (0.000, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (0.877, 7.113, 0.000, 0, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (0, 0, 0, 0)
AGV 2, estado=(ENABLED,NAO_ALOCADO,-), est_mov_ant=DISABLED
 posicao= -, ult_posicao= -
 orient=0, tarefa_agv= -, num_mov_avan=0, t_mud_estado=0.192, taxa_util=100.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (0.000, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (0.192, 0.000, 0.000, 0, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (0, 0, 0, 0)
AGV 3, estado=(PARADO,NAO_ALOCADO,DESCARREGADO), est_mov_ant=ENABLED
 posicao=141, ult_posicao=141
 orient=1, tarefa_agv= -, num_mov_avan=0, t_mud_estado=23.416, taxa_util=100.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (0.000, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (3.263, 20.153, 0.000, 0, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (0, 0, 0, 0)
AGV 4, estado=(ENABLED,NAO_ALOCADO,-), est_mov_ant=DISABLED
 posicao= -, ult_posicao= -
 orient=0, tarefa_agv= -, num_mov_avan=0, t_mud_estado=1.172, taxa_util=100.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (0.000, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (1.172, 0.000, 0.000, 0, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (0, 0, 0, 0)
AGV 5, estado=(ENABLED,NAO_ALOCADO,-), est_mov_ant=DISABLED
 posicao= -, ult_posicao= -
 orient=0, tarefa_agv= -, num_mov_avan=0, t_mud_estado=0.346, taxa_util=100.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (0.000, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (0.346, 0.000, 0.000, 0, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (0, 0, 0, 0)

Lista de Tarefas que mudaram de estado: {5}

Lista de AGV's que foram reafectados: {}

SGAGV: notificacao do SGM da mudanca de estado da Tarefa 5,
 atraves do envio da mensagem:
    35.315:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,5,PASSO1,1,-)

SGAGV: tarefa com origem no ponto 1 passou ao estado PASSO1 (Lf=1, Rf=0)#

SGAGV: notificacao do SGM sobre a passagem do AGV 1 PARADO
```

```
 ao estado BLOQUEADO, atraves do envio da mensagem:

35.315:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,1,BLOQUEADO,NAO_ALOCADO,DESCAR
REGADO)

SGAGV: notificacao do SGM sobre a passagem do AGV 1 BLOQUEADO
 ao estado EX_MOVIM, atraves do envio da mensagem:

35.315:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,1,EX_MOVIM,NAO_ALOCADO,DESCARR
EGADO)

SGAGV: ordena ao SPAGV o inicio de um movim. de transl. do AGV 1,
 atraves do envio da mensagem:
    35.315:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,135)

SGAGV: envia ao SPAGV um movim. em avanco para o AGV 1,
 atraves do envio da mensagem:
    35.315:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,139)

SGM: notificacao da alteracao do estado do AGV 1,
 atraves da recepcao da mensagem:

35.315:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,1,BLOQUEADO,NAO_ALOCADO,DESCAR
REGADO)

SGM: notificacao da alteracao do estado do AGV 1,
 atraves da recepcao da mensagem:

35.315:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,1,EX_MOVIM,NAO_ALOCADO,DESCARR
EGADO)

SGM: Tarefa 5 passa ao estado PASSO1, apos a recepcao de:
    35.315:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,5,PASSO1,1,-)

SPAGV: recebe uma ordem de movimento de translaccao para o AGV 1, atraves da
mensagem:
    35.315:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,135)

SPAGV: inicio de um movimento de translaccao do AGV 1 (134->135)
       Vai demorar 7.5 segundos

SPAGV: Notifica introd. AGV 5, enviando a mensagem:
    36.266:(id_agv,tipo_mens,p1,p2)=(5,INTRODUCAO,101,1)

SGAGV: recepcao da mensagem do SPAGV:
    36.266:(id_agv,tipo_mens,p1,p2)=(5,INTRODUCAO,101,1)

SGAGV: notificacao do SPAGV sobre o sucesso da introducao do AGV 5,
 atraves do envio da mensagem:
    36.266:(id_agv,tipo_mens,p1,p2)=(5,ESTADO_INTR,AGV_INTROD,-)

SGAGV: notificacao do SGM sobre a introducao do AGV 5,
 atraves do envio da mensagem:

36.266:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,PARADO,NAO_ALOCADO,DESCARREG
ADO)

SGM: notificacao da passagem do AGV 5 ao modo automatico,
 atraves da recepcao da mensagem:

36.266:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,PARADO,NAO_ALOCADO,DESCARREG
ADO)

SPAGV: AGV 5 ENABLED passa a estar PARADO apos a recepcao de:
    36.266:(id_agv,tipo_mens,p1,p2)=(5,ESTADO_INTR,AGV_INTROD,-)
```

```
CRMP: nova palete pronta a ser colocada numa das filas de saida (t=38.951)

CRMP: insercao de uma palete na fila 2 (t=38.951)

CRMP: inicio da recepcao de uma palete (t=38.951)
  chegara a zona das filas de saida em t=98.507

FABRICA: envio da ordem 2->4 (t=38.951)

SGAGV: insercao de uma palete na fila 2. (Lf=1, Rf=1)

SGM: lancamento de uma nova tarefa normal atraves do envio de:
    38.951:(id_mens,tipo_mens,p1,p2,p3,p4)=(6,TAREFA_NORMAL,QUALQUER,2,4,0)

SGAGV: recepcao da mensagem relativa a uma nova tarefa:
    38.951:(id_mens,tipo_mens,p1,p2,p3,p4)=(6,TAREFA_NORMAL,QUALQUER,2,4,0)

SGAGV: nova Tarefa 6 no estado FILA inserida na Lista de Tarefas.

SGAGV: o Escalonador vai ser chamado.

escalonador do tipo NV/TMCV:
  numero de afectacoes=3
  forma lista de tarefas e lista de agvs intervenientes...
   Lista de Tarefas a Escalonar: {5,6}
   Lista de AGV's a Escalonar: {1,3,5}

  forma matriz de eficiencia...
   matriz de eficiencia:
        334         274 4294967295
        601         541 4294967295
        221         160 4294967295

  executa o algoritmo hungaro...

   matriz de eficiencia apos uma iteracao do Alg. Hungaro:
         60           0 4294967021
         60           0 4294966754
         61           0 4294967135

    numero de solucoes testadas=4

   menor numero de tracos que cobrem os zeros da matriz (passo 2):
    linhas={}; colunas={2}; num_tracos=1


   matriz de eficiencia apos uma iteracao do Alg. Hungaro:
          0           0 4294966961
          0           0 4294966694
          1           0 4294967075

    numero de solucoes testadas=9

   menor numero de tracos que cobrem os zeros da matriz (passo 2):
    linhas={}; colunas={1,2}; num_tracos=2


   matriz de eficiencia apos uma iteracao do Alg. Hungaro:
          0           0         267
          0           0           0
          1           0         381

    numero de solucoes testadas=12
```

```
   menor numero de tracos que cobrem os zeros da matriz (passo 2):
    linhas={1,2}; colunas={2}; num_tracos=3

  lista de afectacoes: (1, 1), (2, 3), (3, 2)
  actualiza a lista de AGVs e a fila de Tarefas...
Fim do escalonador VMP/TMCV

Lista de Tarefas:
Tarefa 5, tipo=NORMAL, prior=0, estado=PASSO1, est_antes_desp=PASSO1
 origem=1, destino=3, AGV=1
 ctraj1=334, traj1=139(43), 153(57), 154(60), 121(73), 122(43), 1(58)
 ctraj2=190, traj2=2(66), 3(124)
 (t_lancam, t_mud_estado, dt_fila, dt_p1, dt_p2, dt_p3, dt_p4, num_af)=
  (35.315, 35.315, 0.000, 0.000, 0.000, 0.000, 0.000, 1)
Tarefa 6, tipo=NORMAL, prior=0, estado=PASSO1, est_antes_desp=FILA
 origem=2, destino=4, AGV=5
 ctraj1=160, traj1=120(60), 124(43), 2(57)
 ctraj2=275, traj2=3(124), 4(151)
 (t_lancam, t_mud_estado, dt_fila, dt_p1, dt_p2, dt_p3, dt_p4, num_af)=
  (38.951, 38.951, 0.000, 0.000, 0.000, 0.000, 0.000, 1)

Lista de AGV's
AGV 1, estado=(EX_MOVIM,NAO_ALOCADO,DESCARREGADO), est_mov_ant=BLOQUEADO
 posicao=135, ult_posicao=134
 orient=1, tarefa_agv=5, num_mov_avan=1, t_mud_estado=35.315, taxa_util=0.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (27.326, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (0.877, 7.113, 0.000, 0, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (0, 1, 0, 0)
AGV 2, estado=(ENABLED,NAO_ALOCADO,-), est_mov_ant=DISABLED
 posicao= -, ult_posicao= -
 orient=0, tarefa_agv= -, num_mov_avan=0, t_mud_estado=0.192, taxa_util=100.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (0.000, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (0.192, 0.000, 0.000, 0, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (0, 0, 0, 0)
AGV 3, estado=(PARADO,NAO_ALOCADO,DESCARREGADO), est_mov_ant=ENABLED
 posicao=141, ult_posicao=141
 orient=1, tarefa_agv= -, num_mov_avan=0, t_mud_estado=23.416, taxa_util=100.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (0.000, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (3.263, 20.153, 0.000, 0, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (0, 0, 0, 0)
AGV 4, estado=(ENABLED,NAO_ALOCADO,-), est_mov_ant=DISABLED
 posicao= -, ult_posicao= -
 orient=0, tarefa_agv= -, num_mov_avan=0, t_mud_estado=1.172, taxa_util=100.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (0.000, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (1.172, 0.000, 0.000, 0, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (0, 0, 0, 0)
AGV 5, estado=(PARADO,NAO_ALOCADO,DESCARREGADO), est_mov_ant=ENABLED
 posicao=101, ult_posicao=101
 orient=1, tarefa_agv=6, num_mov_avan=0, t_mud_estado=36.266, taxa_util=100.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (0.000, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (0.346, 35.920, 0.000, 0, 0)
```

```
   (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
  (0, 0, 0, 0)

 Lista de Tarefas que mudaram de estado: {6}

 Lista de AGV's que foram reafectados: {}

 SGAGV: notificacao do SGM da mudanca de estado da Tarefa 6,
  atraves do envio da mensagem:
     38.951:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,6,PASSO1,5,-)

 SGAGV: tarefa com origem no ponto 2 passou ao estado PASSO1 (Lf=1, Rf=0)#

 SGAGV: notificacao do SGM sobre a passagem do AGV 5 PARADO
  ao estado BLOQUEADO, atraves do envio da mensagem:

 38.951:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,BLOQUEADO,NAO_ALOCADO,DESCAR
 REGADO)

 SGAGV: notificacao do SGM sobre a passagem do AGV 5 BLOQUEADO
  ao estado EX_MOVIM, atraves do envio da mensagem:

 38.951:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,EX_MOVIM,NAO_ALOCADO,DESCARR
 EGADO)

 SGAGV: ordena ao SPAGV o inicio de um movim. de transl. do AGV 5,
  atraves do envio da mensagem:
     38.951:(id_agv,tipo_mens,p1,p2)=(5,MOVIMENTO,mOV_TRANSL,120)

 SGAGV: envia ao SPAGV um movim. em avanco para o AGV 5,
  atraves do envio da mensagem:
     38.951:(id_agv,tipo_mens,p1,p2)=(5,MOVIMENTO,MOV_TRANSL,124)

 SGM: notificacao da alteracao do estado do AGV 5,
  atraves da recepcao da mensagem:

 38.951:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,BLOQUEADO,NAO_ALOCADO,DESCAR
 REGADO)

 SGM: notificacao da alteracao do estado do AGV 5,
  atraves da recepcao da mensagem:

 38.951:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,EX_MOVIM,NAO_ALOCADO,DESCARR
 EGADO)

 SGM: Tarefa 6 passa ao estado PASSO1, apos a recepcao de:
     38.951:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,6,PASSO1,5,-)

 SPAGV: recebe uma ordemnde movimento de translaccao para o AGV 5, atraves da
 mensagem:
     38.951:(id_agv,tipo_mens,p1,p2)=(5,MOVIMENTO,MOV_TRANSL,120)

 SPAGV: inicio de um movimento de translaccao do AGV 5 (101->120)
        Vai demorar 6.0 segundos

 SPAGV: Notifica o fim do movimento do AGV 1, enviando a mensagem:
     42.815:(id_agv,tipo_mens,p1,z2)=(1,FIM_MOVIMENTO,-,-)

 SPAGV: recebe uma ordem de movimento de translaccao para o AGV 1, atraves da
 mensagem:
     42.815:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,139)

 SPAGV: iniciw de um movimento de translaccao do AGV 1 (135->139)
        Vai demorar 4.3 segundos
```

```
SGAGV: notificacao pelo SPAGV do fim de um movimento do AGV 1,
 atraves da mensagem:
    42.815:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

SGAGV: AGV 1 terminou o movimento 134->135
 vai executar o movimento 135->139

SGAGV: envia ao SPAGV um movim. em avanco para o AGV 1,
 atraves do envio da mensagem:
    42.815:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,153)

SPAGV: Notifica o fim do movimento do AGV 5, enviando a mensagem:
    44.951:(id_agv,tipo_mens,p1,p2)=(5,FIM_MOVIMENTO,-,-)

SPAGV: recebe uma ordem de movimento de translaccao para o AGV 5, atraves da
mensagem:
    44.951:(id_agv,tipo_mens,p1,p2)=(5,MOVIMENTO,MOV_TRANSL,124)

SPAGV: inicio de um movimento de translaccao do AGV 5 (120->124)
        Vai demorar 4.3 segundos

SGAGV: notificacao pelo SPAGV do fim de um movimento do AGV 5,
 atraves da mensagem:
    44.951:(id_agv,tipo_mens,p1,p2)=(5,FIM_MOVIMENTO,-,-)

SGAGV: AGV 5 terminou o movimento 101->120
 vai executar o movimento 120->124

SGAGV: envia ao SPAGV um movim. em avanco para o AGV 5,
 atraves do envio da mensagem:
    44.951:(id_agv,tipo_mens,p1,p2)=(5,MOVIMENTO,MOV_TRANSL,2)

SPAGV: Notifica o fim do movimento do AGV 1, enviando a mensagem:
    47.115:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

SPAGV: recebe uma ordem de movimento de translaccao para o AGV 1, atraves da
mensagem:
    47.115:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,153)

SPAGV: inicio de um movimento de translaccao do AGV 1 (139->153)
        Vai demorar 5.7 segundos

SGAGV: notificacao pelo SPAGV do fim de um movimento do AGV 1,
 atraves da mensagem:
    47.115:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

SGAGV: AGV 1 terminou o movimento 135->139
 vai executar o movimento 139->153

SGAGV: envia ao SPAGV um movim. em avanco para o AGV 1,
 atraves do envio da mensagem:
    47.115:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,154)

SPAGV: Notifica o fim do movimento do AGV 5, enviando a mensagem:
    49.251:(id_agv,tipo_mens,p1,p2)=(5,FIM_MOVIMENTO,-,-)

SPAGV: recebe uma ordem de movimento de translaccao para o AGV 5, atraves da
mensagem:
    49.251:(id_agv,tipo_mens,p1,p2)=(5,MOVIMENTO,MOV_TRANSL,2)

SPAGV: inicio de um movimento de translaccao do AGV 5 (124->2)
        Vai demorar 5.7 segundos

SGAGV: notificacao pelo SPAGV do fim de um movimento do AGV 5,
 atraves da mensagem:
```

```
    49.251:(id_agv,tipo_mens,p1,p2)=(5,FIM_MOVIMENTO,-,-)

 SGAGV: AGV 5 terminou o movimento 120->124
  vai executar o movimento 124->2

 SPAGV: Notifica o fim do movimento do AGV 1, enviando a mensagem:
    52.815:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

 SPAGV: recebe uma ordem de movimento de translaccao para o AGV 1, atraves da
 mensagem:
    52.815:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,154)

 SPAGV: inicio de um movimento de translaccao do AGV 1 (153->154)
        Vai demorar 6.0 segundos

 SGAGV: notificacao pelo SPAGV do fim de um movimento do AGV 1,
  atraves da mensagem:
    52.815:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

 SGAGV: AGV 1 terminou o movimento 139->153
  vai executar o movimento 153->154

 SGAGV: envia ao SPAGV um movim. em avanco para o AGV 1,
  atraves do envio da mensagem:
    52.815:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,121)

 SPAGV: Notifica o fim do movimento do AGV 5, enviando a mensagem:
    54.951:(id_agv,tipo_mens,p1,p2)=(5,FIM_MOVIMENTO,-,-)

 SGAGV: notificacao pelo SPAGV do fim de um movimento do AGV 5,
  atraves da mensagem:
    54.951:(id_agv,tipo_mens,p1,p2)=(5,FIM_MOVIMENTO,-,-)

 SGAGV: AGV 5 terminou o movimento 124->2

 SGAGV: notificacao do SGM sobre a alteracao do estado do AGV 5,
  atraves do envio da mensagem:

 54.951:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,BLOQUEADO,NAO_ALOCADO,DESCAR
 REGADO)

 SGAGV: notificacao do SGM sobre a passagem do AGV 5 BLOQUEADO
  ao estado CARGA_AGV, atraves do envio da mensagem:

 54.951:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,CARGA_AGV,NAO_ALOCADO,DESCAR
 REGADO)

 SGAGV: ordena ao SPAGV o inicio de um movim. de carga do AGV 5,
  atraves do envio da mensagem:
    54.951:(id_agv,tipo_mens,p1,p2)=(5,MOVIMENTO,MOV_CARGA,0)

 SGAGV: notificacao do SGM sobre a mudanca do estado da Tarefa 6,
  atraves do envio da mensagem:
    54.951:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,6,PASSO2,5,-)

 SGM: Tarefa 6 passa ao estado PASSO2, apos a recepcao de:
    54.951:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,6,PASSO2,5,-)

 SGM: notificacao da alteracao do estado do AGV 5,
  atraves da recepcao da mensagem:

 54.951:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,BLOQUEADO,NAO_ALOCADO,DESCAR
 REGADO)

 SGM: notificacao da alteracao do estado do AGV 5,
```

```
atraves da recepcao da mensagem:

54.951:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,CARGA_AGV,NAO_ALOCADO,DESCAR
REGADO)

SPAGV: recebe uma ordem de movimento de transf. de carga para o AGV 5, atraves da
mensagem:
    54.951:(id_agv,tipo_mens,p1,p2)=(5,MOVIMENTO,MOV_CARGA,0)

SPAGV: inicio de uma operacao de carga com o AGV 5,
 no no 2, para o lado Direito

CRMP: inicio da carga de um AGV na fila 2 (t=54.951)
  vai demorar 20.623 s

SPAGV: Notifica o fim do movimento do AGV 1, enviando a mensagem:
    58.815:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

SPAGV: recebe uma ordem de movimento de translaccao para o AGV 1, atraves da
mensagem:
    58.815:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,121)

SPAGV: inicio de um movimento de translaccao do AGV 1 (154->121)
        Vai demorar 7.3 segundos

SGAGV: notificacao pelo SPAGV do fim de um movimento do AGV 1,
 atraves da mensagem:
    58.815:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

SGAGV: AGV 1 terminou o movimento 153->154
 vai executar o movimento 154->121

SGAGV: envia ao SPAGV um movim. em avanco para o AGV 1,
 atraves do envio da mensagem:
    58.815:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,122)

SPAGV: Notifica o fim do movimento do AGV 1, enviando a mensagem:
    66.115:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

SPAGV: recebe uma ordem de movimento de translaccao para o AGV 1, atraves da
mensagem:
    66.115:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,122)

SPAGV: inicio de um movimento de translaccao do AGV 1 (121->122)
        Vai demorar 4.3 segundos

SGAGV: notificacao pelo SPAGV do fim de um movimento do AGV 1,
 atraves da mensagem:
    66.115:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

SGAGV: AGV 1 terminou o movimento 154->121
 vai executar o movimento 121->122

SGAGV: envia ao SPAGV um movim. em avanco para o AGV 1,
 atraves do envio da mensagem:
    66.115:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,1)

CRMP: nova palete pronta a ser colocada numa das filas de saida (t=67.811)

CRMP: insercao de uma palete na fila 1 (t=67.811)

CRMP: inicio da recepcao de uma palete (t=67.811)
  chegara a zona das filas de saida em t=113.484

SGAGV: insercao de uma palete na fila 1 (Lf=2, Rf=1)
```

```
SPAGV: Notifica o fim do movimento do AGV 1, enviando a mensagem:
    70.415:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

SPAGV: recebe uma ordem de movimento de translaccao para o AGV 1, atraves da
mensagem:
    70.415:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,1)

SPAGV: inicio de um movimento de translaccao do AGV 1 (122->1)
       Vai demorar 5.8 segundos

SGAGV: notificacao pelo SPAGV do fim de um movimento do AGV 1,
 atraves da mensagem:
    70.415:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

SGAGV: AGV 1 terminou o movimento 121->122
 vai executar o movimento 122->1

CRMP: fim da carga de um AGV na fila 2 (t=75.574)

SPAGV: Notifica o fim do movimento de carga do AGV 5, enviando a mensagem:
    75.574:(id_agv,tipo_mens,p1,p2)=(5,FIM_MOVIMENTO,-,-)

SGAGV: notificacao pelo SPAGV do fim de um movimento do AGV 5,
 atraves da mensagem:
    75.574:(id_agv,tipo_mens,p1,p2)=(5,FIM_MOVIMENTO,-,-)

SGAGV: saiu uma palete da fila 2 (Lf=0, Rf=0)#

SGAGV: AGV 5 terminou a operacao de carga no ponto 2

SGAGV: notificacao do SGM sobre a alteracao do estado do AGV 5,
 atraves do envio da mensagem:

75.574:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,BLOQUEADO,NAO_ALOCADO,CARREG
ADO)

SGAGV: notificacao do SGM sobre a passagem do AGV 5 BLOQUEADO
 ao estado EX_MOVIM, atraves do envio da mensagem:

75.574:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,EX_MOVIM,NAO_ALOCADO,CARREGA
DO)

SGAGV: ordena ao SPAGV o inicio de um movim. de transl. do AGV 5,
 atraves do envio da mensagem:
    75.574:(id_agv,tipo_mens,p1,p2)=(5,MOVIMENTO,MOV_TRANSL,3)

SGAGV: envia ao SPAGV um movim. em avanco para o AGV 5,
 atraves do envio da mensagem:
    75.574:(id_agv,tipo_mens,p1,p2)=(5,MOVIMENTO,MOV_TRANSL,4)

SGAGV: notificacao do SGM sobre a mudanca do estado da Tarefa 6,
 atraves do envio da mensagem:
    75.574:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,6,PASSO3,5,-)

SGM: notificacao da alteracao do estado do AGV 5,
 atraves da recepcao da mensagem:

75.574:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,BLOQUEADO,NAO_ALOCADO,CARREG
ADO)

SGM: notificacao da alteracao do estado do AGV 5,
 atraves da recepcao da mensagem:
```

```
75.574:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,EX_MOVIM,NAO_ALOCADO,CARREGA
DO)

SGM: Tarefa 6 passa ao estado PASSO3, apos a recepcao de:
    75.574:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,6,PASSO3,5,-)

SPAGV: recebe uma ordem de movimento de translaccao para o AGV 5, atraves da
mensagem:
    75.574:(id_agv,tipo_mens,p1,p2)=(5,MOVIMENTO,MOV_TRANSL,3)

SPAGV: inicio de um movimento de translaccao do AGV 5 (2->3)
        Vai demorar 2.4 segundos

SPAGV: Notifica o fim do movimento do AGV 1, enviando a mensagem:
    76.215:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

SGAGV: notificacao pelo SPAGV do fim de um movimento do AGV 1,
 atraves da mensagem:
    76.215:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

SGAGV: AGV 1 terminou o movimento 122->1

SGAGV: notificacao do SGM sobre a alteracao do estado do AGV 1,
 atraves do envio da mensagem:

76.215:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,1,BLOQUEADO,NAO_ALOCADO,DESCAR
REGADO)

SGAGV: notificacao do SGM sobre a passagem do AGV 1 BLOQUEADO
 ao estado CARGA_AGV, atraves do envio da mensagem:

76.215:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,1,CARGA_AGV,NAO_ALOCADO,DESCAR
REGADO)

SGAGV: ordena ao SPAGV o inicio de um movim. de carga do AGV 1,
 atraves do envio da mensagem:
    76.215:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_CARGA,0)

SGAGV: notificacao do SGM sobre a mudanca do estado da Tarefa 5,
 atraves do envio da mensagem:
    76.215:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,5,PASSO2,1,-)

SGM: Tarefa 5 passa ao estado PASSO2, apos a recepcao de:
    76.215:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,5,PASSO2,1,-)

SGM: notificacao da alteracao do estado do AGV 1,
 atraves da recepcao da mensagem:

76.215:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,1,BLOQUEADO,NAO_ALOCADO,DESCAR
REGADO)

SGM: notificacao da alteracao do estado do AGV 1,
 atraves da recepcao da mensagem:

76.215:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,1,CARGA_AGV,NAO_ALOCADO,DESCAR
REGADO)

SPAGV: recebe uma ordem de movimento de transf. de carga para o AGV 1, atraves da
mensagem:
    76.215:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_CARGA,0)

SPAGV: inicio de uma operacao de carga com o AGV 1,
 no no 1, para o lado Direito
```

```
CRMP: inicio da carga de um AGV na fila 1 (t=76.215)
  vai demorar 20.547 s

SPAGV: Notifica o fim do movimento do AGV 5, enviando a mensagem:
    77.974:(id_agv,tipo_mens,p1,p2)=(5,FIM_MOVIMENTO,-,-)

SPAGV: recebe uma ordem de movimento de translaccao para o AGV 5, atraves da
mensagem:
    77.974:(id_agv,tipo_mens,p1,p2)=(5,MOVIMENTO,MOV_TRANSL,4)

SPAGV: inicio de um movimento de translaccao do AGV 5 (3->4)
       Vai demorar 5.1 segundos

SGAGV: notificacao pelo SPAGV do fim de um movimento do AGV 5,
 atraves da mensagem:
    77.974:(id_agv,tipo_mens,p1,p2)=(5,FIM_MOVIMENTO,-,-)

SGAGV: AGV 5 terminou o movimento 2->3
 vai executar o movimento 3->4

SPAGV: Notifica o fim do movimento do AGV 5, enviando a mensagem:
    83.074:(id_agv,tipo_mens,p1,p2)=(5,FIM_MOVIMENTO,-,-)

SGAGV: notificacao pelo SPAGV do fim de um movimento do AGV 5,
 atraves da mensagem:
    83.074:(id_agv,tipo_mens,p1,p2)=(5,FIM_MOVIMENTO,-,-)

SGAGV: AGV 5 terminou o movimento 3->4

SGAGV: notificacao do SGM sobre a alteracao do estado do AGV 5,
 atraves do envio da mensagem:

83.074:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,BLOQUEADO,NAO_ALOCADO,CARREG
ADO)

SGAGV: notificacao do SGM sobre a passagem do AGV 5 BLOQUEADO
 ao estado DESCARGA_AGV, atraves do envio da mensagem:

83.074:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,DESCARGA_AGV,NAO_ALOCADO,CAR
REGADO)

SGAGV: ordena ao SPAGV o inicio de um movim. de descarga do AGV 5,
 atraves do envio da mensagem:
    83.074:(id_agv,tipo_mens,p1,p2)=(5,MOVIMENTO,MOV_DESCARGA,0)

SGAGV: notificacao do SGM sobre a mudanca do estado da Tarefa 6,
 atraves do envio da mensagem:
    83.074:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,6,PASSO4,5,-)

SGM: notificacao da alteracao do estado do AGV 5,
 atraves da recepcao da mensagem:

83.074:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,BLOQUEADO,NAO_ALOCADO,CARREG
ADO)

SGM: notificacao da alteracao do estado do AGV 5,
 atraves da recepcao da mensagem:

83.074:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,5,DESCARGA_AGV,NAO_ALOCADO,CAR
REGADO)

SGM: Tarefa 6 passa ao estado PASSO4, apos a recepcao de:
    83.074:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,6,PASSO4,5,-)
```

```
SPAGV: recebe uma ordem de movimento de transf. de carga para o AGV 5, atraves da
mensagem:
    83.074:(id_agv,tipo_mens,p1,p2)=(5,MOVIMENTO,MOV_DESCARGA,0)

SPAGV: inicio de uma operacao de descarga com o AGV 5,
 no no 4, para o lado Direito

AAMP: inicio da descarga de um AGV na fila 4 (t=83.074)
  vai demorar 19.990 s

SPAGV: Notifica introd. AGV 2, enviando a mensagem:
    95.743:(id_agv,tipo_mens,p1,p2)=(2,INTRODUCAO,104,1)

SGAGV: recepcao da mensagem do SPAGV:
    95.743:(id_agv,tipo_mens,p1,p2)=(2,INTRODUCAO,104,1)

SGAGV: notificacao do SPAGV sobre o sucesso da introducao do AGV 2,
 atraves do envio da mensagem:
    95.743:(id_agv,tipo_mens,p1,p2)=(2,ESTADO_INTR,AGV_INTROD,-)

SGAGV: notificacao do SGM sobre a introducao do AGV 2,
 atraves do envio da mensagem:

95.743:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,2,PARADO,NAO_ALOCADO,DESCARREG
ADO)

SGM: notificacao da passagem do AGV 2 ao modo automatico,
 atraves da recepcao da mensagem:

95.743:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,2,PARADO,NAO_ALOCADO,DESCARREG
ADO)

SPAGV: AGV 2 ENABLED passa a estar PARADO apos a recepcao de:
    95.743:(id_agv,tipo_mens,p1,p2)=(2,ESTADO_INTR,AGV_INTROD,-)

CRMP: fim da carga de um AGV na fila 1 (t=96.763)

SPAGV: Notifica o fim do movimento de carga do AGV 1, enviando a mensagem:
    96.763:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

FABRICA: envio da ordem 1->3 (t=96.763)

SGAGV: notificacao pelo SPAGV do fim de um movimento do AGV 1,
 atraves da mensagem:
    96.763:(id_agv,tipo_mens,p1,p2)=(1,FIM_MOVIMENTO,-,-)

SGAGV: saiu uma palete da fila 1 (Lf=1, Rf=1)#

SGAGV: AGV 1 terminou a operacao de carga no ponto 1

SGAGV: notificacao do SGM sobre a alteracao do estado do AGV 1,
 atraves do envio da mensagem:

96.763:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,1,BLOQUEADO,NAO_ALOCADO,CARREG
ADO)

SGAGV: notificacao do SGM sobre a passagem do AGV 1 BLOQUEADO
 ao estado EX_MOVIM, atraves do envio da mensagem:

96.763:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,1,EX_MOVIM,NAO_ALOCADO,CARREGA
DO)

SGAGV: ordena ao SPAGV o inicio de um movim. de transl. do AGV 1,
 atraves do envio da mensagem:
    96.763:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,2)
```

```
SGAGV: envia ao SPAGV um movim. em avanco para o AGV 1,
 atraves do envio da mensagem:
    96.763:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,3)

SGAGV: notificacao do SGM sobre a mudanca do estado da Tarefa 5,
 atraves do envio da mensagem:
    96.763:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,5,PASSO3,1,-)

SGM: notificacao da alteracao do estado do AGV 1,
 atraves da recepcao da mensagem:

96.763:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,1,BLOQUEADO,NAO_ALOCADO,CARREG
ADO)

SGM: notificacao da alteracao do estado do AGV 1,
 atraves da recepcao da mensagem:

96.763:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,1,EX_MOVIM,NAO_ALOCADO,CARREGA
DO)

SGM: lancamento de uma nova tarefa normal atraves do envio de:
    96.763:(id_mens,tipo_mens,p1,p2,p3,p4)=(7,TAREFA_NORMAL,QUALQUER,1,3,0)

SGM: Tarefa 5 passa ao estado PASSO3, apos a recepcao de:
    96.763:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,5,PASSO3,1,-)

SPAGV: recebe uma ordem de movimento de translaccao para o AGV 1, atraves da
mensagem:
    96.763:(id_agv,tipo_mens,p1,p2)=(1,MOVIMENTO,MOV_TRANSL,2)

SPAGV: inicio de um movimento de translaccao do AGV 1 (1->2)
        Vai demorar 6.6 segundos

SGAGV: recepcao da mensagem relativa a uma nova tarefa:
    96.763:(id_mens,tipo_mens,p1,p2,p3,p4)=(7,TAREFA_NORMAL,QUALQUER,1,3,0)

SGAGV: nova Tarefa 7 no estado FILA inserida na Lista de Tarefas.

SGAGV: o Escalonador vai ser chamado.

escalonador do tipo NV/TMCV:
  numero de afectacoes=2
  forma lista de tarefas e lista de agvs intervenientes...
   Lista de Tarefas a Escalonar: {7}
   Lista de AGV's a Escalonar: {2,3}

  forma matriz de eficiencia...
   matriz de eficiencia:
        410 4294967295
        601 4294967295

  executa o algoritmo hungaro...

   matriz de eficiencia apos uma iteracao do Alg. Hungaro:
          0 4294966885
          0 4294966694

    numero de solucoes testadas=3

   menor numero de tracos que cobrem os zeros da matriz (passo 2):
    linhas={}; colunas={1}; num_tracos=1


   matriz de eficiencia apos uma iteracao do Alg. Hungaro:
```

```
          0         191
          0           0

    numero de solucoes testadas=5

   menor numero de tracos que cobrem os zeros da matriz (passo 2):
    linhas={2}; colunas={1}; num_tracos=2

  lista de afectacoes: (1, 1), (2, 2)
  actualiza a lista de AGVs e a fila de Tarefas...
Fim do escalonador VMP/TMCV

Lista de Tarefas:
Tarefa 5, tipo=NORMAL, prior=0, estado=PASSO3, est_antes_desp=PASSO1
 origem=1, destino=3, AGV=1
 ctraj1=0, traj1=(NULL)
 ctraj2=124, traj2=3(124)
 (t_lancam, t_mud_estado, dt_fila, dt_p1, dt_p2, dt_p3, dt_p4, num_af)=
  (35.315, 96.763, 0.000, 40.900, 20.547, 0.000, 0.000, 1)
Tarefa 6, tipo=NORMAL, prior=0, estado=PASSO4, est_antes_desp=FILA
 origem=2, destino=4, AGV=5
 ctraj1=0, traj1=(NULL)
 ctraj2=0, traj2=(NULL)
 (t_lancam, t_mud_estado, dt_fila, dt_p1, dt_p2, dt_p3, dt_p4, num_af)=
  (38.951, 83.074, 0.000, 16.000, 20.623, 7.500, 0.000, 1)
Tarefa 7, tipo=NORMAL, prior=0, estado=PASSO1, est_antes_desp=FILA
 origem=1, destino=3, AGV=2
 ctraj1=410, traj1=117(57), 118(60), 119(60), 120(72), 121(60), 122(43), 1(58)
 ctraj:=190, traj2=2(66), 3(124)
 (t_lancam, t_mud_estado, dt_fila, dt_p1, dt_p2, dt_p3, dt_p4, num_af)=
  (96.763, 96.763z 0.000, 0.000, 0.000, 0.000, 0.000, 1)

Lista de AGV's
AGV 1, estado=(EX_MOVIM,NAO_ALOCADO,CARREGADO), est_movwant=BLOQUEADO
 posicao=2, ult_posicao=1
 orient=1, tarefa_agv=5, num_mov_avan=1, t_mud_estado=96.763, taxa_util=69.218
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (27.326, 0.000, 40.900, 20.547, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (0.877, 7.113, 0.000, 1, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (1, 1, 0, 1)
AGV 2, estado=(PARADO,NAO_ALOCADO,DESCARREGADO), est_mov_ant=ENABLED
 posicao=104, ult_posicao=104
 orient=1, tarefa_agv=7, num_mov_avan=0, t_mud_estado=95.743, taxa_util=100.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (0.000, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (0.192, 95.552, 0.000, 0, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (0, 0, 0, 0)
AGV 3, estado=(PARADO,NAO_ALOCADO,DESCARREGADO), est_mov_ant=ENABLED
 posicao=141, ult_posicao=141
 orient=1, tarefa_agv= -, num_mov_avan=0, t_mud_estado=23.416, taxa_util=100.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (0.000, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (3.263, 20.153, 0.000, 0, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (0, 0, 0, 0)
AGV 4, estado=(ENABLED,NAO_ALOCADO,-), est_mov_ant=DISABLED
 posicao= -, ult_posicao= -
 orient=0, tarefa_agv= -, num_mov_avan=0, t_mud_estado=1.172, taxa_util=100.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (0.000, 0.000, 0.000, 0.000, 0.000, 0.000)
```

```
  (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
  (1.172, 0.000, 0.000, 0, 0)
   (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
  (0, 0, 0, 0)
AGV 5, estado=(DESCARGA_AGV,NAO_ALOCADO,CARREGADO), est_mov_ant=BLOQUEADO
 posicao=4, ult_posicao=4
 orient=1, tarefa_agv=6, num_mov_avan=0, t_mud_estado=83.074, taxa_util=94.265
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (2.684, 0.000, 16.000, 20.623, 7.500, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (0.346, 35.920, 0.000, 1, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (1, 1, 1, 1)

Lista de Tarefas que mudaram de estado: {7}

Lista de AGV's que foram reafectados: {}

SGAGV: notificacao do SGM da mudanca de estado da Tarefa 7,
 atraves do envio da mensagem:
    96.763:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,7,PASSO1,2,-)

SGAGV: tarefa com origem no ponto 1 passou ao estado PASSO1 (Lf=1, Rf=0)#

SGAGV: notificacao do SGM sobre a passagem do AGV 2 PARADO
 ao estado BLOQUEADO, atraves do envio da mensagem:

96.763:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,2,BLOQUEADO,NAO_ALOCADO,DESCAR
REGADO)

SGAGV: notificacao do SGM sobre a passagem do AGV 2 BLOQUEADO
 ao estado EX_MOVIM, atraves do envio da mensagem:

96.763:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,2,EX_MOVIM,NAO_ALOCADO,DESCARR
EGADO)

SGAGV: ordena ao SPAGV o inicio de um movim. de transl. do AGV 2,
 atraves do envio da mensagem:
    96.763:(id_agv,tipo_mens,p1,p2)=(2,MOVIMENTO,MOV_TRANSL,117)

SGAGV: envia ao SPAGV um movim. em avanco para o AGV 2,
 atraves do envio da mensagem:
    96.763:(id_agv,tipo_mens,p1,p2)=(2,MOVIMENTO,MOV_TRANSL,118)

SGM: notificacao da alteracao do estado do AGV 2,
 atraves da recepcao da mensagem:

96.763:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,2,BLOQUEADO,NAO_ALOCADO,DESCAR
REGADO)

SGM: notificacao da alteracao do estado do AGV 2,
 atraves da recepcao da mensagem:

96.763:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,2,EX_MOVIM,NAO_ALOCADO,DESCARR
EGADO)

SGM: Tarefa 7 passa ao estado PASSO1, apos a recepcao de:
    96.763:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,7,PASSO1,2,-)

SPAGV: recebe uma ordem de movimento de translaccao para o AGV 2, atraves da
mensagem:
    96.763:(id_agv,tipo_mens,p1,p2)=(2,MOVIMENTO,MOV_TRANSL,117)

SPAGV: inicio de um movimento de translaccao do AGV 2 (104->117)
       Vai demorar 5.7 segundos
```

```
CRMP: nova palete pronta a ser colocada numa das filas de saida (t=98.507)

CRMP: insercao de uma palete na fila 2 (t=98.507)

CRMP: inicio da recepcao de uma palete (t=98.507)
  chegara a zona das filas de saida em t=131.913

FABRICA: envio da ordem 2->4 (t=98.507)

SGAGV: insercao de uma palete na fila 2 (Lf=1, Rf=1)

SGM: lancamento de uma nova tarefa normal atraves do envio de:
    98.507:(id_mens,tipo_mens,p1,p2,p3,p4)=(8,TAREFA_NORMAL,QUALQUER,2,4,0)

SGAGV: recepcao da mensagem relativa a uma nova tarefa:
    98.507:(id_mens,tipo_mens,p1,p2,p3,p4)=(8,TAREFA_NORMAL,QUALQUER,2,4,0)

SGAGV: nova Tarefa 8 no estado FILA inserida na Lista de Tarefas.

SGAGV: o Escalonador vai ser chamado.

escalonador do tipo NV/TMCV:
  numero de afectacoes=2
  forma lista de tarefas e lista de agvs intervenientes...
   Lista de Tarefas a Escalonar: {7,8}
   Lista de AGV's a Escalonar: {2,3}

  forma matriz de eficiencia...
   matriz de eficiencia:
        353         292
        601         541

  executa o algoritmo hungaro...

   matriz de eficiencia apos uma iteracao do Alg. Hungaro:
         61           0
         60           0

    numero de solucoes testadas=3

   menor numero de tracos que cobrem os zeros da matriz (passo 2):
    linhas={}; colunas={2}; num_tracos=1


   matriz de eficiencia apos uma iteracao do Alg. Hungaro:
          1           0
          0           0

    numero de solucoes testadas=5

   menor numero de tracos que cobrem os zeros da matriz (passo 2):
    linhas={2}; colunas={2}; num_tracos=2

  lista de afectacoes: (1, 2), (2, 1)
  actualiza a lista de AGVs e a fila de Tarefas...
Fim do escalonador VMP/TMCV

Lista de Tarefas:
Tarefa 5, tipo=NORMAL, prior=0, estado=PASSO3, est_antes_desp=PASSO1
 origem=1, destino=3, AGV=1
 ctraj1=0, traj1=(NULL)
 ctraj2=124, traj2=3(124)
 (t_lancam, t_mud_estado, dt_fila, dt_p1, dt_p2, dt_p3, dt_p4, num_af)=
  (35.315, 96.763, 0.000, 40.900, 20.547, 0.000, 0.000, 1)
```

```
Tarefa 6, tipo=NORMAL, prior=0, estado=PASSO4, est_antes_desp=FILA
 origem=2, destino=4, AGV=5
 ctraj1=0, traj1=(NULL)
 ctraj2=0, traj2=(NULL)
 (t_lancam, t_mud_estado, dt_fila, dt_p1, dt_p2, dt_p3, dt_p4, num_af)=
  (38.951, 83.074, 0.000, 16.000, 20.623, 7.500, 0.000, 1)
Tarefa 7, tipo=NORMAL, prior=0, estado=PASSO1, est_antes_desp=PASSO1
 origem=1, destino=3, AGV=3
 ctraj1=601, traj1=142(60), 133(57), 134(75), 135(75), 139(43), 153(57), 154(60),
121(73), 122(43), 1(58)
 ctraj2=190, traj2=2(66), 3(124)
 (t_lancam, t_mud_estado, dt_fila, dt_p1, dt_p2, dt_p3, dt_p4, num_af)=
  (96.763, 96.763, 0.000, 0.000, 0.000, 0.000, 0.000, 2)
Tarefa 8, tipo=NORMAL, prior=0, estado=PASSO1, est_antes_desp=FILA
 origem=2, destino=4, AGV=2
 ctraj1=292, traj1=118(60), 119(60), 120(72), 124(43), 2(57)
 ctraj2=275, traj2=3(124), 4(151)
 (t_lancam, t_mud_estado, dt_fila, dt_p1, dt_p2, dt_p3, dt_p4, num_af)=
  (98.507, 98.507, 0.000, 0.000, 0.000, 0.000, 0.000, 1)

Lista de AGV's
AGV 1, estado=(EX_MOVIM,NAO_ALOCADO,CARREGADO), est_mov_ant=BLOQUEADO
 posicao=2, ult_posicao=1
 orient=1, tarefa_agv=5, num_mov_avan=1, t_mud_estado=96.763, taxa_util=69.218
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (27.326, 0.000, 40.900, 20.547, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (0.877, 7.113, 0.000, 1, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (1, 1, 0, 1)
AGV 2, estado=(EX_MOVIM,NAO_ALOCADO,DESCARREGADO), est_mov_ant=BLOQUEADO
 posicao=117, ult_posicao=104
 orient=1, tarefa_agv=8, num_mov_avan=1, t_mud_estado=96.763, taxa_util=0.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (1.019, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (0.192, 95.552, 0.000, 0, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (0, 1, 0, 0)
AGV 3, estado=(PARADO,NAO_ALOCADO,DESCARREGADO), est_mov_ant=ENABLED
 posicao=141, ult_posicao=141
 orient=1, tarefa_agv=7, num_mov_avan=0, t_mud_estado=23.416, taxa_util=100.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (0.000, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (3.263, 20.153, 0.000, 0, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (0, 0, 0, 0)
AGV 4, estado=(ENABLED,NAO_ALOCADO,-), est_mov_ant=DISABLED
 posicao= -, ult_posicao= -
 orient=0, tarefa_agv= -, num_mov_avan=0, t_mud_estado=1.172, taxa_util=100.000
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (0.000, 0.000, 0.000, 0.000, 0.000, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (1.172, 0.000, 0.000, 0, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
 (0, 0, 0, 0)
AGV 5, estado=(DESCARGA_AGV,NAO_ALOCADO,CARREGADO), est_mov_ant=BLOQUEADO
 posicao=4, ult_posicao=4
 orient=1, tarefa_agv=6, num_mov_avan=0, t_mud_estado=83.074, taxa_util=94.265
 (dt_espera,dt_bloqueado,dt_movim_descarr,dt_carga,dt_movim_carr,dt_descarga)=
 (2.684, 0.000, 16.000, 20.623, 7.500, 0.000)
 (dt_disabled,dt_enabled,dt_erro,n_cargas,n_descargas)=
 (0.346, 35.920, 0.000, 1, 0)
  (n_arr_carr,n_arr_descarr,n_mov_carr,n_mov_descarr)=
```

```
(1, 1, 1, 1)

Lista de Tarefas que mudaram de estado: {8,7}

Lista de AGV's que foram reafectados: {2}

SGAGV: ordena ao SPAGV o cancelamento de todos os movimentos em avanco do AGV 2,
 atraves do envio da mensagem:
    98.507:(id_agv,tipo_mens,p1,p2)=(2,CANCEL_MOV_AVAN,-,-)

SGAGV: notificacao do SGM da mudanca de estado da Tarefa 8,
 atraves do envio da mensagem:
    98.507:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,8,PASSO1,2,-)

SGAGV: tarefa com origem no ponto 2 passou ao estado PASSO1 (Lf=1, Rf=0)#

SGAGV: notificacao do SGM da mudanca de estado da Tarefa 7,
 atraves do envio da mensagem:
    98.507:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,7,PASSO1,3,-)

SPAGV: inicio do cancelamento de movimentos em avanco do AGV 2,
 apos a recepcao de:
    98.507:(id_agv,tipo_mens,p1,p2)=(2,CANCEL_MOV_AVAN,-,-)

SPAGV: movimento em avanco do AGV 2 cancelado, correspondente a mensagem:
(id_agv,tipo_mens,p1,p2)=(2,MOVIMENTO,MOV_TRANSL,118)

SPAGV: notifica cancelamento dos movimentos em avanco do AGV 2,
 atraves do envio de:
    98.507:(id_agv,tipo_mens,p1,p2)=(2,MOV_AVAN_CANCEL,-,-)

SGM: Tarefa 8 passa ao estado PASSO1, apos a recepcao de:
    98.507:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,8,PASSO1,2.-)

SGM: notificacao da reafectacao da Tarefa 7, que estava afectada ao AGV 2
SGM: Tarefa 7 passa ao estado PASSO1, apos a recepcao de:
    98.507:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_TAREFA,7,PASSO1,3,-)

SGAGV: notificacao do cancelamento de todos os movimentos em avanco do AGV 2,
 atraves da recepcao da mensagem do SPAGV:
    98.507:(id_agv,tipo_mens,p1,p2)=(2,MOV_AVAN_CANCEL,-,-)

SGAGV: notificacao do SGM sobre a passagem do AGV 3 PARADO
 ao estado BLOQUEADO, atraves do envio da mensagem:

98.507:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,3,BLOQUEADO,NAO_ALOCADO,DESCAR
REGADO)

SGAGV: notificacao do SGM sobre a passagem do AGV 3 BLOQUEADO
 ao estado EX_MOVIM, atraves do envio da mensagem:

98.507:(id_mens,tipo_mens,p1,p2,p3,p4)=(0,ESTADO_AGV,3,EX_MOVIM,NAO_ALOCADO,DESCARR
EGADO)

SGAGV: ordena ao SPAGV o inicio de um movim. de transl. do AGV 3,
 atraves do envio da mensagem:
    98.507:(id_agv,tipo_mens,p1,p2)=(3,MOVIMENTO,MOV_TRANSL,142)

SGAGV: envia ao SPAGV um movim. em avanco para o AGV 2,
 atraves do envio da mensagem:
    98.507:(id_agv,tipo_mens,p1,p2)=(2,MOVIMENTO,MOV_TRANSL,118)

  (...)
```

BIBLIOTECA F.E. PORTO

FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO
BIBLIOTECA
0000050860