Faculdade de Engenharia da Universidade do Porto

Departamento de Engenharia Electrotécnica e de Computadores

FERRAMENTA DE APOIO AO DESENVOLVIMENTO DE SISTEMAS DE AUTOMAÇÃO INDUSTRIAL

Rui Manuel do Amaral Lira

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 em Informática Industrial)

Dissertação realizada sob a supervisão de
Professor José António Faria,

do Departamento de Engenharia Electrotécnica e de Computadores

da Faculdade de Engenharia da Universidade do Porto

Porto, Novembro de 1998

681.5(043) LIRA/FER
Faculdade de E...
BIBLIOTECA M
N.º 44008
CDU
Data ___/___/19__

Agradecimentos

Desejo expressar os meus sinceros agradecimentos, ao Professor José António Faria, orientador desta dissertação, pelo total apoio, colaboração e disponibilidade sempre manifestados durante a sua realização.

Resumo

A vasta utilização de autómatos programáveis na implementação de sistemas de automatização industrial, torna imperativo a percepção das diferentes etapas envolvidas no desenvolvimento destes sistemas.

O trabalho desenvolvido no âmbito da presente dissertação de mestrado, teve como objectivo a concepção de um conjunto integrado de ferramentas de software de índole didáctico, de apoio ao desenvolvimento de sistemas automatizados baseados em autómato programável, que permitissem executar em PC todas as etapas do desenvolvimento destes sistemas, nomeadamente a modelação em GRAFCET do comportamento do sistema automatizado, a codificação do programa do autómato programável e por fim o teste e a validação do programa.

O núcleo destas ferramentas consiste num simulador de autómato programável que permite editar e simular a execução de programas do AP, tendo a montante um editor de GRAFCET e um editor de texto e a jusante simuladores de processos físicos.

O programa a ser executado pelo simulador pode ser implementado em lista de instruções, utilizando o editor de texto que lhe está directamente associado, ou graficamente através do editor de grafcet desenvolvido.

Uma vez que se trata de um sistema didáctico, o simulador foi dotado de uma interface, onde estão disponíveis, e passíveis de serem condicionados pelo utilizador, os vários módulos e funcionalidades implementados, nomeadamente, entradas, saídas, flags auxiliares, temporizadores e contadores. Através desta interface o utilizador pode efectuar a actuação das entradas, visualizar o estado das saídas e flags auxiliares, configurar e visualizar a evolução dos temporizadores e contadores e definir os modos de funcionamento dos temporizadores.

Como etapa final de desenvolvimento, pode-se efectuar a ligação do simulador do AP a um simulador de sistema físico, de forma a permitir simular um sistema automatizado real.

Résumée

Le travail developpé dans le cadre de cette thèse, a eu comme objectif fondamental la conception d'un ensemble intégré d'outils de logiciel, à carácter didactique, d'aide à la conception de systèmes automatisés basés en automate programmable (AP).

Ces outils pemettent d'exécuter en ordinateur pesonnel toutes les étapes du cycle de développement de ces systéms, notamment la modélisation en GRAFCET du comportment du système automatisé, le codage du programme e le test et la validation de l'application.

Le noyau de ces outils consiste dans un simulateur de l'autómate programmable qui permet d'éditer et de simuler l'exécution des programmes de l'AP. En amount de ce simulateur, il y a l'editeur de GRAFCET et, en aval, les simulateurs des processus physiques contrôlés.

Le programme a être exécuté par le simulateur peut être introduit en liste d'instructions, en utilisant un éditeur de texte qui lui est directement associé, ou graphiquemente, à travers d'un éditeur de GRAFCET qui a été développé, aussi dans le cadre de ce travail.

Une fois qu'il s'agit d'un système didactique, le simulateur a été doté d'une interface utilisateur, où sont disponibles pour manipulation les modules et fonctionnalités implémentées, notamment les entrées, sorties, temporatisateurs et compteurs. À travers cette interface, l'utilisateur peut actuer sur les entrées, visualiser l'état des sorties, configurer et visualiser l'état des compteurs, definir les modes de fonctionnement et la configuretion et visualiser l'état des temporisateurs.

À la fin du processus de développement de l'application, on peut établir la liasion entre le module de simulation de l'automate programmable et le module de simulation du processus physique et, ainsi, achever une simulation global du système automatisé.

Résumé

The project developed within this post-graduation lecture aimed at designing a set of software tools for didatic purpose, in support of the development of automated systems based on a programmable logic device (PLC), which would allow the achievement in PCs of all the steps of development of these systems, chiefly the monitoring in GRAFCET of the automated system's behaviour, the program encoding of the PLC itself and, in the end, the testing and reliability of the program.

The core of these tools is a PLC simulator which allows editting and acting the running of PLC programmes, with the previous set of an GRAFCET editor and a text editor, and any simulators of phisical process in the end.

The programme to be run by the simulator can be implemented by means of a string of commands, by using the text editor directly coupled to it, or graphically by means of the GRAFCET editor developed.

Since it is a teaching system, the simulator was designed with a interface, where eventually under the user´s monitoring.

The different modes and acomplishments developed are available, such as, inputs, outputs, auxiliar flags, timers ans counters. Through the interface the user can actuate the inputs, display the outputs and auxiliar flags state, configure and display the timers and counters at work and define the settings of the timers.

As a final step of development the simulator can be connected to a phisical system simulator which will allow the acting of a real automated system.

Índice

Capitulo 1 - Introdução 12

Capítulo 2 - Simulador do autómato programável - SimAP 22

2.1 Descrição funcional 24

2.1.1 Geração dos programas do SimAP 24

2.1.2 Edição/Execução do programa 25

2.1.3 Simulação de grandezas do autómato programável 26

2.1.4 Diagramas temporais on-line/off-line 30

2.1.5 Ligação aos simuladores de processos físicos 31

2.2 Linguagem de programação 34

2.2.1 Instruções 35

2.2.2 Sintaxe da linguagem de programação 40

2.2.3 Detecção de erros 41

2.3 Descrição da implementação 43

2.3.1 Mapa de memória 43

2.3.2 Compilação do programa do autómato programável 45

2.3.2.1 Compilação das instruções de salto para subrotina/linha do programa 48

2.3.2.2 Compilação das instruções de temporização 52

2.3.2.3 Compilação das instruções de contagem 53

2.4 Execução do programa 55

2.4.1 Execução cíclica do programa 55

2.4.2 Execução das instruções de salto 57

2.4.3 Execução das insruções de temporização/contagem 60

2.4.4 Detecção de flancos 61

2.4.5 Fluxograma 62

2.4.6 Tempo de ciclo/Diagramas temporais 63

2.5 Traçado de diagramas temporais 65

2.6 Tempo de ciclo 68
2.7 Ligação aos simuladores de processos físicos 70

Capítulo 3 - GRAFCET 75
3.1 Apresentação 76
3.1.1 Estrutura/Interpretação do GRAFCET 76

3.2 Descrição funcional 87
3.2.1 Edição do GRAFCET 88
3.2.2 Edição da estrutura de etapa/transições 89
3.2.3 Edição de acções e receptividades 95

3.3 Descrição da implementação 98
3.3.1 Representação interna do GRAFCET 98
3.3.2 Gravação/reedição do GRAFCET 107
3.3.3 Edição de símbolos 111

3.4 Geração do programa para o SimAP 119
3.4.1 Método assíncrono 119
3.4.2 Método síncrono 124

Capítulo 4 - Simuladores de sistemas físicos 132
4.1 Apresentação 133
4.2 Análise funcional 134
4.2.1 Parque de estacionamento 134
4.2.2 Célula de fabrico 137

4.3 Descrição da implementação 140
4.3.1 "Parque de estacionamento" 143

Capítulo 5 - Conclusões e perspectivas de desenvolvimento 149

Bibliografia 153

Índice de figuras

Figura 1 - Diagrama de blocos dos módulos implementados 14
Figura 2 - Interface com o utilizador 15
Figura 3 - Diagrama temporal on-line 16
Figura 4 - Diagrama temporal off-line 17
Figura 5 - Execução do programa 17
Figura 6 - Interpretação do GARFCET 18
Figura 7 - Ligação ao simulador de sistemas físicos 19
Figura 8 - Lista de atribuições 20
Figura 9 - SimAP: Interface de monitorização 25
Figura 10 - SimAP: Entradas/Saídas/Flags auxiliares 27
Figura 11 - SimAP: temporizadores 28
Figura 12 - SimAP: Configuração dos temporizadores 28
Figura 13 - SimAP: Contadores 29
Figura 14 - SimAP: Configuração do contadores 29
Figura 15 - SimAP: Diagramas temporais on-line 30
Figura 16 - SimAP: Diagramas temporais off-line 31
Figura 17 - Temporizadores: Diagrama temporal TON 37
Figura 18 - Temporizadores: Diagrama temporal TOF 38
Figura 19 - Temporizadores: Diagrama temporal TP 39
Figura 20 - Detecção de erros: Interface de monitorização 41
Figura 21 - Fluxogramas 61
Figura 22 - Diagrama temporal on-line: Monitorização 65
Figura 23 - Diagrama temporal off-line: Monitorização 66
Figura 24 - Vector de registos on-line 66
Figura 25 - Diagrama temporal on-line: Actualização de variáveis 67
Figura 26 - Fluxograma do ciclo de execução 69
Figura 27 - Ligação ao simulador de processo físico: Diagrama de blocos 71
Figura 28 - Ligação do SimAP ao simulador 73
Figura 29 - GRAFCET: Etapa 77
Figura 30 - GRAFCET: Etapa inicial 77
Figura 31 - GRAFCET: Transição 78
Figura 32 - GRAFCET: Transmissão 78

Figura 33 – GRAFCET: Ligação sequencial 79
Figura 34 – GRAFCET: "AND" convergente 80
Figura 35 – GRAFCET: "AND" divergente 81
Figura 36 – GRAFCET: "OR" convergente 81
Figura 37 – GRAFCET: "OR" divergente 82
Figura 38 – Macro etapa: Símbolo 83
Figura 39 – Macro etapa: Exemplo 83
Figura 40 – Acção contínua 84
Figura 41 – Acção condicional simples 84
Figura 42 – Acção condicional com atraso 85
Figura 43 – Acção condicional limitada 85
Figura 44 – Acção condicional memorizada 86
Figura 45 – GRAFCET: Interface com o utilizador 87
Figura 46 – GRAFCET: Tipos de ligação 89
Figura 47 – GRAFCET: Células de desenho 90
Figura 48 – Desenho de símbolos (exemplo 1) 92
Figura 49 – Desenho de símbolos (exemplo 2) 92
Figura 50 – GRAFCET: Ligação entre etapas 93
Figura 51 – GRAFCET: Ligação "AND" convergente 94
Figura 52 – GRAFCET: Varrimento horizontal/vertical 95
Figura 53 – GRAFCET: Edição de acções e receptividades 96
Figura 54 – GRAFCET: Edição de acções e receptividades (exemplo) 96
Figura 55 – GRAFCET: Identificação de células de desenho 99
Figura 56 – GRAFCET: Ligação sequencial (exemplo 1) 100
Figura 57 – GRAFCET: Ligação sequencial (exemplo 2) 102
Figura 58 – GRAFET: Ligação "AND" 103
Figura 59 – GRAFCET: Ligação "OR" 105
Figura 60 – GRAFCET: Exemplo 107
Figura 61 – GRAFCET: Ligação sequencial 112
Figura 62 – GRAFCET: "AND" convergente de duas etapas 115
Figura 63 – GRAFCET: Ligação sequencial (exemplo) 117
Figura 64 – GRAFCET: Ligação "AND" convergente e divergente 118
Figura 65 – Interpretação do GRAFCET: Exemplo 120
Figura 66 – GRAFCET: Estado de conflito 122

Figura 67 – GRAFCET: Reactivação de etapas 123
Figura 68 – Modelo GRAFCET 129
Figura 69 – Parque de estacionamento: Interface com o utilizador 134
Figura 70 – Parque de estacionamento: Análise funcional 136
Figura 71 - Célula de fabrico: Interface com o utilizador 137
Figura 72 – Célula de fabrico: Análise funcional 138
Figura 73 – Execução da aplicação simulador: Abertura da cancela 140
Figura 74 – Execução da aplicação simulador: Diagrama de estados (1ª abordagem) 141
Figura 75 - Execução da aplicação simulador: Abertura da cancela 141
Figura 76 – Execução da aplicação simulador: Diagrama de estados (2ª abordagem) 141
Figura 77 – Simulador "Parque de estacionamento": veículos 144
Figura 78 – Parque de estacionamento 144
Figura 79 – Semáforos 145
Figura 80 – Cancelas 146

Índice de tabelas

Tabela 1 - Vector de registos em memória 44
Tabela 2 - Vector de endereços da função 46
Tabela 3 - Exemplo 1: Código fonte 47
Tabela 4 - Exemplo 1: Código executável 47
Tabela 5 - Instrução de salto 48
Tabela 6 - Exemplo 2: Código fonte 48
Tabela 7 - Exemplo 2: Código executável 48
Tabela 8 - Instrução de salto para subrotina: Código fonte 49
Tabela 9 - Instrução de salto para subrotina: Código executável 49
Tabela 10 - Instrução de salto para linha do programa: Código fonte 49
Tabela 11 - Instrução de salto para linha do programa: Código executável 49
Tabela 12 - Exemplo de programa 51
Tabela 13 - Vector de registos em memória para subrotinas 51
Tabela 14 - Vector de registos em memória para labels 52
Tabela 15 - Temporização: Inicialização 52
Tabela 16 - Temporização: Sintaxe 52
Tabela 17 - Temporização: código executável 53
Tabela 18 - Contagem: Sintaxe 53
Tabela 19 - Contagem: código executável 54
Tabela 20 -Leitura de linhas do programa 55
Tabela 21 - Função "Load" 56
Tabela 22 - Programa: Exemplo 58
Tabela 23 - Código executável resultante 58
Tabela 24 - Vector de registo em memória para subrotinas 59
Tabela 25 - Vector de registo em memória para labels 59
Tabela 26 - Função "Fim_programa" 60
Tabela 27 - Temporização: código do programa 60
Tabela 28 - GRAFCET: Estrutura interna de dados 102
Tabela 29 - GRFACET: Estrutura interna de dados da ligação !AND" 104
Tabela 30 - GRAFCET: Estrutura interna de dados da ligação "OR" 106
Tabela 31 - GRAFCET: Ficheiro guardado em disco 109
Tabela 32 - Estrutura de dados da tabela de registos 112

Tabela 33 - Estrutura de dados da tabela etapa-transição 112
Tabela 34 - Tabela tabela de registos 113
Tabela 35 - Tabela etapa-transição 113
Tabela 36 - Execução da aplicação 147
Tabela 37 - Parque de estacionamento: Notificação e actualização de variáveis 148

Capítulo 1

Introdução

1. Introdução

A vasta utilização de autómatos programáveis na implementação de sistemas de automatização industrial, torna imperativo a percepção das diferentes etapas envolvidas no desenvolvimento destes sistemas.

O trabalho desenvolvido no âmbito da presente dissertação de mestrado, teve como objectivo a concepção de um conjunto integrado de ferramentas de software de índole didáctico, de apoio ao desenvolvimento de sistemas automatizados baseados em autómato programável, que permitissem executar em PC todas as etapas do desenvolvimento destes sistemas, nomeadamente a modelação em GRAFCET do comportamento do sistema automatizado, a codificação do programa do autómato programável e por fim o teste e a validação do programa.

O núcleo destas ferramentas consiste num simuldor de autómato programável que permite editar e simular a execução de programas do AP, tendo a montante o editor de GRAFCET e um editor de texto e a jusante simuladores de processos físicos.

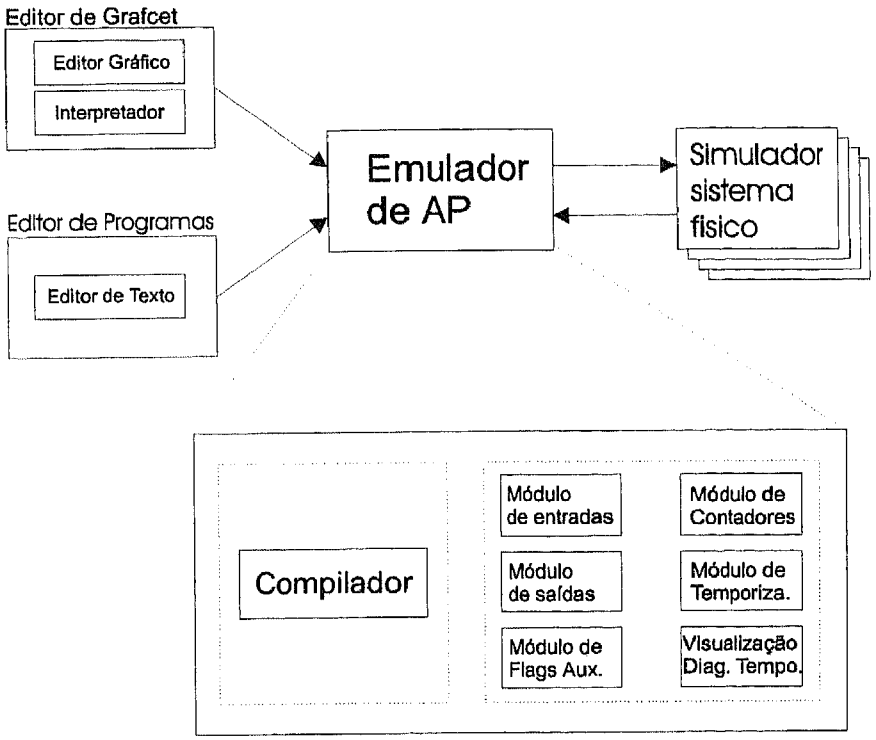


Figura 1 - Diagrama de blocos dos módulos implementados

O programa a ser executado pelo simulador pode ser implementado em lista de instruções, utilizando o editor de texto que lhe está directamente associado, ou graficamente através do editor de grafcet desenvolvido. No primeiro caso terá que haver o cuidado, por parte do utilizador, em respeitar a linguagem de programação adoptada. No segundo caso, desde que sejam respeitadas as regras de edição gráfica do grafcet, é a própria ferramenta de edição que se encarrega da interpretação e conversão do grafo realizado, para a linguagem de programação do simulador

Devido ao facto da pretensa utilização, por parte da FEUP, da ferramenta desenvolvida, adoptou-se para linguagem de programação do simulador a utilizada pelos autómatos programáveis em uso na FEUP (linguagem PL7 da Telemecanique). Desta forma, evita-se que o aluno, em especial numa fase de aprendizagem, tenha que utilizar dois tipos de linguagem de programação distintas, uma para a programação dos autómatos existentes, a outra para a utilização do simulador.

Na implementação da linguagem não foram consideradas operações aritméticas nem, de forma geral, operações orientadas ao byte/word, por questões de simplicidade de funcionamento, uma vez que, não se pretendeu simular de forma exaustiva o AP de um fabricante em particular, mas sim privilegiar a globalidade das etapas de desenvolvimento de um sistema automatizado.

Tratando-se de um sistema didáctico, torna-se preponderante que depois de efectuada a edição do programa, utilizando um dos editores descritos, a sua execução no simulador possa ser acompanhada pelo utilizador. Por este motivo, o simulador foi dotado de uma interface, onde estão disponíveis, e passíveis de serem condicionados pelo utilizador, os vários módulos e funcionalidades implementados, nomeadamente, entradas, saídas, flags auxiliares, temporizadores e contadores.

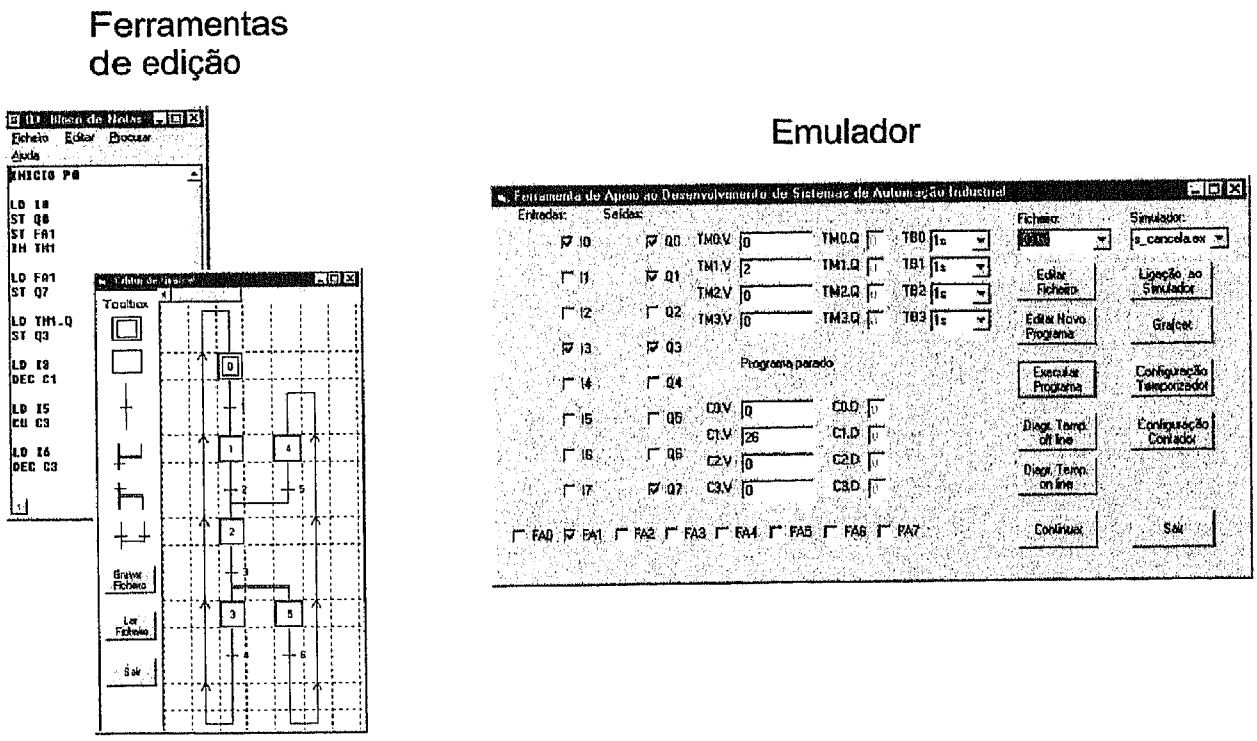


Figura 2 - Interface com o utilizador

Através desta interface o utilizador pode efectuar a actuação das entradas, visualizar o estado das saídas e flags auxiliares, configurar e visualizar a evolução

dos temporizadores e contadores e definir os modos de funcionamento dos temporizadores.

Além das funcionalidades descritas, o simulador permite efectuar a ligação aos editores de programa e aos simuladores de sistemas físicos e efectuar o traçado de diagramas temporais "on-line" e "off-line"

Apesar da interface do simulador permitir, só por si, visualizar e acompanhar o estado de todas as variáveis do sistema ao longo da execução do programa, o traçado dos diagramas temporais torna-se fundamental para verificação das relações evento/acção entre variáveis, em particular das activas ao flanco.
O traçado do diagrama temporal "on-line", representado na figura 3, permite visualizar o estado das variáveis ao longo da execução do programa.

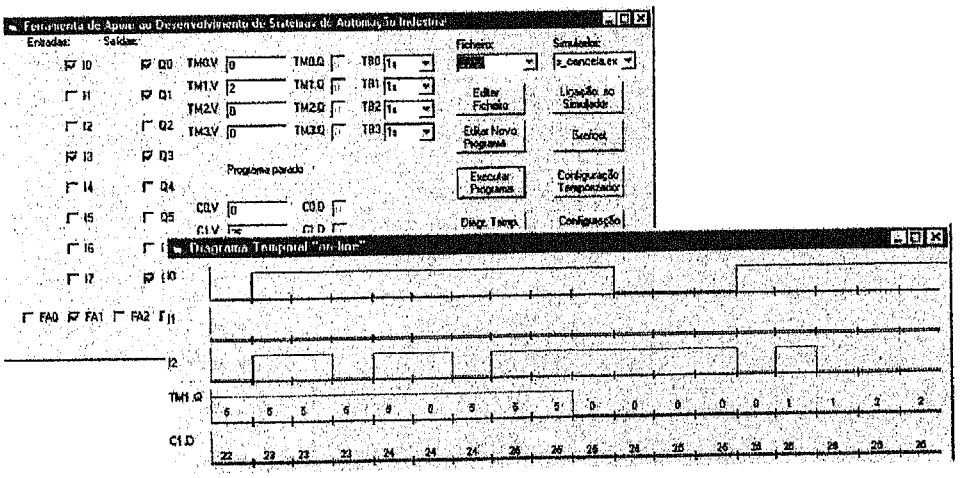


Figura 3 – Diagrama temporal "on-line"

O traçado do diagrama temporal "off-line" , representado na figura 4, permite visualizar o estado das variáveis resultantes da execução do programa.

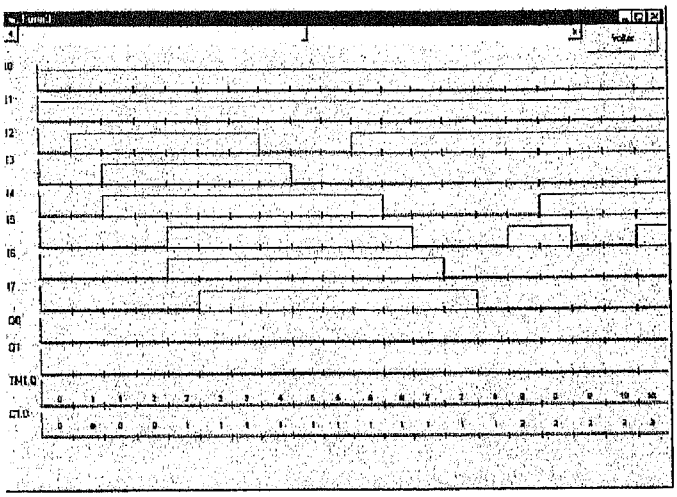


Figura 4 - Diagrama temporal "off-line"

Se o programa for realizado no editor de grafcet, a sua evolução pode ser acompanhada através do próprio grafo, por visualização directa das etapas que vão sendo activas ao longo da execução do programa.

Neste caso, é o próprio simulador que, além de executar o programa, efectua a actualização das variáveis do editor de grafcet, que permitem identificar as etapas activas.

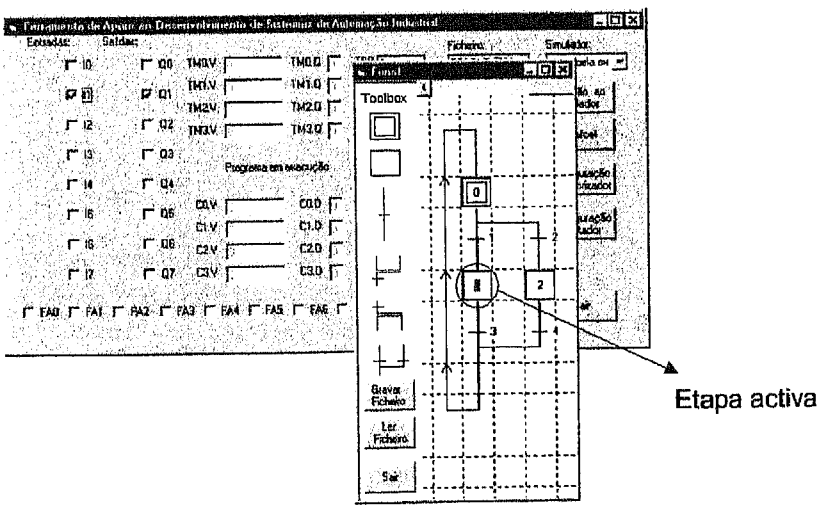


Figura 5 - Execução do programa

À semelhança do que acontece num autómato programável, pretende-se que a execução cíclica do programa no simulador seja a mais rápida possível. Para se cumprir este requisito, o código fonte proveniente dos editores é convertido num código executável, que permite ao simulador uma interpretação e execução mais rápida do programa. Esta conversão é realizado no simulador, por um compilador que realiza a leitura do programa linha a linha, efectua a verificação sintáctica e o converte numa estrutura de dados mais simples.

O código fonte fornecido ao simulador é independente do editor de programas utilizado (texto ou grafcet). No entanto, enquanto que no editor de texto o código fonte coincide com o código escrito pelo utilizador, no editor de grafcet o código fonte resulta da interpretação gráfica do grafcet que é efectuada pelo editor.

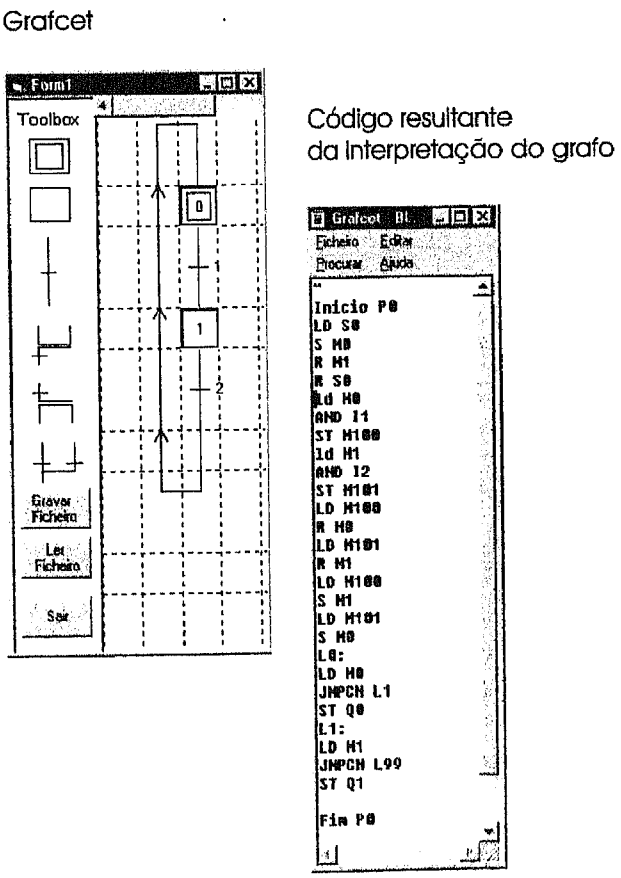


Figura 6 – Interpretação do Grafcet

Como etapa final de desenvolvimento, pode-se efectuar a ligação do simulador do autómato programável, a um simulador de sistema físico, de forma a permitir simular um sistema automatizado real, conforme se representa na figura 7.

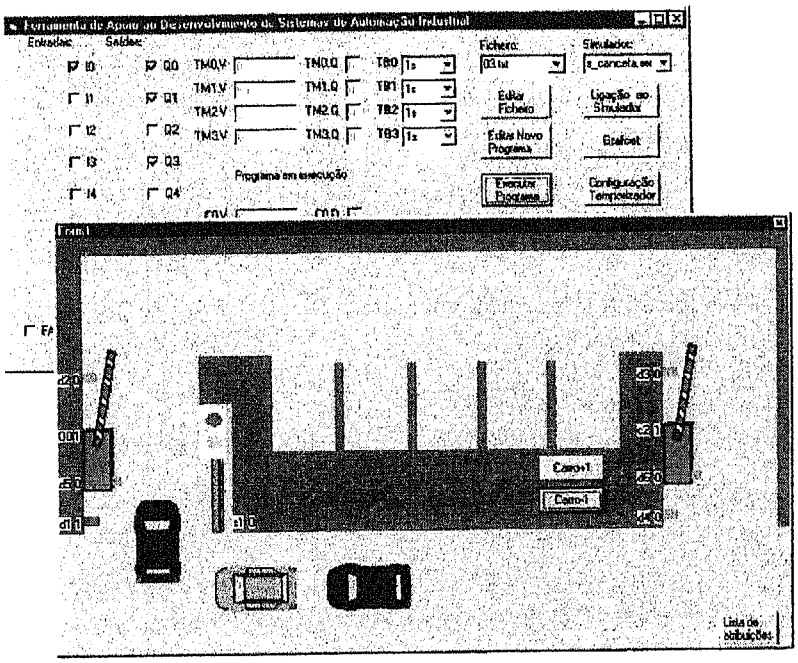


Figura 7 – Ligação ao simulador de sistemas físicos

Um simulador é constituído por um conjunto de objectos animados, aos quais correspondem um conjunto de variáveis de controlo. Cada uma destas variáveis liga a uma entrada ou saída do simulador do autómato programável de acordo com uma lista de atribuições pré definida, conforme se representa na figura 8.

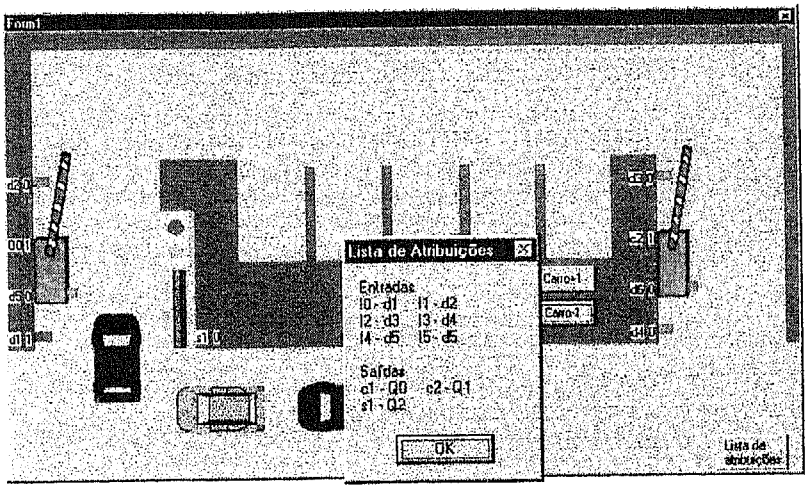


Figura 8 - Lista de atribuições

Apesar do simulador do autómato programável e o simulador do processo físico serem aplicações distintas, é efectuada uma ligação lógica entre variáveis de forma semelhante a uma ligação física entre as entradas/saídas do simulador do AP e os detectores/accionadores do simulador do processo físico. Desta forma à leitura/accionamento de uma entrada/saída do simulador do AP, corresponderá a verificação/accionamento de um detector/accionador no simulador do processo físico.

O funcionamento do sistema automatizado é facilmente acompanhado através da visualização dos seus objectos, uma vez que ao accionamento de uma variável corresponde uma mudança gráfica do objecto. No exemplo representado na figura 8, se a saída Q1 for actuada a cancela abre, caso contrário encontra-se fechada.

Cada um destes módulos será apresentado nos capítulos subsequentes, de acordo com a seguinte distribuição.

O segundo capítulo aborda o simulador do Autómato Programável. São descritas as suas funcionalidades, os módulos constituintes, a interface gráfica com o utilizador, o conjunto de instruções suportadas, o algoritmo de compilação e a interligação aos restantes módulos de software, nomeadamente, ao editor de Grafcet e aos simuladores de sistemas físicos.

O terceiro capítulo descreve o editor de Grafcet. É apresentada a interface gráfica com o utilizador e são descritas as estruturas de dados resultantes da interpretação do Grafcet, utilizadas para a geração do programa do autómato programável e para o registo e posterior edição do grafo desenvolvido.

O quarto capítulo apresenta os simuladores de sistemas físicos. É efectuada a descrição funcional de cada um dos sistemas implementados e a descrição da estrutura de dados associada a cada um dos objectos constituintes do sistema. Finaliza-se com a descrição da interligação dos sistemas ao simulador do autómato programável.

No quinto capítulo são apresentadas as conclusões e algumas sugestões para o desenvolvimento do trabalho apresentado.

Capítulo 2

Simulador do Autómato programável

2. Simulador do Autómato Programável - SimAP

O simulador do Autómato Programável, como núcleo do conjunto de ferramentas desenvolvidas, foi dotado de um conjunto de funcionalidades que permitem gerir o funcionamento de todo o sistema. Estas funcionalidades podem ser divididas em dois grupos, um relacionado com a geração de programas e outro relacionado com a sua execução.

Funcionalidades:

Geração de programas

- Ligação ao editor de texto
- Ligação ao editor de grafcet

Execução do programa

- Edição/Execução de programas
- Emulação de grandezas do AP
    - Entradas/Saídas
    - Temporizadores/Contadores
    - Flags auxiliares
- Traçado de diagramas temporais on-line/off-line
- Ligação a simuladores de processos físicos

Na descrição que se segue são apresentados os seguintes itens:

2.1 Descrição funcional

2.2 Linguagem de programação

2.3 Descrição da implementação

2.4 Execução de programas

2.5 Traçado de diagramas temporais

2.6 Tempo de ciclo

2.7 Ligação aos simuladores de processos físicos

2.1 Descrição funcional

Na descrição que se segue são apresentados os seguintes itens:

2.1.1 Geração dos programas dos SimAP

2.1.2 Edição/Execução de programas

2.1.3 Simulação de grandezas do autómato programável

2.1.4 Diagramas temporais on line/off line

2.1.5 Ligação aos simuladores de processos físicos

2.1.1 Geração dos programas do SimAP

Um dos primeiros aspectos a ser considerado no desenvolvimento desta ferramenta, foi o suporte para a edição do programa a ser executado pelo SimAP. Entre as opções disponíveis, que passavam pela utilização de um editor de texto existente ou pela criação de um editor integrado no próprio simulador, optou-se pelo primeiro caso. O motivo desta opção prende-se com o facto da aplicação desenvolvida ser para "Windows" e este já incluir um conjunto de editores de texto, (p.ex. "NotePad", "Write") que se adequam às necessidades de edição pretendidas. Desta forma, sempre que o utilizador pretender efectuar a edição de um novo programa ou de um programa existente, o simulador efectuar a abertura, por defeito, do "NotePad do "Windows". Refira-se no entanto, que apesar de existir esta opção, a edição do programa pode ser realizada utilizando um editor de texto genérico. Esta funcionalidade, conjuntamente com as restantes, está disponível no menu principal do simulador, conforme se representa na figura 9.

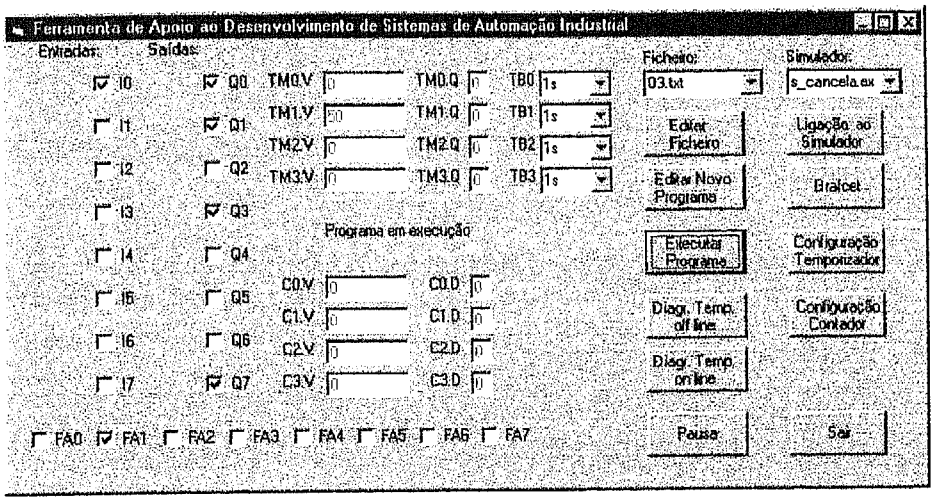


Figura 9 - SimAP: Interface de monitorização

Outra das formas de edição de programas é através da utilização do editor de grafcet, cuja apresentação será desenvolvida no capítulo seguinte desta dissertação.

2.1.2 Edição/Execução do programa

Resolvida a questão relacionada com a edição de programas, é então altura de discutir de que forma será realizada a sua execução no SimAP. Sendo o objectivo principal do SimAP, simular o funcionamento de um autómato programável, é necessário tomar em consideração a característica principal do funcionamento destes dispositivos, ou seja, a execução cíclica e rápida do programa a executar.

Em termos de execução podem-se destinguir três tipos. O ciclo síncrono para entradas e saídas, o ciclo síncrono para entradas/assíncrono para saídas e o ciclo totalmente assíncrono. No primeiro caso o autómato programável começa por efectuar a leitura das entradas para memória, que mantém até final do ciclo, em seguida executa as instruções utilizando os valores memorizados gerando saídas que guarda em memória e no fim do ciclo efectua a actualização das saídas de acordo com os valores guardados. No segundo caso, as entradas do autómato programável continuam a ser lidas e guardadas no início do ciclo, mas as saídas vão sendo

actualizadas de acordo com a evolução do programa. No terceiro caso, as entradas são lidas e as saídas actualizadas de acordo com a evolução do programa.

Para o SimAP adoptou-se o ciclo síncrono para entradas e saídas, de modo a compatibilizar a sua filosofia de funcionamento com a maioria dos autómatos actuais.

No que diz respeito à rapidez de execução, foi necessário tomar em atenção que a forma como nos é apresentado o programa proveniente do editor de texto, compromete este critério fundamental, uma vez que para se executar as instruções subjacentes a cada uma das linhas do programa, é necessário começar por interpretar cada uma das linhas de texto antes de se poder passar à sua execução. Se considerarmos que teremos que realizar este tipo de acção de forma cíclica chegaremos facilmente à conclusão, que este tipo de funcionamento se torna inviável e que a solução a adoptar, terá que ser sem dúvida outra. No caso concreto, a solução implementada consistiu na conversão (compilação) prévia do código fonte, proveniente do editor de texto, num código executável que permitiu, ao SimAP, uma maior rapidez de execução.

Refira-se, que o código executável, resultante da compilação, não extingue a necessidade de se interpretar, de forma cíclica, cada uma das linhas do novo código, à semelhança do que aconteceria se fosse utilizado directamente o código fonte. O incremento da rapidez e até a fiabilidade de execução melhora, não por eliminação da necessidade de interpretação, mas sim pela simplicidade da estrutura de dados resultante do código executável.

2.1.3 Simulação de grandezas do autómato programável

Considerando o facto, da ferramenta desenvolvida se tratar de um sistema didáctico, é fundamental que esta permita simular as grandezas fundamentais de um autómato programável. Deste modo, no menu principal do SimAP, foram disponibilizadas um conjunto de grandezas, a seguir descritas, comuns aos

autómatos programáveis, que permitem ao utilizador interagir com o sistema desenvolvido, acompanhando e condicionando a execução de programas.

Entradas/Saídas/Flags Auxiliares

No menu principal do SimAP, estão disponíveis 8 entradas, 8 saídas e 8 flags auxiliares, designadas respectivamente por I0 a I7, Q0 a Q7 e FA0 a FA7, conforme se representa na figura 10.

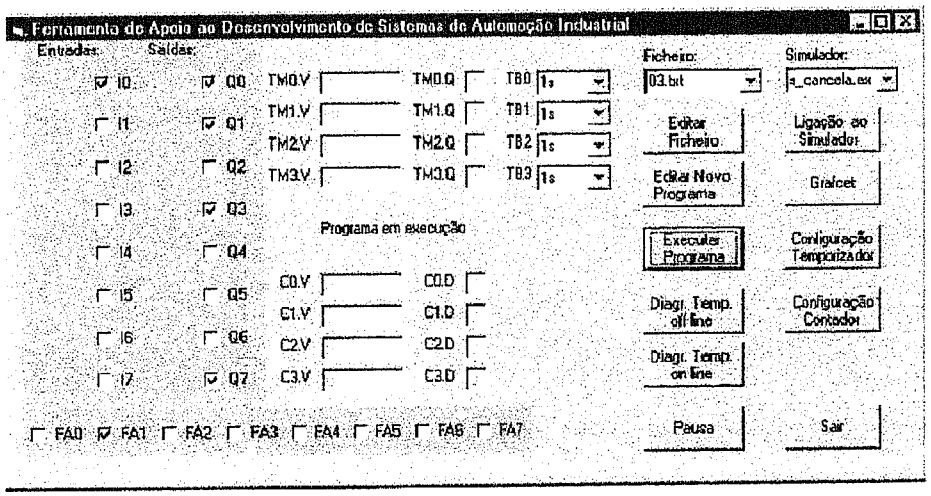


Figura 10 - SimAP: Entradas/Saídas/Flags auxiliares

As entradas podem ser actuadas pelo utilizador durante a execução do programa, de forma a permitirem a simulação de sinais provenientes de sistemas físicos. As saídas permitem a visualização de resultados decorrentes da execução do programa, constituindo uma das formas de verificação da conformidade dos resultados obtidos.

As flags auxiliares são apenas utilizadas pelo sistema para a realização de operações intermédias, encontrando-se disponíveis na interface com o utilizador para permitirem um melhor acompanhamento da execução do programa.

Temporizadores

As funções de temporização são necessárias para permitirem a realização de acções temporizadas durante a execução do programa. O SimAP, foi dotado de 4

temporizadores, encontrando-se disponíveis no menu principal o valor corrente (TM0.V...TM3.V), a saída (TM0.Q...TM3.Q) e a base de tempo (TB0...TB3) de cada temporizador, conforme se representa na figura 11.

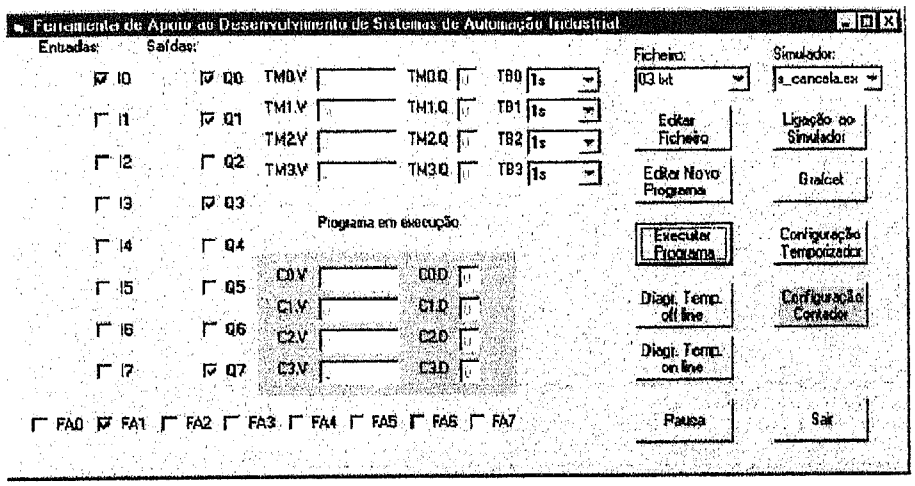


Figura 11 - SimAP: Contadores

Tendo em vista a necessidade de definir diferentes tipos de acções temporizadas, foram definidos 3 modos de funcionamento para os temporizadores: a temporização com atraso à operação: tipo TON, temporização com atraso à "desoperação": tipo TOF e utilização em monostável: tipo TP.

A definição do modo de funcionamento é realizada no menu principal através da opção "Configuração Temporizador", que permite aceder ao menu de configurações, conforme se representa na figura 12.

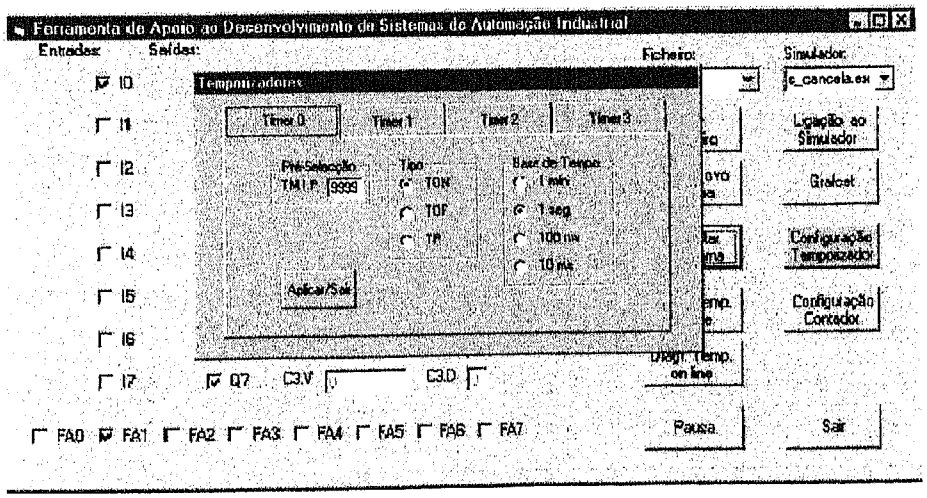


Figura 12 - SimAP: Configuração dos temporizadores

No menu de configuração dos temporizadores é possível definir o valor de pré selecção e seu modo de funcionamento.

Contadores

O SimAP possui 4 contadores, estando disponível no menu principal o valor corrente da contagem (C0.V...C3.V) e o valor lógico do bit de saída de cada um dos contadores, conforme se representa na figura 13. O bit de saída toma o valor lógico "1" quando, o valor da contagem atinge o valor de pré-slecção.

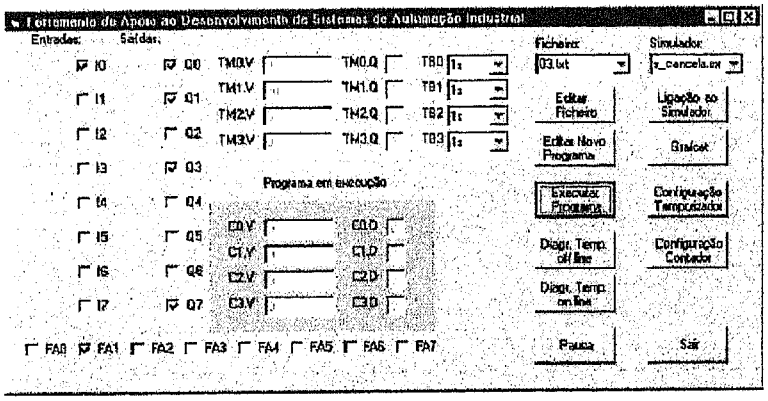


Figura 13 – SimAP: Contadores

À semelhança do que acontece com os temporizadores, através do menu principal também se pode efectuar a configuração dos contadores, que consiste na escolha do valor de pré selecção da contagem, conforme se representa na figura 14.

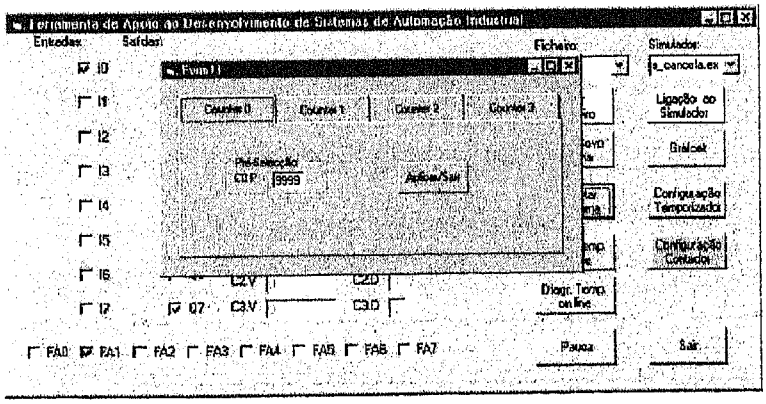


Figura 14 - SimAP: Configuração do contadores

2.1.4 Diagramas temporais on line/off line

Apesar das funcionalidades descritas anteriormente, disponíveis no menu principal do SimAP, permitirem que o utilizador acompanhe o estado de todas as variáveis do sistema ao longo da execução do programa, o traçado de diagramas temporais tornam-se bastante úteis em determinadas situações, nomeadamente no que diz respeito à verificação das relações evento/acção entre variáveis, em particular das activas ao flanco. Considerando este facto, o SimAP foi dotado da possibilidade de efectuar o traçado de diagramas temporais on-line e off-line.

Os diagramas temporais on-line permitem visualizar a variação do estado de um conjunto de variáveis ao longo da execução do programa, conforme se representa na figura 15.

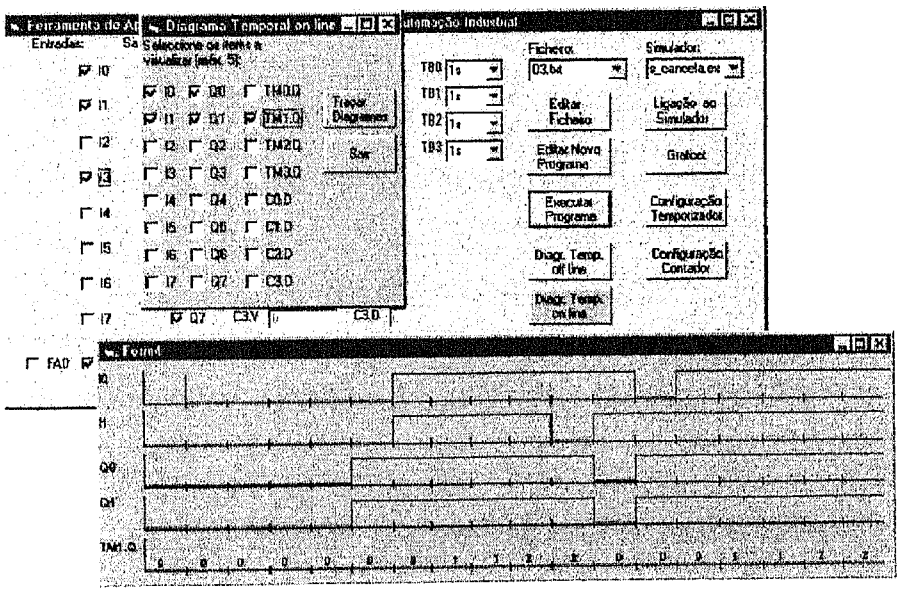


Figura 15 – SimAP: Diagrama temporal on line

Os diagramas temporais permitem visualizar a evolução das entradas I0...I7, das saídas Q0...Q7, das saídas dos temporizadores TM0.Q...TM3.Q e das saídas dos contadores C0.D...C3.D. Qualquer variação nas grandezas seleccionadas é visualizada on-line no diagrama temporal.

Os diagramas temporais off-line permitem visualizar a evolução das variáveis do sistema após a execução do programa.

Comparativamente ao traçado dos diagramas temporais on-line, permite ainda a visualização de um numero de variáveis mais abrangente. Na figura 16 representa-se o traçado do diagrama temporal off-line do conjunto de variáveis seleccionadas, I0...I7, Q0...Q1 e TM1.Q.

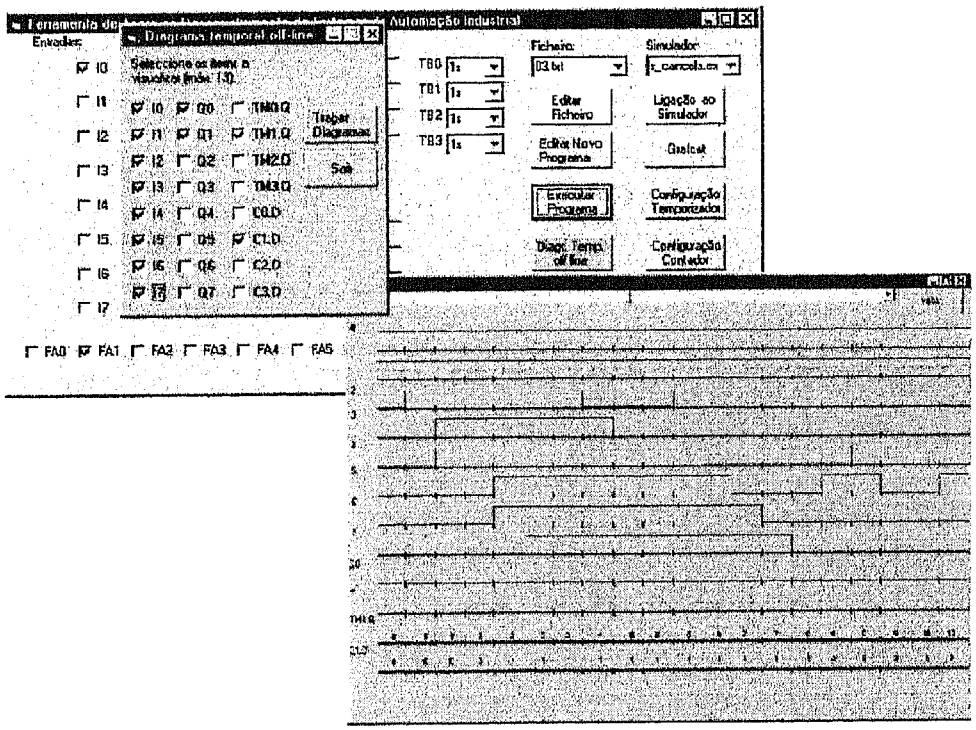


Figura 16 - SimAP: Diagrama temporal off-line

2.1.5 Ligação aos simuladores de processos físicos

A fase final do processo de desenvolvimento de um sistema de automação industrial, consiste na ligação e verificação experimental do sistema desenvolvido. Tendo presente este facto, a ferramenta que se apresenta inclui também a possibilidade de simular a ligação do autómato programável a um sistema automatizado real, completando desta forma o ciclo de desenvolvimento.

Nesta fase final podemos destinguir dois módulos de software distintos, o simulador do autómato programável e o simulador do processo físico que se pretende controlar.

Para que possa existir uma interacção entre estes dois módulos é realizada uma ligação lógica entre as variáveis de entradas/saídas do SimAP e as variáveis

correspondentes aos detectores/accionadores do simulador, de forma semelhante ao que se faria num sistema real.

Na figura 17 representa-se a ligação entre o SimAP e o simulador representativo do funcionamento de um parque de estacionamento.

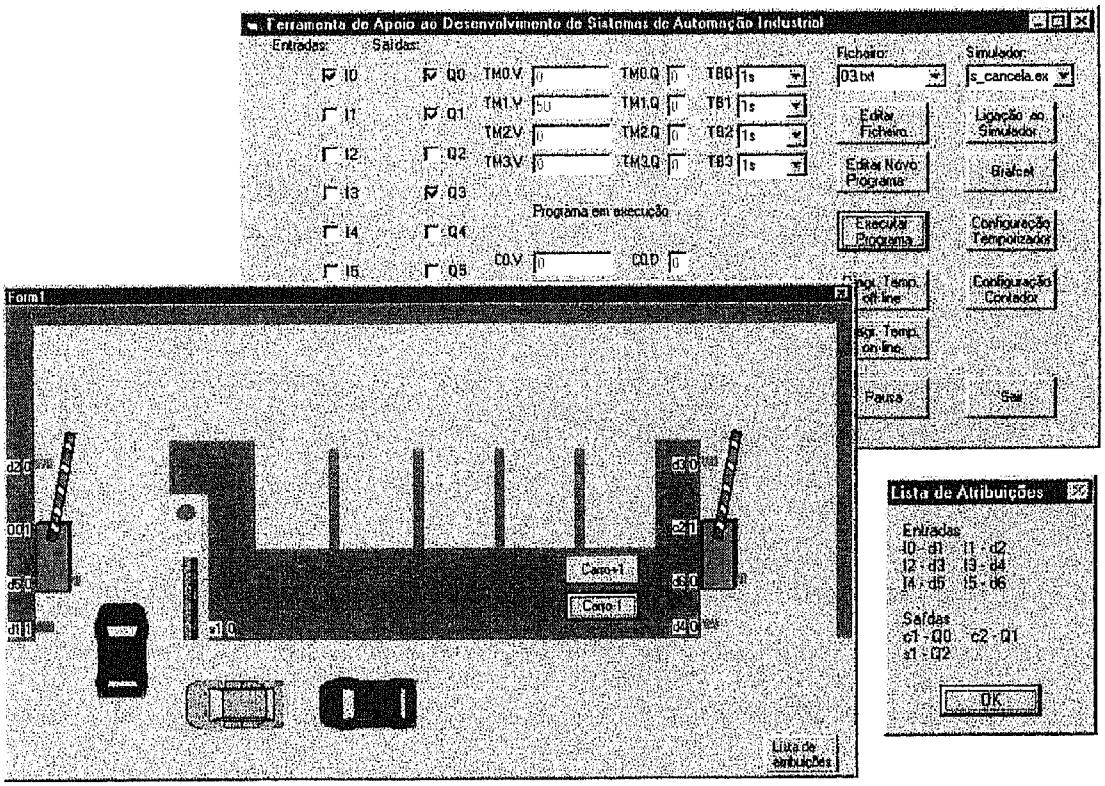


Figura 17 – Simap: Ligação ao simulador de sistema físico

A ligação entre as variáveis do autómato e do simulador são realizadas de acordo com uma lista de atribuições pré definida. No caso concreto, os 6 detectores do simulador são ligados, respectivamente às entradas I0 a I6 do SimAP, as cancelas c1 e c2 são ligadas às saídas Q0 e Q1 e semáforo S1 ligado à saída Q2.

Através da lista de atribuições, torna-se simples efectuar a correspondência entre as variáveis do SimAP e os objectos do simulador. Outra das opções seria, permitir que o utilizador defini-se livremente a ligação entre os objectos do simulador e as entradas/saídas do SimAP. A primeira vista, considerar-se-ia uma vantagem uma vez que tornaria a aplicação mais flexível, no entanto é necessário não

esquecer que tratando-se de um sistema didáctico a opção adoptada torna-se mais versátil, nomeadamente para a elaboração de enunciados de trabalhos em grupo, onde se pretende que o objecto a controlar seja o mesmo para o grupo de trabalho, apesar, de como será de esperar, as soluções de controlo adoptadas poderem ser distintas.

O realismo do funcionamento do simulador, é conseguido através da animação de cada um dos objectos que o constitui, ou seja, a actuação de uma variável no simulador corresponderá à mudança do grafismo do objecto que representa.

2.2 Linguagem de programação

Para a escolha da linguagem de programação foram consideradas como hipóteses, a criação de um linguagem proprietária à ferramenta desenvolvida e a implementação de uma linguagem já existente. Atendendo ao facto de que o software desenvolvido pretende constituir uma ferramenta de apoio à concepção de sistemas automatizados reais, haverá todo o interesse na possibilidade de se utilizar o programa desenvolvido para o simulador, no autómato que será utilizado no sistema real. Considerando este aspecto, e o intuito da FEUP utilizar este software, adoptou-se como linguagem de programação, a linguagem utilizada pelos autómatos em uso na FEUP, linguagem PL7 da "Telemecanique". Outra das vantagens desta opção, relativamente ao uso de uma pretensa linguagem proprietária, prende-se com o facto do aluno, em especial numa fase de aprendizagem, não ter que utilizar duas linguagens de programação distinta, uma para o simulador e outra para programar o autómato que pretende utilizar.

Como já se referiu, a ferramenta desenvolvida não tem como objectivo emular de forma exaustiva um autómato programável, mas sim privilegiar as etapas de desenvolvimento de um sistema automatizado. Por este motivo, da linguagem de programação adoptada, PL7 da "Telemecanique", apenas foram implementadas as instruções orientadas ao bit tendo sido excluídas as operações aritméticas e, de uma forma geral, as operações orientadas ao byte/word por questões de simplicidade de utilização.

Na descrição que se segue são apresentados os seguintes items:

2.2.1 Instruções

2.2.2 Sintaxe da linguagem de programação

2.2.3 Detecção de erros

2.2.1 Instruções

As instruções implementadas incluem:

- Bit de estado: a 1 durante o primeiro ciclo de execuçaõ do programa.
- Instruções de carregamento:
  LD - "Load", LDN - "Load not", LDR - "Load rise", LDF - "Load Fall"
- Instruções E Lógico:
  AND, ANDN, ANDR, ANDF
- Instruções OU Lógico:
  OR, ORN, ORR, ORF
- Instruções OU exclusivo:
  XOR, XORN, XORR, XORF

As instruções LDR, LDF, ANDR, ANDF, ORR, ORF, XORR e XORF referem-se a instruções activas ao flanco, nas quais a letra R e F indica respectivamente "Rise Edge" (flanco ascendente) e "Fall Edge" (flanco descendente).

- Instruções de afectação:
  ST - "Store", STN - "Store Not", S - "Set", R - "Reset"
- Instruções de negação:
  N - "Not"
- Instruções de salto:
  JMP, JMPC, JMPCN"

Esta instrução pode desencadear dois tipos de acções distintas, que dependem do tipo de operando utilizado. Se o operando for uma "Label" do tipo "L0...L99", p.ex. JMP L3, a instrução resulta num salto a linha do programa identificada pela referida "Label". Se o operando for do tipo "P1...P99", p.ex. JMP P2, a instrução resulta no salto para uma subrotina identificada por "Inicio P0...Inicio P99".

- Blocos de função temporizadores

O SimAP possui 4 temporizadores. A cada temporizador foi associado um menu de configuração que permite definir o seu modo de funcionamento e o valor de pré - selecção (TMi.P).

Cada temporizador é constituído por um registo de valor corrente (TMi.V), incrementado de acordo com a evolução do temporizador, um bit de saída (TMi.Q) e por uma base de tempo (TBi) que define a cadência de funcionamento do temporizador. Cada um destes elementos pode ser visualizado no menu principal do SimAP.

Para uma maior flexibilidade de funcionamento, o valor da base de tempo e o valor corrente do temporizador podem ser alterados durante a execução do programa, através do menu principal, o primeiro "on-line" durante o ciclo de execução, o segundo desde que seja efectuada a pausa do programa. O bit de saída pode ser apenas ser lida pelo programa .

As características dos temporizadores resumem-se a:

Numero do temporizador TMi (i=0...3)

Base de tempo, TB

Define a cadência de evolução do temporizador, com valores de 1mn (por defeito), 1s, 100ms e 10ms. Encontra-se disponível no menu principal do SimAP e pode ser alterada em qualquer instante de funcionamento.

Valor de pré - selecção, TMi.P

Valor entre 0 e 9999 a definir pelo utilizador no menu de configuração. Por defeito assume o valor 9999. Sendo o valor do atraso definido por TMi.P x BT.

Valor corrente, TMi.V

Palavra que aumenta de 0 a TMi.P durante o funcionamento do temporizador. Encontra-se disponível no menu principal do SimAP e o seu valor pode ser alterado durante a execução do programa em modo "PAUSA".

Instrução IN

No flanco ascendente no modo TON ou TP, ou flanco descendente no modo TO, arranca o temporizador.

Saída do temporizador, Q

O bit associado TMi.Q encontra-se disponível no menu principal do SimAP, a sua colocação a 1 depende do modo de funcionamento utilizado: TON, TOF ou TP.

Relativamente ao funcionamento dos temporizadores são definidos 3 modos:

Modo TON - Atraso à operação

Neste modo de funcionamento o temporizador é colocado em funcionamento no flanco ascendente da instrução "IN".
Sempre que um temporizador é colocado em funcionamento o seu valor (TMi.V) é incrementado à cadência definida pela base de tempo, que poderá ser de 1s, 100ms ou 10ms. Quando o valor do temporizador atinge o valor de pré - selecção, a sua saída (TMi.Q) é colocada a nível lógico 1.

No flanco descendente da instrução "IN", a saída e o valor do temporizador são colocados a nível lógico zero, independentemente dos seus valores actuais.
O funcionamento do temporizador é representado no diagrama temporal da figura 17.

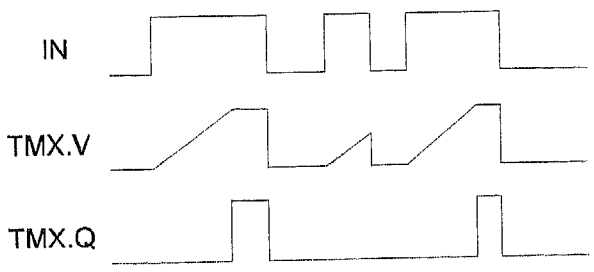


Figura 17 - Temporizadores: Diagrama temporal TON

Modo TOF – Atraso à desoperação

Neste modo de funcionamento, no flanco ascendente da instrução "IN" o valor do temporizador é colocado a zero e a sua saída colocada a nível lógico "1". Assim que surgir o flanco descendente da referida instrução o valor do temporizador começa a ser incrementado à cadência definida pela base de tempo. Quando o seu o valor atingir o valor de pré - selecção a saída TMi.Q é colocada a nível lógico zero.
O funcionamento do temporizador é representado no diagrama temporal da figura 18.

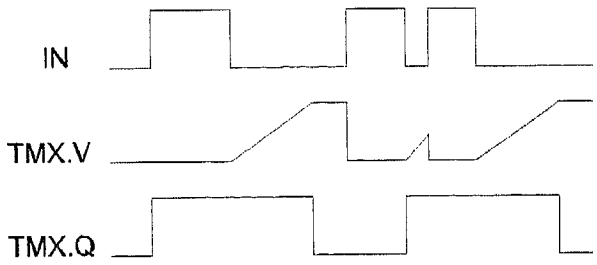


Figura 18 – Temporizadores: Diagrama temporal TOF

Modo TP - Monostável

Neste modo de funcionamento, no flanco ascendente da instrução "IN" a saída do temporizador é colocada a nível lógico 1 e o seu valor começa a ser incrementado à cadência definida pela base de tempo. Quando o seu valor atingir o valor de pré - selecção a saída TMi.D é colocada a nível lógico zero e o seu valor passa a zero, se o nível lógico da instrução "IN" for zero, caso contrário mantém o valor de pré - selecção até que surja o flanco ascendente da instrução, altura em que é colocado a zero. Depois do temporizador ser inicializado e enquanto o seu valor não atingir o valor de pré-selecção qualquer flanco ascendente ou descendente não surtirá qualquer efeito na saída ou valor do temporizador
O funcionamento do temporizador é representado no diagrama temporal da figura 19.

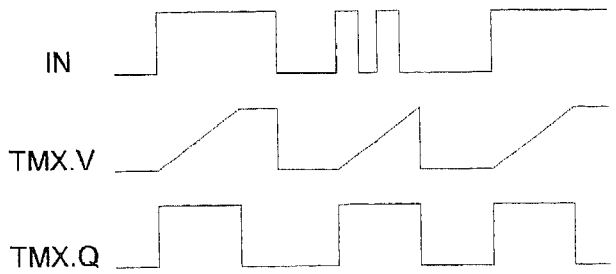


Figura 19 - Temporizadores: Diagrama temporal TP

- Blocos de função contador

O SimAP possui 4 contadores. A cada contador foi associado um menu de configuração que permite seleccionar o valor de pré - selecção (Ci.P).
Cada contador é constituído por um registo de valor corrente (Ci.V), cujo conteúdo pode ser alterado durante a pausa do programa e por um bit de saída (Ci.D), apenas para leitura, ambos disponíveis no menu principal do SimAP.

Este bloco de função permite efectuar a contagem de acontecimentos através do incremento ou decremento do valor corrente do contador. Quando o valor do contador é igual ao valor de pré - selecção, a saída é colocada a nível lógico "1", caso contrário mantém-se a nível lógico "0".
As características dos temporizadores resumem-se a:

Numero do contador: Ci (i=0...3)

Valor de pré - selecção, Ci.P

Valor entre 0 e 9999 a definir pelo utilizador no menu de configuração. Por defeito assume o valor 9999.

Valor corrente, Ci.V

Palavra incrementada ou decrementada em função das entradas CU e CD.

Entrada CU ("counter up")

Incrementa o valor de CU no flanco ascendente

Entrada CD ("counter down")

Decrementa Ci.V no flanco ascendente

Entrada R, retorno a zero

No estado 1, coloca Ci.V=0

Entrada S, pré selecção

No estado 1, coloca Ci.V=Ci.P

Saída de pré - selecção atingida D

Ci.D = 1 quando Ci.V = Ci.P

2.2.2 Sintaxe da linguagem de programação

Cabe ao utilizador a realização do programa. Para o efeito dispõe do editor de texto para escrita do programa e de um editor de GRAFCET onde pode modelar de forma gráfica o programa a implementar. No primeiro caso, o programa é compilado directamente a partir do programa editado. No segundo caso, é compilado através do ficheiro resultante da interpretação do gráfico, realizada pelo editor de GRAFCET. Em qualquer um dos casos, o ficheiro resultante tem que obedecer a uma sintaxe pré definida, de modo a poder ser correctamente interpretada aquando da conversão (compilação) do código fonte em código executável

A sintaxe adoptada pretendeu estruturar o programa de forma a permitir, ao utilizador, uma edição e interpretação simples do seu conteúdo, associada a um edição flexível, desta forma definiu-se o seguinte: a execução cíclica do programa consiste na execução do conjunto de instruções de uma rotina principal. O início e fim da rotina principal tem que ser identificada, respectivamente, por "Inicio P0" e "Fim P0". Na rotina principal podem ainda existir chamadas a subrotinas. O início e fim de subrotinas têm que ser identificadas, respectivamente por "Inicio Pxx" e "Fim Pxx", em que "xx" pode ser qualquer valor entre 1 e 99. A ordem pela qual surgem as subrotinas e rotina principal é indiferente.

Em relação às linhas de instrução impõe-se apenas que, entre o código da instrução e o operando exista pelo menos um espaço e que os comentários comecem pelo caracter ";".

Nas rotinas também é possível efectuar saltos para linhas do programa a montante ou jusante da linha actual, desde que a linha destino seja identificada pela label "Lxx", em que "xx" pode ser qualquer valor entre 0 e 99.

A ordem pela qual surge a rotina principal e subrotinas é indiferente, impondo-se apenas que o texto termine com o caracter ".", esta ultima restrição permite que o compilador detecte o fim do programa, evitando que sejam processadas um conjunto de linhas em branco que possam existir no fim do texto.

2.2.3 Detecção de erros

Para que o programa realizado possa ser executado pelo SimAP, é necessário que não existam erros de sintaxe nem de mnemónicas desconhecidas. Considerando este facto, à medida que cada uma das linhas do código fonte vai sendo convertida (compilada) em código executável, é realizada de forma simultânea a detecção de eventuais erros que possam existir. O erro ou erros detectados, vão sendo guardados em memória até que o programa editado seja totalmente compilado, altura em que são indicados ao utilizador através de uma janela de informação dedicada para o efeito, conforme se representa na figura 20.

Na informação transmitida ao utilizador é indicado o tipo de erro e a linha do programa em que se encontra.

Pretende-se desta forma dar a conhecer, de uma só vez, a totalidade de erros cometidos e possibilitar um reconhecimento fácil e rápido do local do erro.

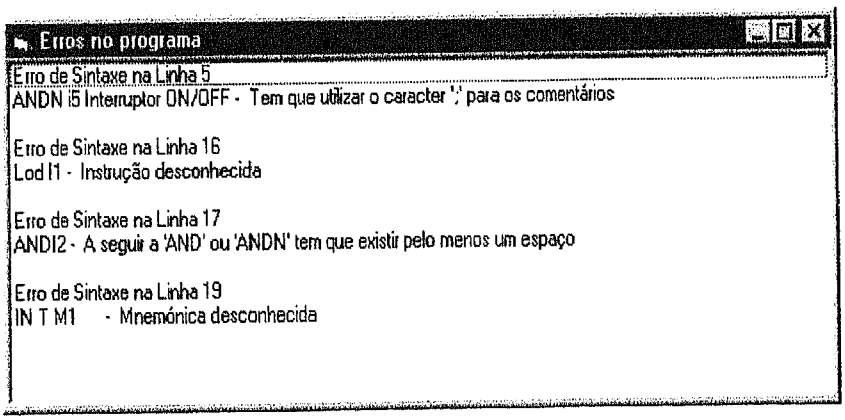


Figura 20 - Detecção de erros: Interface de monitorização

Depois de efectuada a correcção de erros, o programa é novamente compilado, é efectuada novamente a detecção de erros e, caso não haja erros, o SimAP passa à execução cíclica do programa

2.3 Descrição da implementação

Na descrição que se segue são apresentados os seguintes items:

2.3.1 Mapa de memória

2.3.2 Compilação do programa

2.3.1 Mapa de memória

O SimAP executa o programa de acordo com o ciclo síncrono para entradas e saídas, ou seja começa por efectuar a leitura das entradas para memória, que mantém até final do ciclo, em seguida executa as instruções utilizando os valores memorizados gerando saídas que guarda em memória e no fim do ciclo efectua a actualização das saídas de acordo com os valores guardados.

Como se constata, à execução do programa está subjacente a memorização das diversas variáveis utilizadas pelo AP. No SimAP, esta função é realizada através da utilização de um vector de registos, representados na tabela 1, armazenado em memória.

Tabela 1 - Vector de registos em memória

Endereço	Operando	
0	I0	Entradas
...	...	
...	...	
7	I7	
8	Q0	Saídas
...	...	
...	...	
15	Q7	
16	TM1.V	Temporizadores
...	...	
...	...	
19	TM3.V	
20	TM1.Q	Flags Temp.
...	...	
...	...	
23	TM3.Q	
24	C0.V	Contadores
...	...	
...	...	
27	C3.V	
28	C0.D	Flags Cont,.
...	...	
...	...	
31	C7.D	

Endereço	Operando	
32	FA0	Flags
...	...	
...	...	
39	FA3	
40	M_Etapas0	Etapas Grafcet
...	...	
...	...	
139	M_Etapas99	
140	M_Flags_0	FlagsEtp. Grafcet
...	...	
...	...	
238	M_Flags99	
239	Flags_Grf0	Flags Grafcet
...	...	
...	...	
246	Flags_Grf7	
247	S0	Flag sistema
248	I0_Aux	Entra. Auxiliar
...	...	
...	...	
255	I7_Aux	

Endereço	Operando	
256	Q0_Aux	Saída Auxiliar
...	...	
...	...	
263	Q7_Aux	
264	FA_Aux(0)	Flags Auxiliar
...	...	
...	...	
271	FA_Aux(7)	
272	FT_Aux(0)	Flg. Tmr. Aux
...	...	
...	...	
275	FT_Aux(3)	
276	FC_Aux(0)	Flg. Cont. Aux
...	...	
...	...	
279	FC_Aux(3)	

No vector de registos em memória, representam-se todos os registos para o armazenamento das variáveis utilizadas pelo AP durante a execução do programa. Desta forma, quando no início do programa, o autómato programável efectua a leitura das entradas para memória, em termos práticos o que acontece, é a leitura das entradas para os respectivos registos (endereço 0 a 7) indicados na tabela 1. Por exemplo, a leitura da entrada I0 para memória, resulta no armazenamento do valor lógico da referida entrada na posição (endereço) "0" do vector de registos indicado na tabela anterior . Se falarmos em termos de actualização de saídas, o que se passa é semelhante, ou seja, o valor lógico das saídas, no fim de cada ciclo, resulta da leitura dos respectivos registos em memória (endereço 8 a 15).

2.3.2 Compilação do programa do autómato programável

O compilador do autómato, converte o conjunto de instruções, do código fonte, representativas da sequência das operações lógicas a realizar, no código executável a ser utilizado pelo SimAP para a execução do programa. A necessidade da obtenção de um código executável, foi um dos principais aspectos, inicialmente considerados, quando se abordou a forma como seria executado o programa no SimAP. Definida a necessidade passou-se à implementação da sua estrutura. A ideia base inicial, consistia na obtenção de um código executável, que consistisse na chamada de funções, representativas da instrução lógica a realiza, através da utilização de um mecanismo de ponteiros semelhante à utilizada na linguagem C. No entanto, atendendo às limitações do software de desenvolvimento utilizado, *Visual Basic 5.0*, nomeadamente no que diz respeito à utilização de *ponteiros*, optou-se por uma solução, que apesar de não utilizar este conceito, e portanto não ser tão eficiente, baseia-se num princípio semelhante. Ou seja, em vez do código executável ser, p.ex, constituído por um conjunto de ponteiros que indicam directamente os endereços de memória das funções a realizar, é constituído por um conjunto de identificadores numéricos que, posteriormente na execução do programa, em conjunto com um mecanismo *Select Case*, permitem identificar a função a executar.

Para implementar esta solução associou-se a cada uma das instruções lógicas, da linguagem de programação, uma função à qual se atribuiu um identificador numérico, que designarei por endereço, de acordo com o representado na tabela 2. Cada função possui o conjunto de acções que terá que realizar, sempre que se pretender executar a instrução a que se refere.

Tabela 2 - Vector de endereços de funções

Código da instrução	Função lógica	Endereço	Código da instrução	Função lógica	Endereço
	(space)	0	XOR	Xor	16
AND	And	1	XORN	Xor not	17
OR	Or	2	STN	Store not	18
ST	Store	3	JMPC	Jump unconditional	19
IN	Initialise timer	4	JMPCN	Jump not conditional	20
INC	Increment counter	6	R Cx	Reset counter	21
DEC	Decrement counter	7	LDR	Load rise accumulator	22
LD	Load accumulator	8	LDF	Load fall accumulator	23
CMP	Compare counter	9	ANDR	And rise	24
S	Set	10	ANDF	And fall	25
R	Reset	11	ORR	Or rise	26
JMP	Jump conditional	12	ORF	Or fall	27
LDN	Load not accumulator	13	XORR	Xor rise	28
ANDN	And not	14	XORF	Xor fall	29
ORN	Or not	15	N	Not	30

A compilação do programa é realizada linha a linha, no início da execução do programa. Vejamos como se processa, em termos genéricos, para a maioria das instruções. Consideremos, a título de exemplo, a tabela 3, na qual se representa um conjunto de instruções (código fonte) recebidas pelo SimAP, para serem convertidas

em código executável e a tabela 1 na qual se representa o vector de registo em memória das variáveis do SimAP.

Tabela 3 - Código fonte

Linha do programa	Código	Operando
1	Ld	I1
2	St	Q1
3	Ld	I5
4	Or	I7
5	S	Q5

O compilador começa por identificar o código da instrução e o respectivo operando. Em seguida, converte o código da instrução no endereço da respectiva função lógica a realizar, de acordo com a tabela 2, e o operando no seu endereço de registo em memória, de acordo com a tabela 1., obtendo-se como resultado o código executável (compilado) representado na tabela 4, que será utilizado pelo SimAP para a execução cíclica do programa.

Tabela 4 - Código executável

Linha do programa	Endereço da função lógica	Registo em memória
1	8	1
2	3	9
3	8	5
4	3	7
5	10	13

2.3.2.1 Compilação das instruções de salto para subrotina/linhas do programa

A descrição anterior, aplica-se para a compilação de todas as instruções com excepção das instruções de salto, cujo código resultante constitui uma das excepções relativamente ao código executável gerado para restantes instruções. Este facto deve-se à necessidade de, além de converter o código e o operando da instrução, identificar a linha para onde o programa terá que saltar e a linha actual, no caso de salto para subrotina, para onde o programa deverá retornar assim que voltar da subrotina. A sintaxe da instrução representa-se na tabela 5.

Tabela 5 - Instrução de salto

Código	Operando
JMP/C	XX

Como já foi referido, esta instrução pode desencadear dois tipos de acções distintas, que dependem do tipo de operando utilizado. Se o operando for uma "Label" do tipo "L0...L99" a instrução resulta num salto para a linha do programa identificada pela referida "Label". Se o operando for do tipo "P1...P99" a instrução resulta no salto para uma subrotina identificada por "Inicio P1...Inicio P99". O salto pode ser condicional ou não, consoante o tipo de instrução utilizada, respectivamente JMP ou JMPC.

Ao contrário do que acontece, por exemplo, numa instrução de "LD XX", tabela 6, na qual o código resultante da compilação, tabela 7, é constituído por um endereço para a função lógica e pelo endereço do registo em memória,

Tabela 6 - Instrução "LD": Código fonte

Código	Operando
Ld	I1

Tabela 7 - Instrução "LD": Código executável

Endereço da função lógica	Registo em memória
8	1

numa instrução de salto referente à chamada de uma **subrotina**, tabela 8, o código resultante, tabela 9, é constituído pelo endereço da função "JMP", obtido pela leitura do código da instrução, pelo numero da linha actual da instrução, necessária para a continuidade do programa depois do retorno da subrotina, pelo identificador da subrotina para onde tem que saltar (P1), obtido pela leitura do operando, e por um parâmetro que permite à função "JMP" efectuar a distinção entre a chamada de subrotina e salto para linha de programa. Este ultimo parâmetro, deve-se ao facto de se ter utilizado a mesma função independentemente de se tratar de salto para subrotina ou linha de programa. Esta opção foi adoptada com o intuito de tornar a execução do programa o mais expedita possível, através da restrição do numero de funções utilizadas.

Tabela 8 - Instrução de salto subrotina Código fonte

Código	Operando
JMP	P1

Tabela 9 - Instrução de salto subrotina: Código executável

Endereço da função lógica	Nº da linha actual	Identificador	Pxx/Lxx
12	Linha	1	0

No caso da instrução "JMP" se referir ao salto para uma linha de programa, naturalmente que identificada pela label "L0...L99", tabela 10, o código resultante da compilação, tabela 11, é constituído pelo endereço da função "JMP", pelo identificador da label (L17) e pelo parâmetro que indica à função "JMP" que se trata de salto para linha do programa.

Tabela 10 - Instrução de salto linha programa: Código fonte

Código	Operando
JMP	L17

Tabela 11 - Instrução de salto linha programa: Código executável

Ponteiro para função lógica	Nº da linha	Identificador	Pxx/Lxx
12	-	17	99

Independentemente do tipo de salto, linha de programa ou subrotina, o destino é identificado pelo parâmetro "identificador" resultante da compilação do programa, conforme se representa nas tabelas 9 e 11. Na realidade este parâmetro apenas identifica a subrotina/Label para onde se pretende saltar e não directamente a linha do programa para onde deve ser realizado o salto.

O numero da linha é obtido durante a execução do programa, através da leitura de dois vectores de registos criados quando se efectua a compilação do programa, um que efectua a correspondência entre a as "Labels" (Lxx) utilizadas e as linhas onde se encontram e outro para a correspondência entre o início das subrotinas (Pxx) e as linhas onde se encontram.

Na compilação do programa, sempre que surgir um início de subrotina "Início Pxx", é guardada, num vector de registos, a linha do programa onde se encontra e o identificador da subrotina, P"xx", conforme se representa na tabela 13
A mesma descrição se aplica às "Labels", sempre que surgir uma "label", Lxx, é guardado, num vector de registos, a linha onde se encontra e o seu identificador, L"xx", conforme se representa na tabela 14

Consideremos a titulo de exemplo, a Tabela 12, onde se representa um programa constituído por, programa principal e duas subrotinas. O programa principal contém instruções de salto para as subrotinas e para um linha do programa a jusante.

Perante a label "Início P0" o compilador guarda no vector de registos de subrotinas, o numero da linha da instrução e o identificador da subrotina (P0). Quando surge a instrução "Início P1", o compilador guarda no endereço seguinte do vector de registos o mesmo tipo de informação referente à subrotina P0, procedendo-se da mesma forma para "Início P2", conforme se representa na tabela 13.

Tabela 12 - Exemplo de programa

Linha	Programa	Comentários
0		
1	**Inicio P0**	**;Início do programa principal**
2	JMP P1	;Salto para a subrotina "Início P0"
3	JMP P2	;Salto para a subrotina "Inicio P1"
4	JMP L12	;Salto para a linha do programa L12
5	LD I1	
6	St Q2	
7	**L12:**	
8	S Q1	
9		
10	Fim P0	;Fim do programa Principal
11		
12	**Inicio P1**	**;Início da subrotina P1**
13	Ld I1	
14	St Q1	
15	Fim P1	;Fim da subrotina P1
16		
17	**Inicio P2**	**;Incio da subrotina P2**
18	Ld I2	
19	JMPC L2	
20	St Q2	
21	**L2:**	
22	St Q5	
23	Fim P2	;Fim da subrotina P2

Tabela 13 - Vector de registos em memória para subrotinas

Endereço	Linha da subrotina	Identificador
0	1	0
1	12	1
2	17	2

Quando surgir a label "L12:" o compilador guarda no vector de registo de labels o numero da linha na qual se encontra a label e o respectivo identificador (L12:), quando surgir a label "L2:" o SimAP guardará no endereço seguinte o mesmo tipo de informação, conforme se representa na tabela 14.

Tabela 14 - Vector de registos em memória para labels

Endereço	Linha da subrotina	Identificador
0	7	12
1	21	2

2.3.2.2 Compilação das instruções de temporização

Relativamente à temporização houve apenas necessidade de definir a instrução para inicialização dos temporizadores, cuja sintaxe se representa na tabela 15.

Tabela 15 – Temporização: inicialização

Código	Operando
IN	TMx

Tal como acontece para as instruções de salto, é utilizada a mesma função para todos os temporizadores independentemente do modo de funcionamento escolhido, de forma a restringir-se o numero de funções utilizadas, deste modo a única informação necessária aquando da chamada desta função, durante a execução do programa, é o numero do temporizador. Assim, na compilação desta instrução o que se faz, é converter o código da instrução no endereço da respectiva função e o operando no numero do temporizador que se pretende utilizar.

Consideremos a título de exemplo o programa representado na tabela 16.

Tabela 16 - Temporização: sintaxe

Linha	Programa	Comentários
0		
1	Inicio P0	;Início do programa principal
2	LD I1	
3	**IN TM1**	**;Inicializa o temporizador 1**
4	Fim P0	;Fim do programa principal

Na compilação do programa quando surgir a instrução "IN TM1", o código da instrução é convertido no endereço 4, de acordo com a tabela 2, e o operando "TM1" no identificador do temporizador utilizado, 1, conforme se representa na tabela 17.

Tabela 17 - Temporização: código executável

Endereço da função	Temporizador
4	1

2.3.2.3 Compilação das instruções de contagem

Para a contagem foram definidas as instruções de incremento, INC Cx , decremento, DEC Cx, e reset, R Cx, tendo sido atribuídas a cada uma delas uma função. No entanto, tal como acontece na instrução de temporização, a mesma função é utilizada para os quatro temporizadores, deste modo a única informação necessária aquando da chamada da função, durante a execução do programa, é o numero do contador. Assim na compilação destas instruções o que se faz é converter o código da instrução no endereço da respectiva função e o operando no numero do contador. Consideremos a título de exemplo o programa representado na tabela 18

Tabela 18 - Contagem: sintaxe

Linha	Programa	Comentários
0		
1	Inicio P0	;Início do programa principal
2	LD I1	
3	INC C1	**;Incrementa o contador 1**
4	DEC C3	**;Decrementa o contador 3**
5	R C2	**;Coloca a zero o contador 2**
6	Fim P0	;Fim do programa principal

Na compilação do programa quando surgir a instrução "INC C1", o código da instrução é convertido no endereço da função de incremento do contador, 6, de acordo com a tabela 2, e o operando "C1" no identificador do contador utilizado, 1, quando surgir a instrução "DEC C3" o seu código é convertido no endereço da

função de decremento do contador, 7, e o operando no numero do contador, 3, e finalmente quando surgir a instrução "R C2", o código é convertido no endereço da função de "Reset", 21, e o operando no numero do contador, 2, conforme se representa na tabela 19.

Tabela 19 - Contagem: código executável

Linha	Endereço da função	Temporizador
...	...	...
3	6	1
4	7	3
5	21	2
...	...	...

2.4 Execução do programa

Na apresentação a seguir descrita são abordados os seguintes itens:

2.4.1 Execução cíclica do programa

2.4.2 Execução da instruções de salto

2.4.3 Execução das instruções de temporização/contagem

2.4.4 Fluxogramas

2.4.5 Tempo de ciclo/Diagramas temporais

2.5 Traçado de diagramas temporais

2.4.1 Execução cíclica do programa

O SimAP executa o programa de forma cíclica, efectuando a leitura linha a linha do código executável. Independentemente do tipo da instrução compilada, o programa testa, através de um mecanismo *select case,* o primeiro elemento do código executável, para determinar qual a função que deverá chamar, e para a maioria das instruções, passa em seguida como parâmetro, o segundo elemento de dados representativo do endereço do registo do operando.

Consideremos o exemplo representado na tabela 20, constituída pelo código fonte e o respectivo código executável:

Tabela 20 - Leitura de linhas do programa

Código	Operando
Ld	I1
St	Q1

Endereço da função lógica	Registo em memória
5	1
3	9

Neste caso o SimAP efectuaria a leitura do endereço da função (5), em seguida através do mecanismo *Select Case,* representada no excerto do programa que se apresenta a seguir:

```
Select Case n5        ' n5 contém o endereço da função lógica
  Case 0              'linhas em branco
  Case 1
      X = And_(Lprog(l, 1)) 'se n5=1 executa a função "AND" e passa-lhe como
                            'parâmetro o endereço do registo do operando guardado
                            ' em Lprog (l,1)
  Case 2
      X = Or_(Lprog(l, 1))   'se n5=2 executa a função "OR" e passa-lhe como parâmetro
                           'o endereço do registo do operando
  Case 3
      X = store_(Lprog(l, 1))    'se n5=3 executa a função "STORE" e passa-lhe como
parâmetro
                             'o endereço do registo do operando
  ...

  Case 21
      X = Reset_Counter(Lprog(l, 1))   'se n5=21 executa a função "Reset Counter" e
passa-lhe
                                 'como parâmetro o endereço do operando
  Case 22
      X = load_R(Lprog(l, 1))     'se n5=22 executa a função "Load Rise e passa-lhe como
                                  'parâmetro o endereço do registo do 'operando
  Case 23
       X = Load_F(Lprog(l, 1))     'se n5=23 executa a função "Load Fall e passa-lhe como
                                   'parâmetro o endereço do registo do 'operando
  Case 24
      X = And_R(Lprog(l, 1))      'se n5=24 executa a função "And Rise e passa-lhe como
                                  'parâmetro o endereço do registo do 'operando

Case 101
     X = fim_prog()
DoEvents
End Select
```

seria executada a função ,"Load acumulator" , conforme se representa na tabela 21:

Tabela 21 - Função "Load"

```
Public Function load_acumulador (n3 As Integer)
    Acumulador = V(n3)      ' efectua a leitura para o acumulador do conteúdo do
                            ' conteúdo do vector de registos V(n3).
DoEvents
End Function
```

passando-lhe como parâmetro o endereço do registo em memória da entrada 1 (n3=1). Em termos resultado prático, seria efectuada a leitura para o acumulador do conteúdo do registo em memória referente à entrada 1.

O programa continuaria em execução efectuando a leitura da segunda linha. Seria então efectuada a chamada à função "Store accumulator" , sendo-lhe passada como parâmetro o endereço do registo em memória da saída 1. O resultado prático consistiria na memorização do nível lógico do acumulador no referido endereço. A actualização desta saída seria realizada apenas no fim do ciclo do programa.

2.4.2 Execução das instruções de salto

Uma das particularidades da execução de programas, diz respeito à execução das instruções de salto para subrotinas ou linhas do programa. Já foi descrito que a informação relevante neste caso, não é o armazenamento de dados em memória, como anteriormente, mas sim a identificação da linha a partir da qual o programa passará a ser executado e, no caso de salto para subrotina, a linha actual antes do salto de forma a poder-se dar continuidade ao programa depois de executada a subrotina.

Considere o exemplo representado na tabela 22:

Tabela 22 - Programa: Exemplo

Linha	Programa	Comentários
0		
1	**Inicio P0**	;Início do programa principal
2	JMP P1	;Salto para a subrotina "Início P0"
3	JMP P2	;Salto para a subrotina "Inicio P1"
4	JMP L12	;Salto para a linha do programa L12
5	LD I1	
6	St Q2	
7	**L12:**	
8	S Q1	
9		
10	Fim P0	;Fim do programa Principal
11		
12	**Inicio P1**	**;Início da subrotina P1**
13	Ld I1	
14	St Q1	
15	Fim P1	;Fim da subrotina P1
16		
17	**Inicio P2**	**;Incio da subrotina P2**
18	Ld I2	
19	JMPC L2	
20	St Q2	
21	**L2:**	
22	St Q5	
23	Fim P2	;Fim da subrotina P2

O código executável que iríamos obter seria o representado na tabela 23:

Tabela 23 Código executável resultante

Linha do Programa exe	Endereço da função lógica	Nº da linha actual	Identificador	Pxx/Lxx
...	...	...	...	...
2	12	2	1	0
3	12	3	2	0
4	12	-	12	99
...	...	...	...	...

e os registos auxiliares das subrotinas e "Labels" os representados respectivamente na tabela 24 e 25, conforme já apresentado.

Tabela 24 Vector de registos em memória para subrotinas

Endereço	Linha da subrotina	Identificador
0	1	0
1	12	1
2	17	2

Tabela 25 Vector de registos em memória para labels

Endereço	Linha da subrotina	Identificador
0	7	12
1	21	2

Quando surgir a instrução de salto para subrotina, "JMP Pxx", ou para linha de programa, "JMP Lxx", representadas pelo código resultante da compilação, o simulador efectua a busca do identificador da subrotina ou da label, nos respectivos registos auxiliares armazenados em memória, e efectuará a leitura da linha para onde o programa terá que saltar.

A distinção entre salto para subrotina e salto para linha de programa, é efectuada através da consulta do parâmetro da ultima coluna, respectivamente 0 e 99. No caso de se tratar de salto para subrotina, o simulador guarda, ainda, numa pilha ("stack") do tipo "FILO - first in last out" a informação referente ao "nº da linha", que irá permitir identificar a linha de retorno quando a subrotina chega ao fim.
Uma vez que não foi imposta limitação ao tamanho da pilha, o numero de saltos consecutivos para subrotinas, fica apenas limitado apenas pelo identificador utilizado ("P99").

No exemplo que se apresenta, assim que for efectuada a leitura da linha 2 do código executável, o simulador efectua a chamada da função "JMP", representada no programa pelo seu endereço, 12. Através do parâmetro da ultima coluna, 0, verifica que se trata de uma subrotina e, em seguida, efectua a leitura do identificador da subrotina procurando no vector de registos em memória da tabela 24 a linha para

onde terá que saltar. Como se trata de um salto para subrotina, será guardada na pilha o numero da linha actual, 12. Entretanto seria executada a subrotina pretendida e quando o simulador encontrasse a instrução "Fim Px", referente à rotina em execução, efectuaria a chamada da respectiva função, representada na tabela 26, para obter o numero da próxima linha a ser executada.

Tabela 26 - Função "Fim_programa"

```
Public Function fim_prog()
   st = st – 1                    'stack=stack-1
   inicio = stack(st)             'leitura do conteúdo da stack (linha do programa)
   st = st – 1                    'stack=stack-1
End Function
```

O processo repetir-se-ia para a linha 3. A leitura da linha 4, apesar de representar o salto para uma linha do programa, a única diferença seria a de não haver a necessidade de guardar a linha de retorno na pilha.

2.4.3 Execução das instruções de temporização/contagem

Na execução das instruções de temporização e contagem todas as instruções são activas ao flanco, por exemplo, no código representado na tabela 27:

Tabela 27 - Temporização: Código do programa

Instrução	Operando
LD	I4
IN	TM1
LD	I5
IN	TM2
LD	I1
INC	C1
LD	I2
DEC	C2
LD	I3
R	C3

O temporizador 1 e 2 seriam inicializados, respectivamente, no flanco ascendente da entrada I4 e I5. O contador 1 seria incrementado no flanco ascendente da entrada I1, o contador 2 decrementado no flanco ascendente da entrada I2 e o contador 3 colocado zero no flanco ascendente da entrada I3.
Além destas instruções activas ao flanco, verificamos também, aquando da análise da linguagem de programação, a existência das instruções xxR e xxF activas, respectivamente ao flanco ascendente e descendente.

2.4.4 Detecção de flancos na execução de programas

A detecção de flancos ascendentes ou descendentes é realizada com o auxílio de um conjunto de registos auxiliares em memória. Para cada tipo de operando existe um tipo de registo auxiliar, conforme se representa na tabela 1, para as entradas existem os registos "entrada auxiliar", para as saídas existem os registos "saida auxiliar", para as flags dos temporizadores existem os registos "flg. Tmr. Aux." e para os contadores os registos "flg. cont. aux."

O registo auxiliar guarda o valor da expressão lógica, referente ao ciclo anterior do programa. Sempre que uma instrução, activa por flanco, é executada, o nível lógico do operando é comparado com o nível lógico do respectivo registo auxiliar. Se o nível lógico do operando for "1" e diferente do nível lógico do registo auxiliar, então existiu um flanco ascendente. Se, por outro lado, o nível lógico do operando for "0" e diferente do nível lógico do registo auxiliar, então existiu um flanco descendente

Independentemente da existência ou não de flanco ascendente, o conteúdo do registo auxiliar é sempre actualizado em cada ciclo do programa, com o nível lógico do operando.

2.4.5 Fluxograma

Os fluxogramas representados na figura 21, descrevem a execução do programa no SimAP.

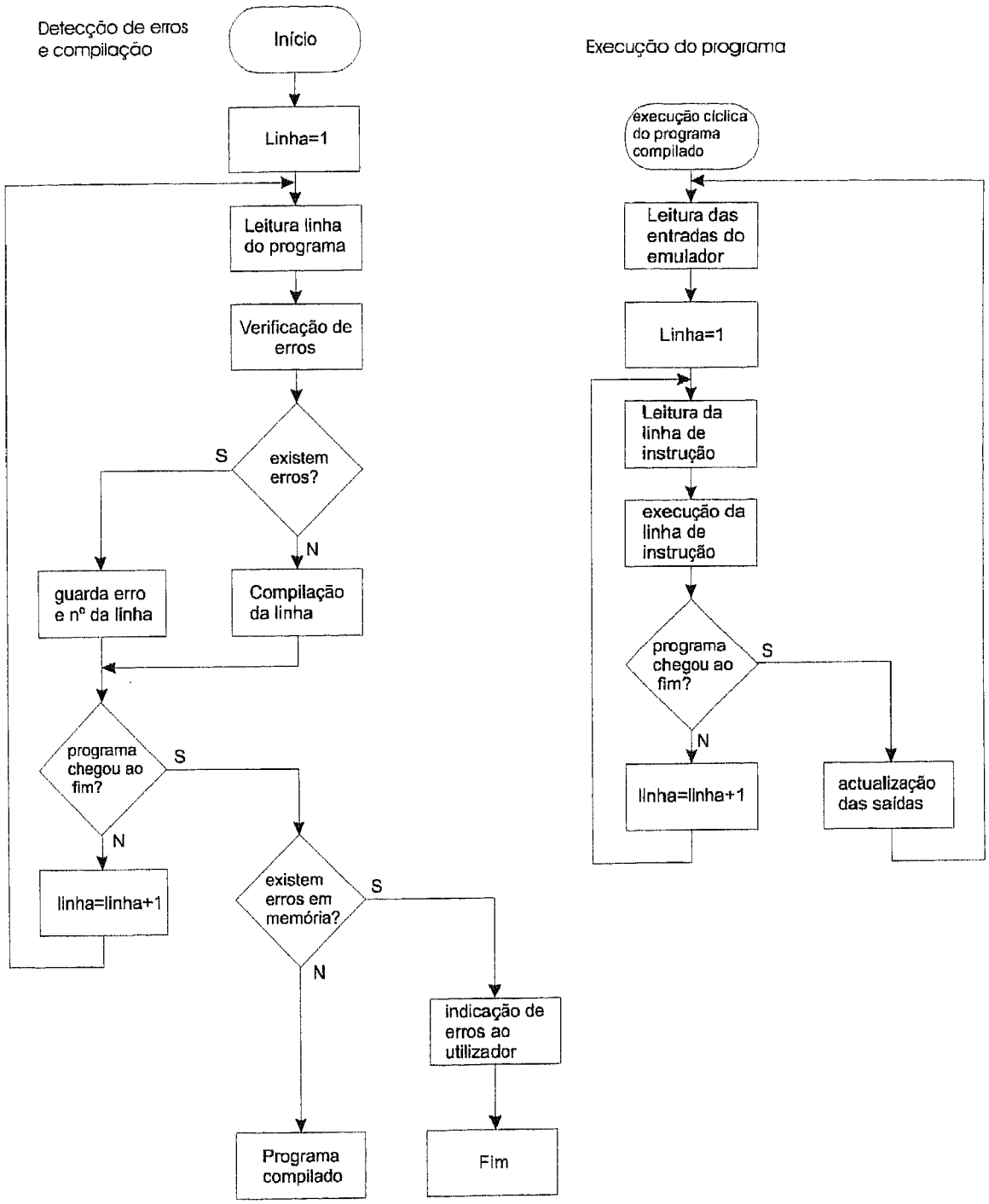


Figura 21 – Fluxogramas

2.4.6 Tempo de ciclo/Diagramas temporais

O tempo que o simulador utiliza para executar um ciclo do programa do autómato programável designa-se por tempo de ciclo.
Atendendo ao facto da aplicação desenvolvida se tratar de um sistema de desenvolvimento, que não vai interagir directamente no controlo de um sistema real, não se colocam problemas críticos em termos de velocidade de processamento. Interessa sim, poder-se acompanhar a evolução do programa e verificar a resposta a eventos.

Houve então a considerar as formas possíveis de acompanhamento do programa, de modo a concluir-se qual o tempo de ciclo mais apropriado.
As hipóteses possíveis são, efectuar o acompanhamento do programa através do menu principal do simulador, ou através da visualização dos diagramas temporais on-line ou off-line. No primeiro caso, o interesse é mais o de visualizar a relação evento/acção, do que propriamente efectuar um estudo em termos temporais, por isso, não houve a preocupação em definir o tempo de ciclo do programa, ficando este dependente do tempo de execução por parte do PC utilizado. O mesmo não acontece no segundo caso.

Quando a execução do programa é acompanhada através da visualização do traçado do diagrama temporal "on-line", torna-se imperativo a utilização de um tempo de ciclo fixo.

A escolha deste valor não tomou em consideração nenhum factor rígido, tendo sido escolhido na base do bom senso, ou seja, tinha que ser utilizado um valor fixo de modo a poder-se efectuar uma correspondência directa entre a evolução dos sinais visualizados e a escala utilizada no diagrama temporal, e que permitisse ao utilizador acompanhar a evolução do próprio diagrama. Tomando-se em consideração que quanto mais rápido for o ciclo de execução, mais rápido será o varrimento do diagrama temporal, tornando difícil o seu acompanhamento, adoptou-se um tempo de ciclo de 0,5s.

Apesar de no traçado do diagrama temporal "off line", não se colocar o problema do acompanhamento directo da evolução do programa e por este motivo não se impor nenhum tempo de ciclo, o registo de todas as variáveis é realizado com intervalos de tempo de 0,5s, de modo a manter a coerência em termos de escala com o diagrama temporal on-line.

2.5 Traçado de diagramas temporais

A evolução de um programa pode ser acompanhada através da visualização do diagrama temporal "on-line". Este diagrama possibilita a visualização simultânea de até 5 formas de onda , conforme se representa na figura 22, e permite verificar de que modo estão relacionadas as variáveis do sistema, em especial aquelas que são activas por flanco.

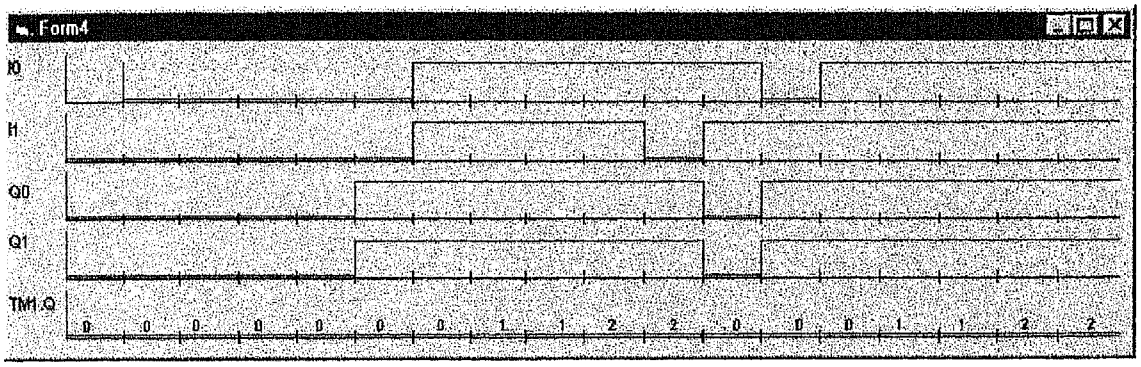


Figura 22 - Diagrama temporal on-line: Monitorização

O utilizador poderá também optar por visualizar os diagramas temporais "off-line", conforme se representa na figura 23, ou seja, depois de executado o programa poder-se-á visualizar a sua evolução de forma pormenorizada e repetitiva. Deste modo, o utilizador poderá analisar de forma mais cuidada e atenta se o programa cumpre os requisitos necessários, por exemplo, para o controlo automatizado de um sistema a implementar. Este diagrama pretende fornece uma visão mais abrangente da globalidade das relações evento/acção entre variáveis, permitindo por este motivo a visualização simultânea do diagrama temporal de até 13 formas variáveis.

Para se efectuar o traçado dos diagramas temporais, o valor lógico que as variáveis do sistema assumem em cada ciclo de execução do simulador, são guardadas em dois vectores de dados distintos, um utilizado para o traçado do diagrama temporal "on-line" e o outro para o traçado do diagrama temporal "off-line".

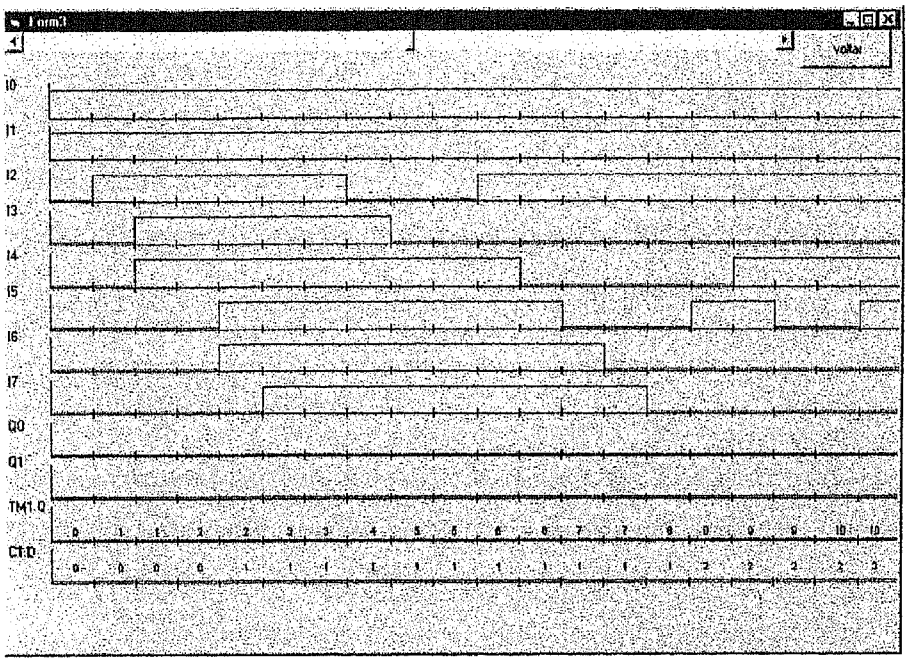


Figura 23 - Diagrama temporal off-line: Monitorização

No traçado do diagrama temporal "on-line", são utilizados vectores de 18 elementos que mantêm actualizados, em cada ciclo de execução, os últimos 18 valores lógicos de todas as variáveis do sistema. O mecanismo para actualização destes vectores processa-se de forma semelhante à de um registo de deslocamento, ou seja, o novo nível lógico da variável, resultante do ultimo ciclo de execução, ao ser armazenado provoca que o valor mais antigo seja eliminado, conforme se representa na figura 24

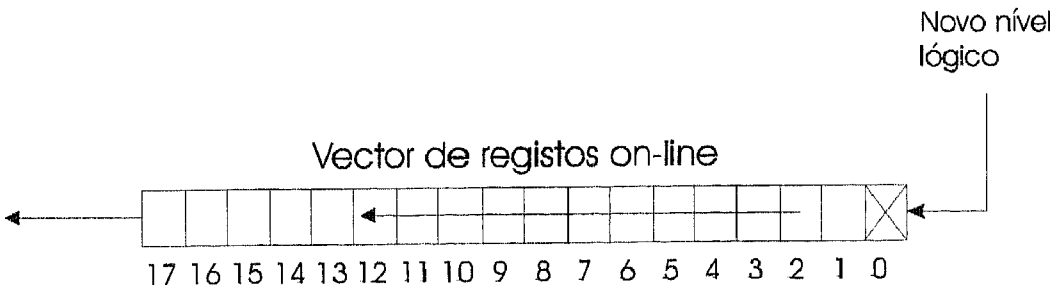


Figura 24 -Vector de registo on-line

O simulador utiliza este vector para, em cada ciclo de execução, efectuar a actualização de todas as variáveis do diagrama temporal, realizando uma correspondência directa entre os elementos armazenados e a escala utilizada, conforme se representa na figura 25.

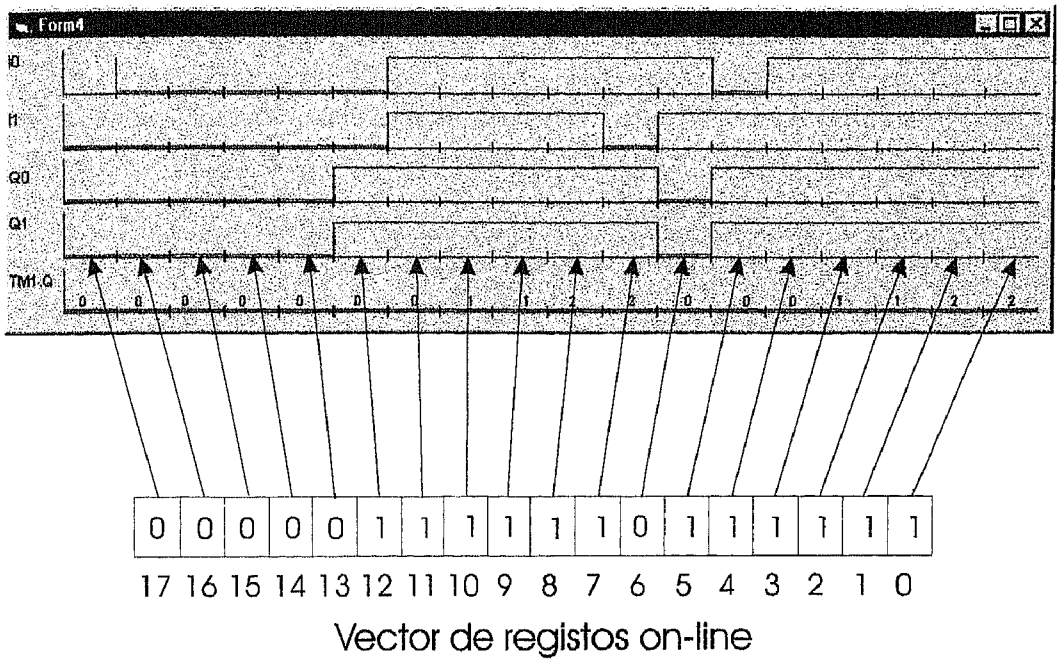


Figura 25 - Actualização de variáveis

2.6 Tempo de ciclo

O tempo que o simulador utiliza para executar um ciclo do programa do autómato programável designa-se por tempo de ciclo.
Atendendo ao facto da aplicação desenvolvida se tratar de um sistema de desenvolvimento, que não vai interagir directamente no controlo de um sistema real, não se colocam problemas críticos em termos de velocidade de processamento. Interessa sim, poder-se acompanhar a evolução do programa e verificar a resposta a eventos.

Houve então a considerar as formas possíveis de acompanhamento do programa, de modo a concluir-se qual o tempo de ciclo mais apropriado.

As hipóteses possíveis são, efectuar o acompanhamento do programa através do menu principal do simulador, ou através da visualização dos diagramas temporais on-line ou off-line. No primeiro caso, o interesse é mais o de visualizar a relação evento/acção, do que propriamente efectuar um estudo em termos temporais, por isso, não houve a preocupação em definir o tempo de ciclo do programa, ficando este dependente do tempo de execução por parte do PC utilizado.

Quando a execução do programa é acompanhada através da visualização do traçado do diagrama temporal "on-line", torna-se imperativo a utilização de um tempo de ciclo fixo.

A escolha deste valor não tomou em consideração nenhum factor rígido, tendo sido escolhido na base do bom senso, ou seja, tinha que ser utilizado um valor fixo de modo a poder-se efectuar uma correspondência directa entre a evolução dos sinais visualizados e a escala utilizada no diagrama temporal, e que permitisse ao utilizador acompanhar a evolução do próprio diagrama. Tomando-se em consideração que quanto mais rápido for o ciclo de execução, mais rápido será o varrimento do diagrama temporal, tornando difícil o seu acompanhamento,

adoptou-se um tempo de ciclo de 0,5s, conforme se representa no fluxograma da figura 26.

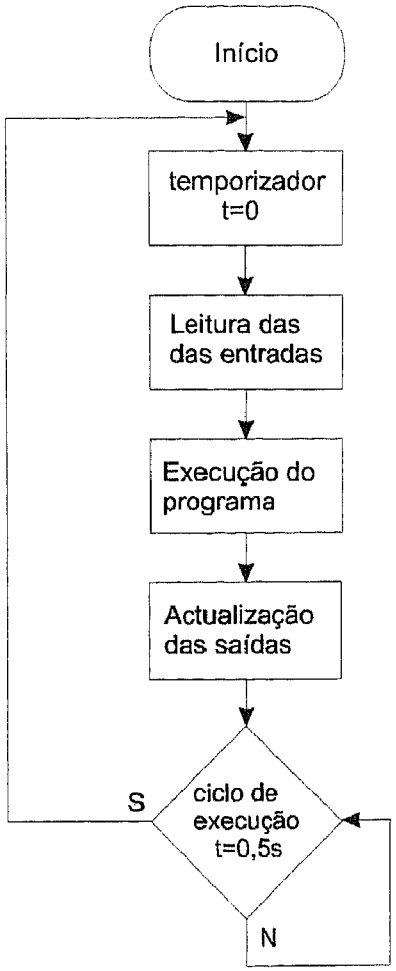


Figura 26 - Fluxograma do ciclo de execução

Apesar de no traçado do diagrama temporal "off line", não se colocar o problema do acompanhamento directo da evolução do programa, o registo de todas as variáveis é realizado em intervalos de tempo de 0,5s, de modo a manter a coerência em termos de escala com o diagrama temporal on-line.

2.7 Ligação aos simuladores de processos físicos

Uma das funcionalidades fundamentais disponíveis na aplicação desenvolvida é a possibilidade do simulador interagir de forma activa com simuladores de processos físicos.

Uma vez que os módulos de software, referentes ao SimAP e simuladores, constituem aplicações distintas foi necessário adoptar uma forma de comunicação expedita, para se realizar uma transferência de dados eficiente entre aplicações.

Da diversidade de soluções disponíveis optou-se por criar, em cada uma das aplicações, um conjunto de variáveis auxiliares de entrada e saída, e efectuar entre elas uma ligação do tipo DDE ("Dinamic data enchange") para realizar a transferência de dados. A vantagem desta opção consiste na utilização de um conjunto de variáveis dedicadas para troca de dados entre aplicações.

Tomando em consideração esta realidade, sempre que for realizada a ligação do SimAP a um simulador, acontecerá que, quando no início do ciclo se efectuar a leitura das entradas do SimAP, na realidade, efectua-se a leitura das suas variáveis auxiliares, da mesma forma que, quando, no fim do ciclo, se efectuar a actualizações das saídas, na realidade está-se a actualizar as variáveis auxiliares. Naturalmente, que as actualizações das saídas auxiliares, só por si, não chega é necessário que existam repercussões na aplicação à qual o SimAP está ligado. Para que isto se torne possível, a ligação do SimAP ao simulador começa por efectuar a ligação DDE entre variáveis auxiliares, conforme se representa na figura 27.

Esta ligação é realizada sempre de forma semelhante e é independente do simulador utilizado, no entanto há que tecer algumas considerações.

A ligação entre as aplicações, consiste na ligação das entradas auxiliares do SimAP às saídas auxiliares do simulador, de modo a que eventuais detectores que o simulador ligue às suas saídas auxiliares possam ser lidas pelo SimAP, e a ligação das saídas auxiliares do SimAP às entradas auxiliares do simulador, de modo a que o SimAP possa controlar eventuais accionadores do simulador que este tenha ligado às suas entradas auxiliares.

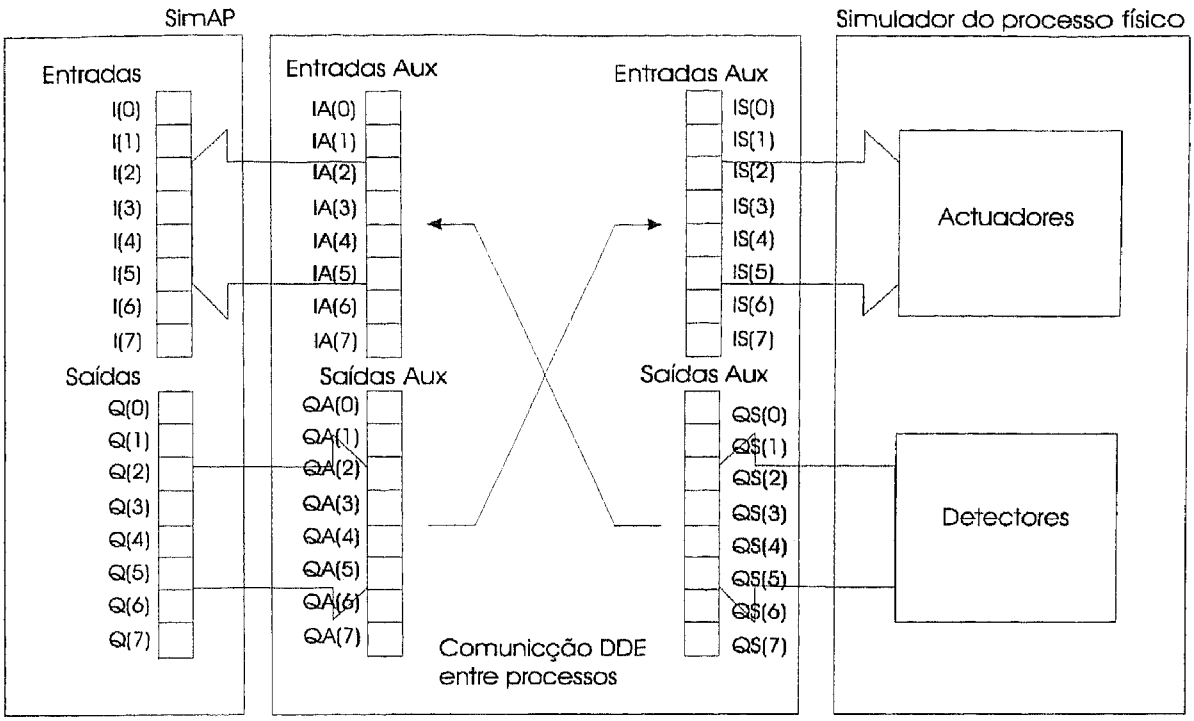


Figura 27 - Ligação do SimAP ao simulador do processo físico: Diagrama de blocos

Numa ligação DDE há sempre uma aplicação responsável por estabelecer a ligação entre variáveis. No módulo de software desenvolvido a aplicação em causa é sempre a do SimAP, que além da referida ligação é também responsável pelo arranque da aplicação do simulador.

Sempre que se efectuar a ligação a um simulador, o SimAP realiza uma ligação DDE entre todas as suas entradas auxiliares e as saídas auxiliares do SimAP. O simulador, por sua vez, encarrega-se de efectuar a ligação DDE das entradas auxiliares que vai utilizar às respectivas saídas auxiliares do AP. De se notar que, enquanto o SimAP efectua a ligação de todas as suas entradas a todas as saídas do simulador, este ultimo efectua apenas a ligação das entradas de que necessita. Isto acontece pelo seguinte motivo, enquanto no simulador é fácil de definir, durante a sua fase de desenvolvimento, o número de entradas que irá necessitar, as quais se mantêm inalteráveis em qualquer altura da sua execução, no SimAP, devido à diversidade de simuladores a que pode estar ligado, o número de entradas necessárias além de variável é também a partida desconhecido pelo SimAP.

A hipótese para contrariar a necessidade de efectuar a ligação de todas as variáveis, seria a de, no SimAP, cada vez que fosse criado um novo simulador, registar o número de entradas que este necessita, de modo a que, quando o simulador fosse chamado, houvesse informação quanto ao numero de ligações a efectuar. No entanto, apesar de isto ser possível durante uma fase de desenvolvimento, uma vez que obriga à alteração do código do programa, torna-se impossível depois de uma versão ultimada do SimAP devido à inacessibilidade do código do produzido.

O inconveniente da solução adoptada prende-se apenas com a maior demora da ligação entre aplicações e a sustentação de um conjunto de ligações que podem não ser utilizadas, mas que em termos práticos não influencia em nada o desenrolar da aplicação .

Consideremos a título de exemplo, a ligação do SimAP ao simulador representativo do funcionamento de um parque de estacionamento, representado na figura 28.

Tal como foi referido, é da responsabilidade do SimAP iniciar a aplicação do simulador e efectuar a ligação das suas entradas auxiliares às saídas auxiliares do simulador. O simulador, por sua vez, efectua a ligação das suas entradas auxiliares às saídas auxiliares do simulador.

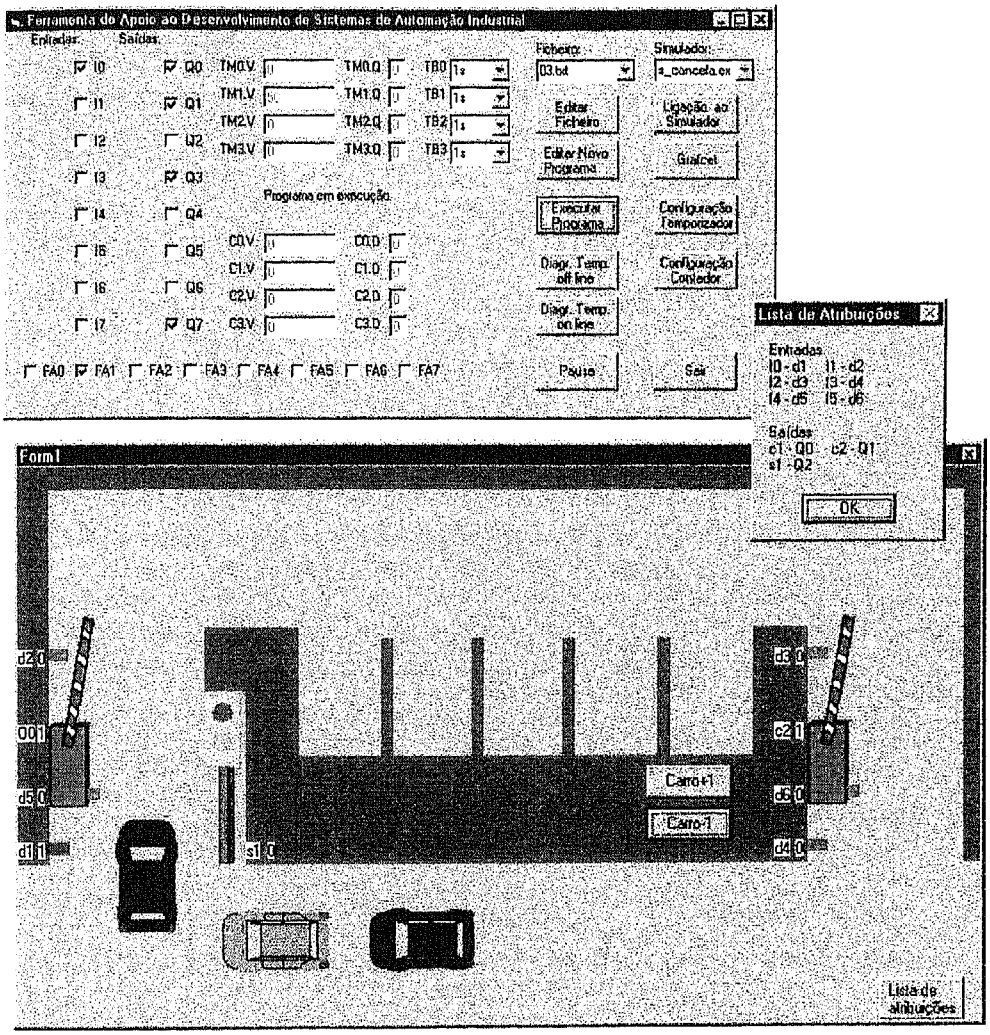


Figura 28 - ligação do SimAP ao simulador

Conforme se pode verificar através da lista de atribuições representada na figura, fornecida pelo simulador ao utilizador a título informativo, o numero de entradas do SimAP, utilizadas pelo simulador são apenas 6, apesar de serem mantidas as 8 ligações. As ligações efectuadas são:

- I(0): Entrada "0" do AP, que efectua a leitura do detector de presença d1, do simulador
- I(1): Entrada "1"do AP, que efectua a leitura do detector de presença d2, do simulador

- I(2): Entrada "2" do AP, que efectua a leitura do detector de presença d3, do simulador
- I(3): Entrada "3" do AP, que efectua a leitura do detector de presença d4, do simulador
- I(4): Entrada "4" do AP, que efectua a leitura do detector de presença d5, do simulador
- I(5): Entrada um do AP, que efectua a leitura do detector de presença d6, do simulador

O numero de saídas do SimAP utilizadas pelo simulador são 3:

- Q(0): Saída "0" do AP, ligada ao actuador c1 do simulador, para controlar a abertura da cancela 1.
- Q(1): Saída "1" do AP, ligada ao actuador c2 do simulador, para controlar a abertura da cancela 2.
- Q(2): Saída "2" do AP, ligada ao actuador S1 do simulador, para controlar o semáforo de entrada.

Desta forma, qualquer alteração do nível lógico destas linhas, terá repercussões idênticas na linha à qual está ligado, de modo a poder-se efectuar o controlo do acesso ao parque de estacionamento de modo adequado. A partir daqui, seria necessário editar o programa, considerando as ligações efectuadas, que realizasse a leitura dos detectores e tomasse as decisões de controlo adequadas para a actuação das saídas e, como consequência, dos actuadores do simulador.

Capítulo 3

GRAFCET

3.1 Apresentação

O GRAFCET (Grafo Funcional de Comando Etapa Transição), também designado por diagrama funcional sequencial, permite descrever o comportamento de um sistema automatizado sequencial através de uma representação gráfica independente da realização tecnológica.

Numa fase de concepção, o grafcet torna-se especialmente útil por possibilitar descrever o funcionamento dos sistemas automatizados, através de modelos que visam estruturar a concepção desses sistemas e a partir das quais pode ser gerado automaticamente o programa para o controlador utilizado.

Na apresentação que se segue são descritos os seguintes itens:

3.1 Apresentação

3.2 Descrição funcional

3.3 Descrição da implementação

3.1.1 Estrutura/Interpretação do GRAFCET

Na descrição que se segue, não se pretende efectuar a apresentação de forma exaustiva do grafcet, mas sim abordar os conceitos de maior relevância para o editor implementado.

A estrutura do grafcet consiste num gráfico cíclico composto, alternativamente de transições e etapas ligadas entre si por arcos orientados.

Etapa

Uma etapa representa o estado de um sistema ou parte dele e caracteriza um comportamento estável do sistema considerado.

A etapa é simbolizada por um quadrado ou rectângulo, com um identificador numérico no interior, conforme se representa na figura 29.

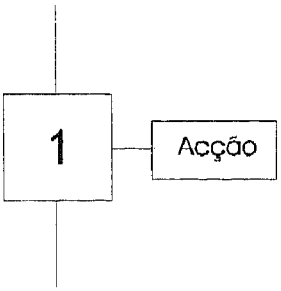


Figura 29 - Etapa

e pode assumir dois estados distintos: activo ou inactivo. A cada etapa está associada uma variável booleana que toma o valor lógico "1" quando a etapa está activa e o valor lógico zero se se encontra inactiva. O conjunto de etapas activas de um grafcet, num determinado instante de tempo, define o estado do sistema automatizado nesse instante

Uma etapa pode ser inicial, se se encontrar activa no arranque do sistema. Este tipo de etapa distinguem-se das restantes por serem desenhadas a traço duplo, conforme se representa na figura 30.

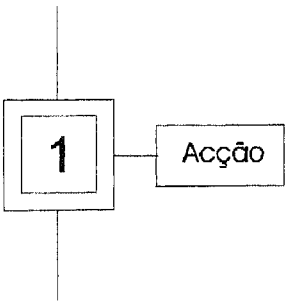


Figura 30 - Etapa inicial

O conjunto destas etapas caracteriza o estado inicial do sistema automatizado. Podemos atribuir uma dada interpretação a uma etapa através de uma acção que lhe está associada. Essa acção será realizada sempre que a respectiva etapa estiver activa.

Transição

Uma transição permite descrever a evolução possível, do estado activo de uma etapa para uma outra etapa. È através da transição que é possível acompanhar a evolução do sistema.

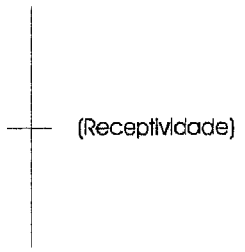


Figura 31 - Transição

Pode-se atribuir uma determinada interpretação a uma transição, através de uma receptividade. A evolução do sistema só será possível, se a informação proveniente da parte operativa tornar verdadeira a receptividade que se pretende transpor.

Transmissão

Por vezes poderá acontecer que na modelação do sistema, surjam demasiados cruzamentos entre linhas, tornando a interpretação do grafo bastante complexa. Para evitar que isto aconteça, são utilizadas transmissões, conforme representado na figura 32, que permitem, "transmitir" a continuidade do programa de um ponto para outro do grafo:

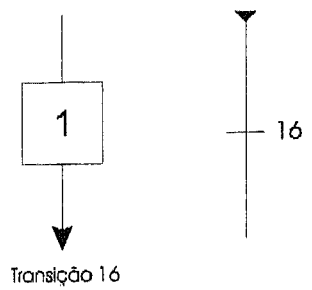


Figura 32 - Transmissão

Ligações entre etapas

A ligação entre etapas pode assumir pode ser realizada de diferentes formas:

- Sequenciais

Neste tipo de ligações, para que uma transição possa ser transposta é necessário que a sua etapa de entrada esteja activa, de forma a validar a transição, e que a respectiva condição de receptividade seja verdadeira, caso contrário mantêm-se na etapa actual. Em termos de edição gráfica, as ligações entre etapas devem ser realizadas na vertical de cima para baixo. Este tipo de ligação é representada na figura 33.

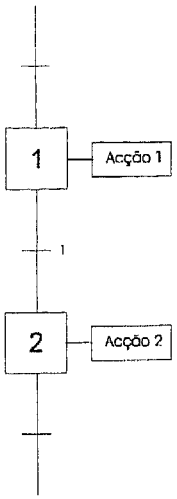


Figura 33 - Ligação sequencial

No exemplo representado na figura anterior, a etapa 2 só será activa se a etapa 1 estiver activa e se a condição de receptividade da transição 1 for verdadeira. Quando isto acontecer a etapa 1 é desactivada e a etapa 2 activada.

- "AND" Convergente

Este tipo de ligação caracteriza-se por um traço horizontal duplo, onde convergem duas ou mais etapas e de onda sai apenas uma, conforme se representa na figura 34

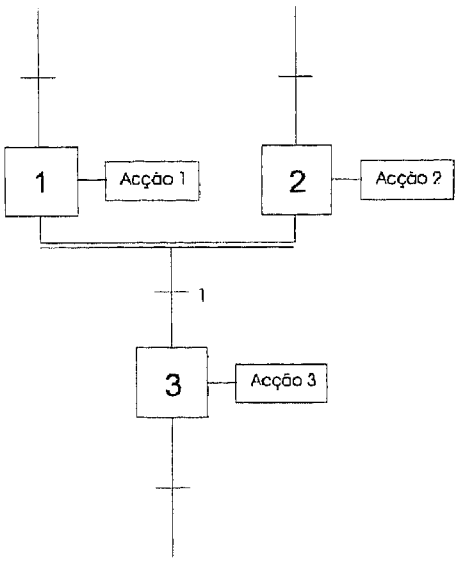


Figura 34 - And convergente

Neste caso para que a transição 1 seja transposta é necessário que as etapas que a antecedem, etapa 1 e 2, estejam ambas activas (uma "**e**" outra) e que a condição de receptividade da transição seja verdadeira. Quando isto acontecer a etapas 1 e 2 ficam inactivas e a etapa 3 é activada, realizando a acção que lhe está associada.

- "AND" Divergente

Este tipo de ligação caracteriza-se por um traço horizontal duplo, onde chega uma etapa e a partir do qual divergem duas ou mais, conforme se representa na figura 35.

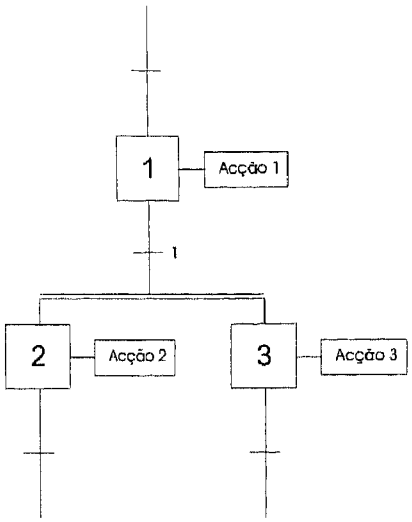


Figura 35 - And divergente

Neste caso para que a transição 1 seja transposta é necessário que a etapa que a antecede, neste caso a etapa 1, esteja activa e que a condição de receptividade da transição seja verdadeira. Quando isto acontecer, a transição é transposta e a etapa 1 fica inactiva, enquanto que as etapas 2 e 3 são activadas simultaneamente.

- "OR" convergente

Este tipo de ligação caracteriza-se por um traço horizontal, onde convergem duas ou mais etapas e de onda sai apenas uma, conforme se representa na figura 36.

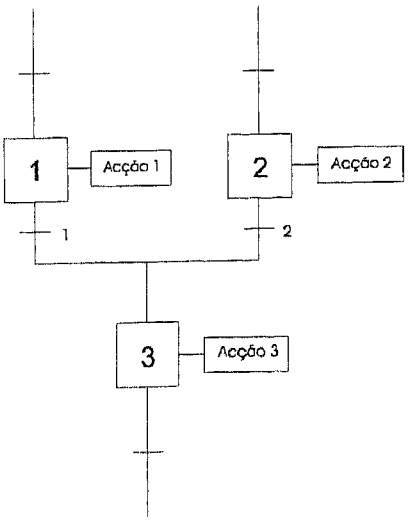


Figura 36 - OR convergente

Neste caso, para que a transição seja transposta é suficiente que uma das etapas antecedentes, etapa 1 ou 2, esteja activa e que a condição de receptividade da respectiva transição , 3 **ou** 4, seja verdadeira. Quando isto acontecer, a etapa antecedente é desactivada e a etapa 3 activada.

- "OR" Divergente

Este tipo de ligação caracteriza-se por um traço horizontal, onde chega uma etapa e a partir do qual divergem duas ou mais, conforme se representa na figura 37.

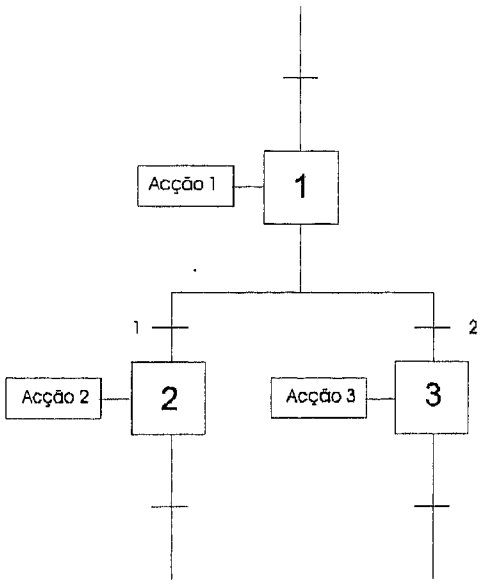


Figura 37 - Or divergente

Neste caso para que a transição seja validada é necessário que a etapa 1 esteja activa. A sua transposição pode ser realizada tornando verdadeira a condição de receptividade da transição 1 **ou** da transição 2. Quando isto acontecer, a etapa 1 é desactivada e a etapa 2 ou 3 é activada, consoante o tipo de receptividade que se torne verdadeira.

Macro-etapas

Uma macro etapa consiste num conjunto de etapas e transições ligadas entre si, nas quais se pretende modular parte do sistema automatizado.

Figura 38 - Macro etapa: Símbolo

Uma macro etapa assemelha-se a uma subrotina. Agrupa um conjunto de etapas e transições, conforme se representa na figura 39, com o objectivo de facilitar a descrição de sistemas mais complexos, através de uma representação gráfica mais organizada.

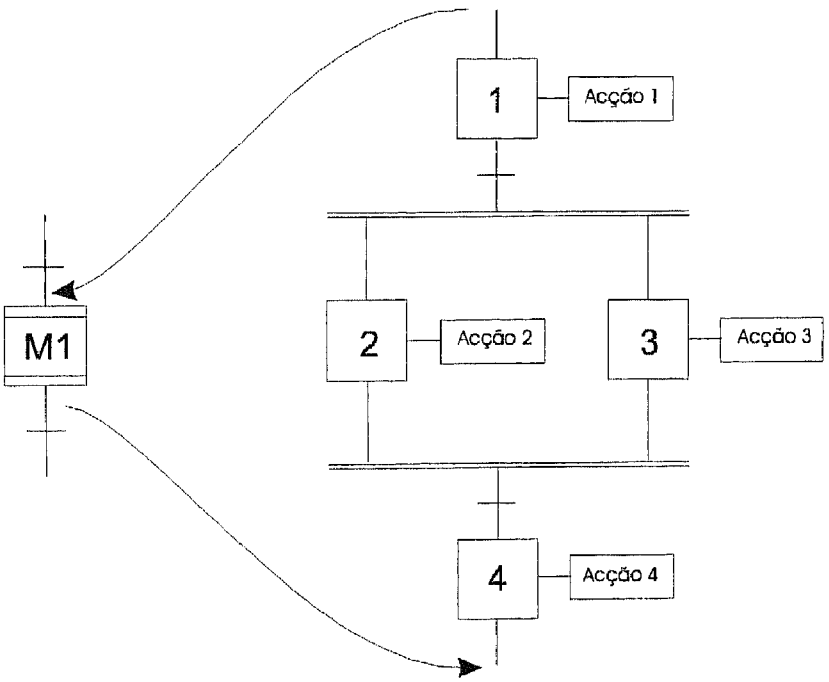


Figura 39 - Macro etapa: Exemplo

Acções

Já verificamos que uma acção pode estar associada a uma etapa e que representa uma saída do sistema lógico que modelizamos. A acção pode ser de três tipos:

- Contínua
- Condicional, podendo pertencer a um dos seguintes subtipos:

- Acções condicionais simples
- Acções condicionais com atraso
- Acções condicionais limitadas no tempo

- Memorizada

Acções contínuas

A saída do sistema correspondente a esta acção mantém-se actuada, enquanto a etapa associada se mantiver activa. Caso contrário, a saída é mantém-se inactiva. O diagrama temporal correspondente, representa-se na figura 40.

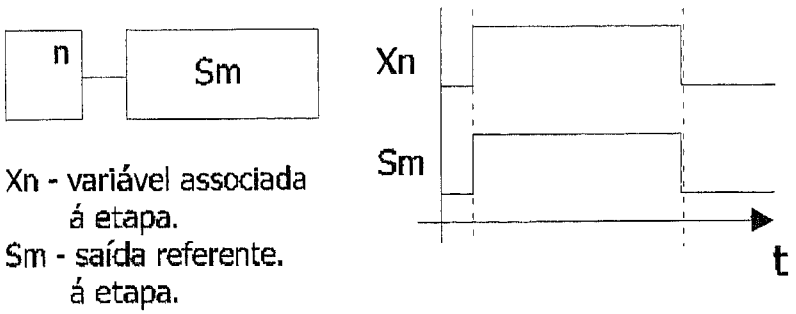


Figura 40 - Acção contínua

Acções condicionais

- simples

Para que a acção seja executada, é necessário que para além da etapa correspondente ter que estar activa, a condição que lhe está associada seja verdadeira, conforme se representa na figura 41.

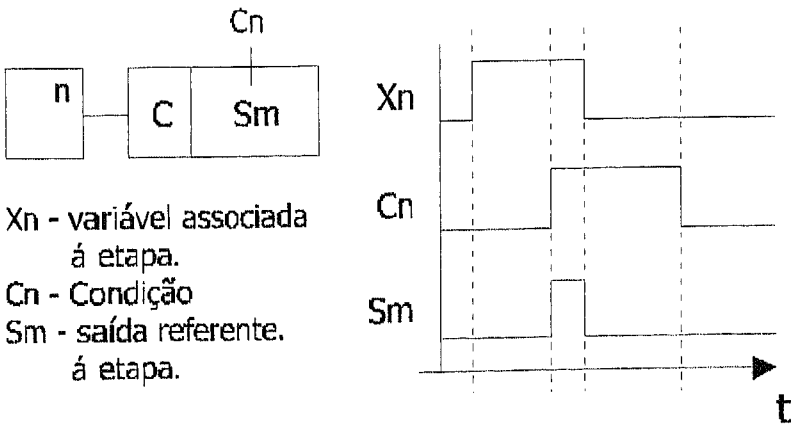


Figura 41 - Acção condicional simples

- Com atraso

Trata-se de um tipo de acção, cuja condição associada se reporta a um atraso, ou seja depois da respectiva etapa ser activa, acção só será executada depois do atraso especificado, conforme se representa na figura 42.

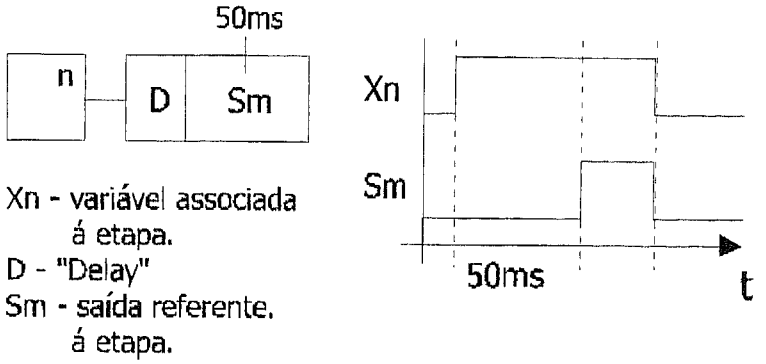


Figura 42 – Acção condicional com atraso

- Limitada

Este tipo de acção é executada logo que a etapa seja activa, mas apenas durante um intervalo de tempo que é especificado, conforme se representa na figura 43.

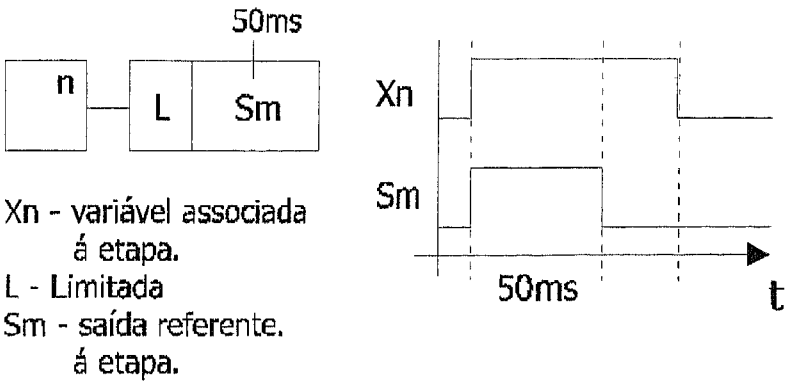


Figura 43 – Acção condicional limitada

Acções memorizadas

A saída do sistema correspondente a esta acção é activada assim que a etapa for activa e mantém-se actuada mesmo depois da desactivação da referida etapa. A

saída apenas poderá ser desactivada por uma outra etapa, onde se efectue uma referência explicita desta intenção. O diagrama temporal correspondente, é representado na figura 44.

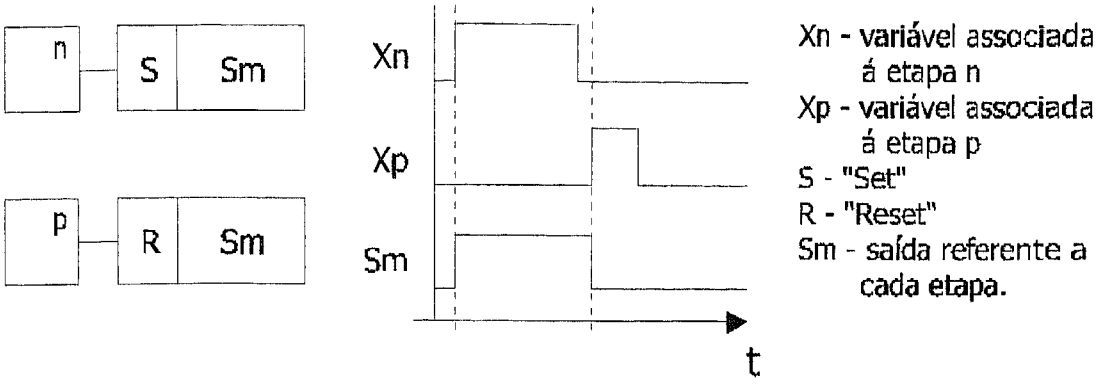


Figura 44 - Acção condicional memorizada

Receptividades

Uma receptividade é uma condição que está associada a uma transição e que determina a possibilidade ou não da evolução do sistema através da respectiva transição.

Uma receptividade exprime-se sob a forma de uma expressão booleana, e resulta da combinação das variáveis de entrada, etapas, saída, temporização....Para que uma transição seja transposta, além da etapa que a precede ter que estar activa é necessário que a respectiva receptividade seja verdadeira.

3.2 Descrição funcional

O editor de GRAFCET implementado, faz parte integrante da ferramenta desenvolvida, e constitui uma das opções para a edição do programa do SimAP.
A grande vantagem do editor de GRAFCET é permitir modelar de forma gráfica o programa que se pretende realizar, através da utilização do conjunto de símbolos descritos no item anterior.
O editor de GRAFCET desenvolvido pretende abranger mais uma das etapas de desenvolvimento de um sistema automatizado, sem a intenção de implementar de forma exaustiva a vasta variedade de soluções definida pela norma, CEI848, do GRAFCET. Por este motivo, e para uma maior facilidade de utilização, não foram implementadas algumas das funcionalidades previstas pelo GRAFCET, nomeadamente o que diz respeito a macro-etapas, transmissões e execução de tarefas paralelas.

Na descrição que se segue são apresentados os seguintes itens:
3.2.1 Edição do GRAFCET
3.2.2 Edição da estrutura de etapas/transições
3.2.3 Edição de acções e receptividades

3.2.1 Edição do GRAFCET

Tratando-se de um sistema didáctico, é importante que as ferramentas de edição gráfica, nomeadamente o conjunto de símbolos utilizados na edição do modelo do programa estejam facilmente acessíveis ao utilizador. Considerando este facto, foi desenvolvida uma interface com o utilizador, conforme se representa na figura 45, onde se apresentam a diversidade de símbolos disponíveis para a edição do programa.

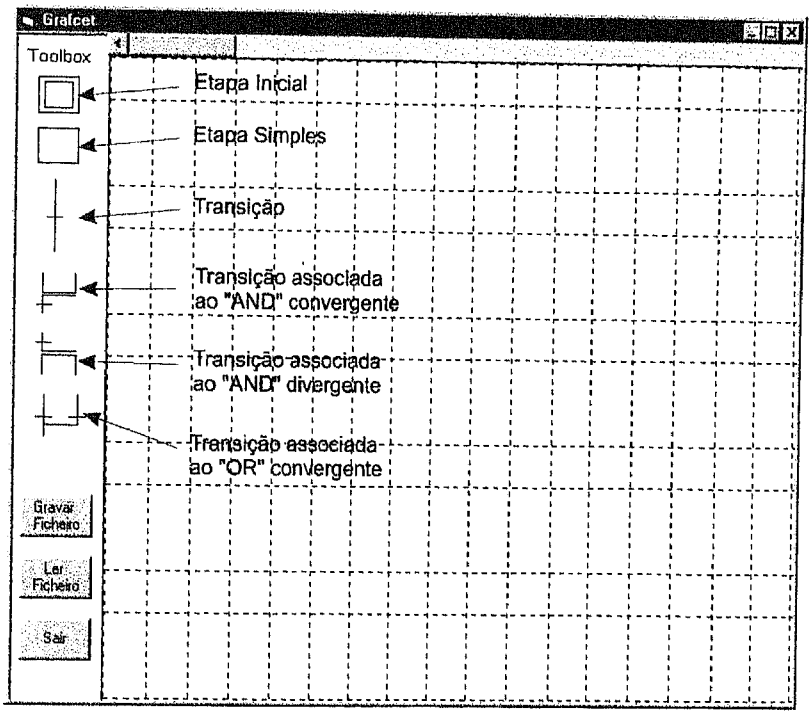


Figura 45 – GRAFCET: Interface com o utilizador

Com este conjunto de símbolos, pretendeu-se dotar este módulo com a possibilidade de editar programas que englobem ligações entre etapas do tipo, sequências, "AND" convergente, "AND" divergente, "OR" convergente, "OR" divergente, numa enorme variedade de combinações possíveis. Na figura 46, representam-se os diversos tipos de ligações que se podem realizar.

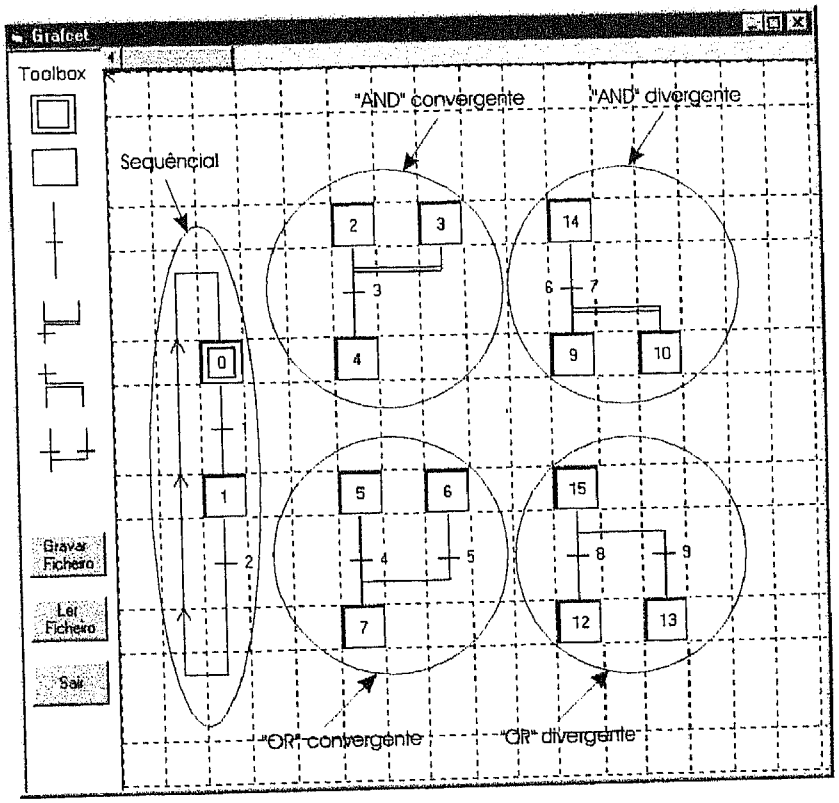


Figura 46 - GRAFCET: Tipo de ligações

3.2.2 Edição da estrutura de etapas/transições

No desenvolvimento do editor de GRAFCET, depois de definido os diversos tipos de símbolos necessários para a edição do modelo gráfico dos programas a realizar para o SimAP, houve a necessidade de se definir o tipo de estrutura gráfica mais adequada para a edição das etapas e transições do modelo. O tipo de estrutura ideal, seria aquela que permitisse ao utilizador uma liberdade total em termos de edição gráfica. No entanto, é necessário considerar que quanto maior for a flexibilidade de edição, mais difícil será a posterior interpretação gráfica para a geração directa do programa para o SimAP. Deste modo optou-se por uma relação flexibilidade de edição/facilidade de interpretação que permitisse, por um lado não impor restrições gráficas demasiadamente rígidas ao utilizador, nem tornar demasiadamente complexa a interpretação gráfica do modelo.

Tendo em vista a necessidade da interpretação do gráfico, começou-se por dividir a folha de desenho em células reservadas a etapas e transições, conforme se representa na figura 47. Note-se que apesar desta solução constituir, na realidade, uma restrição, permite ao utilizador estruturar de forma adequada o grafo que pretende implementar.

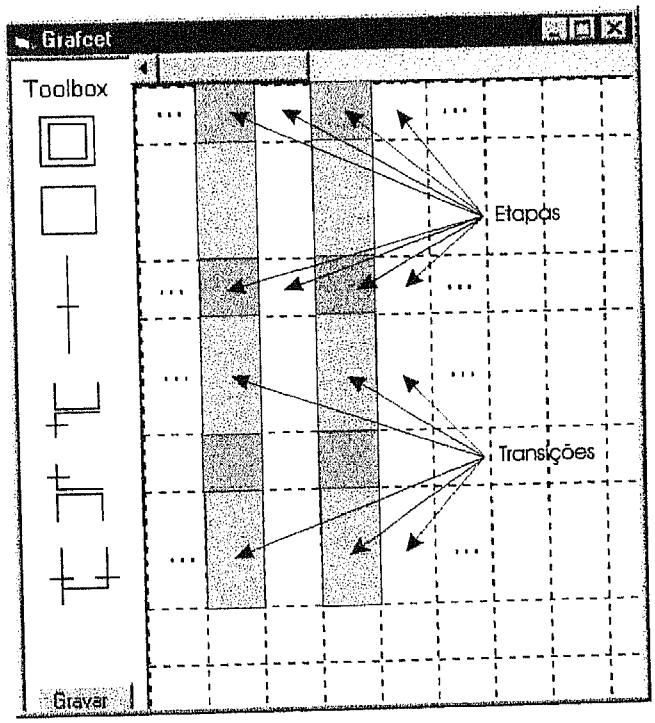


Figura 47 - GRAFCET: Células de desenho

Coloca-se agora o problema da inserção dos símbolos na página de desenho. As opções consideradas à partida foram duas: efectuar uma cópia do símbolo seleccionado na "toolbox", para a página de desenho ou redesenhar o símbolo seleccionado. Apesar de inicialmente se considerar que se poderia optar por uma ou outra solução para todos os símbolos, na realidade em função do símbolo seleccionado pelo utilizador, será utilizada uma das duas soluções apontadas. Deste modo, para os símbolos referentes à etapa inicial e etapa simples optou-se pela primeira solução, para os restantes símbolos pela segunda.

De facto, a utilização da cópia do símbolo, seria à partida a solução mais simples, uma vez que o desenho do símbolo obriga à determinação do conjunto de

coordenadas necessárias ao traçado dos segmentos de recta que constituem o desenho. No entanto, como iremos verificar de seguida, apesar desta ultima hipótese parecer mais complexa, em algumas circunstâncias, nomeadamente quando se pretende efectuar o "AND" e "OR" convergente ou divergente de mais de duas etapas, torna-se sem dúvida a mais versátil.

Comecemos em primeiro lugar por abordar a cópia de símbolos. Neste caso o que acontece é que, sempre que uma das figuras, portadoras dos símbolos correspondentes a etapa inicial ou etapa simples for seleccionado no "Toolbox", o símbolo dessa figura é copiado para uma das figuras carregada no arranque da aplicação do GRAFCET, sendo a numeração que lhe está associada atribuída de forma automática e consecutiva. Esta solução torna-se especialmente simples se posteriormente houver necessidade de se apagar o símbolo, uma vez que para memorizar a posição da imagem é suficiente armazenar duas coordenadas (x,y) da imagem, ao contrário do que se passa no desenho do símbolo em que é necessário guardar o conjunto de coordenadas de todos os segmentos de recta.

Para o traçado das transições, poder-se-ia optar por uma solução semelhante, no entanto teríamos que, ou limitar o numero máximo de etapas nas ligações do tipo "OR" e "AND" ou criar uma infinidade de imagens que permitisse abranger todas as hipóteses em termos de numero e disposição de símbolos. Por este motivo, para o traçado de transições optou-se por efectuar o desenho dos símbolos o que possibilita efectuar a ligação de qualquer numero de etapas conforme se representa na figura 48 e 49.

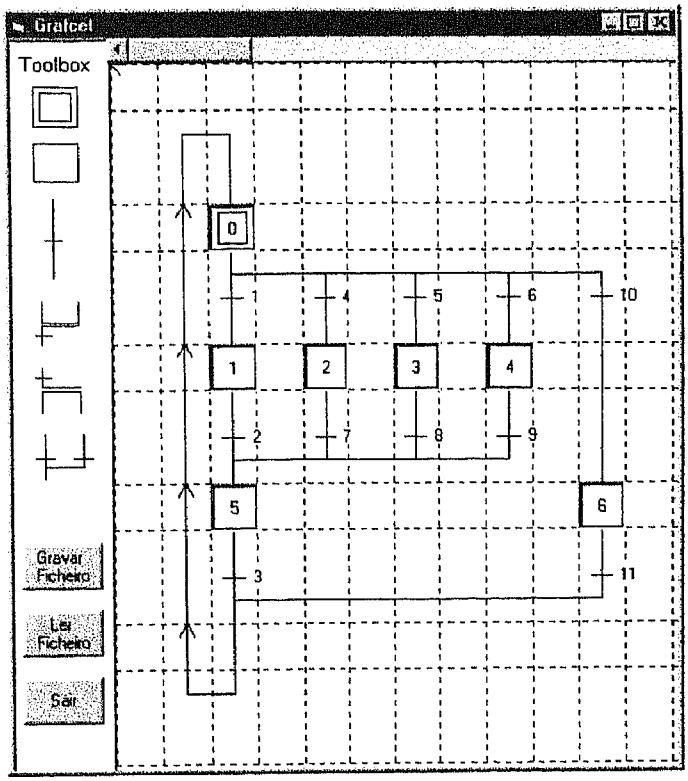


Figura 48 - GRAFCET: Desenho de símbolos (exemplo 1)

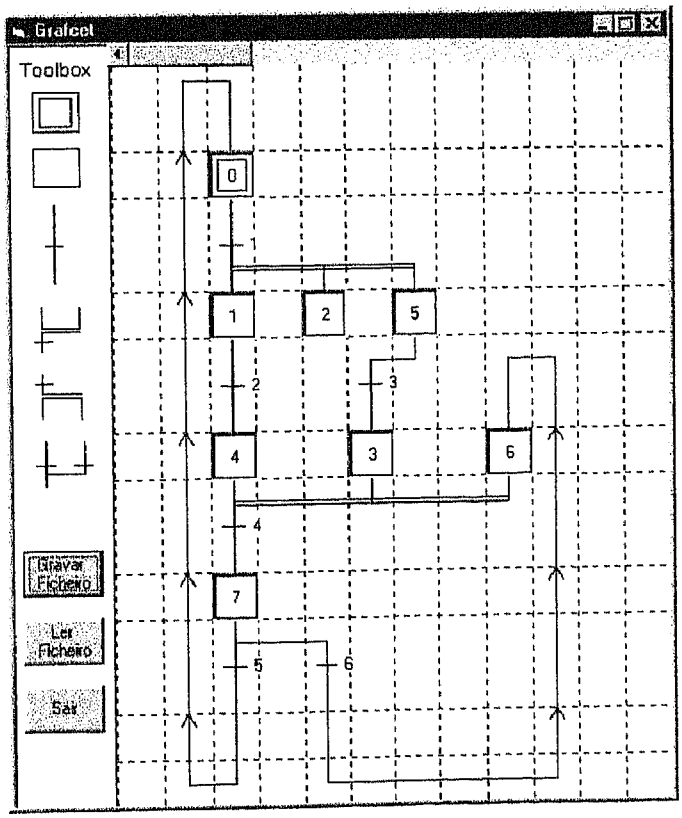


Figura 49 - GRAFCET: Desenho de símbolos (exemplo 2)

Tal como acontece nas etapas, a numeração das transições é realizada de forma automática.

Na ligação entre etapas, há a considerar certas regras de edição particulares a cada tipo de ligação. Em qualquer uma delas a inserção dos símbolos, etapas e transições, é realizada de forma independente, no entanto existem algumas particularidades a considerar. Numa ligação sequencial a inserção de uma transição na página de desenho impõe que tenham sido previamente inseridas no grafo as etapas que se pretendem interligar, uma vez que é necessária a indicação das etapas origem e destino, conforme se representa na sequência de ligações da figura 50.

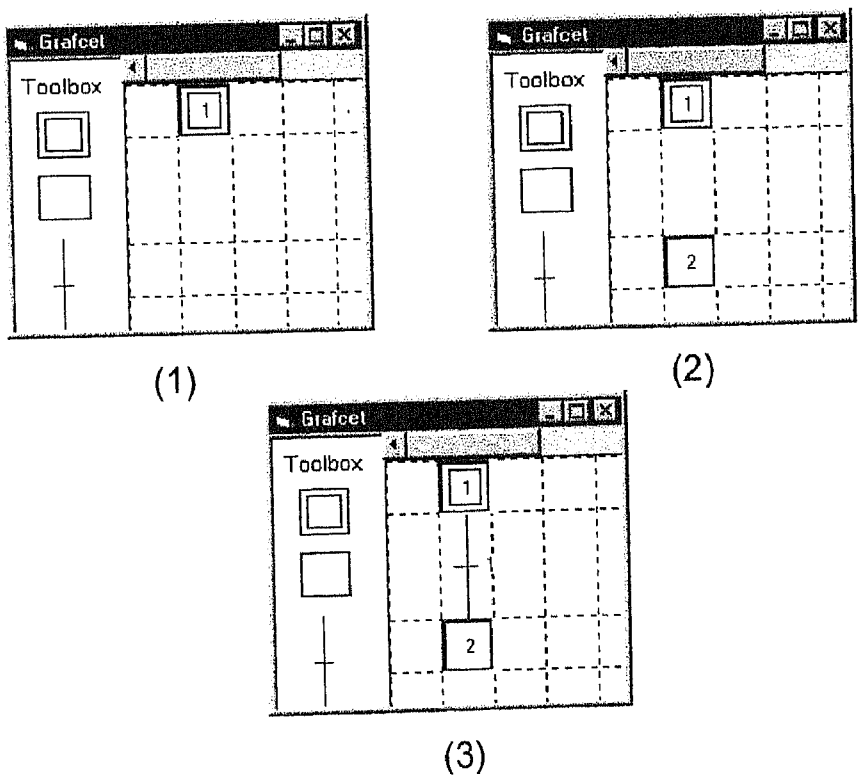


Figura 50 GRAFCET: Ligação entre etapas

As ligações tipo "AND" e "OR", exigem a existência de uma transição origem e de uma etapa destino. Na figura 51, representa-se a título de exemplo a sequência de ligações para uma ligação tipo "AND" convergente de duas etapas.

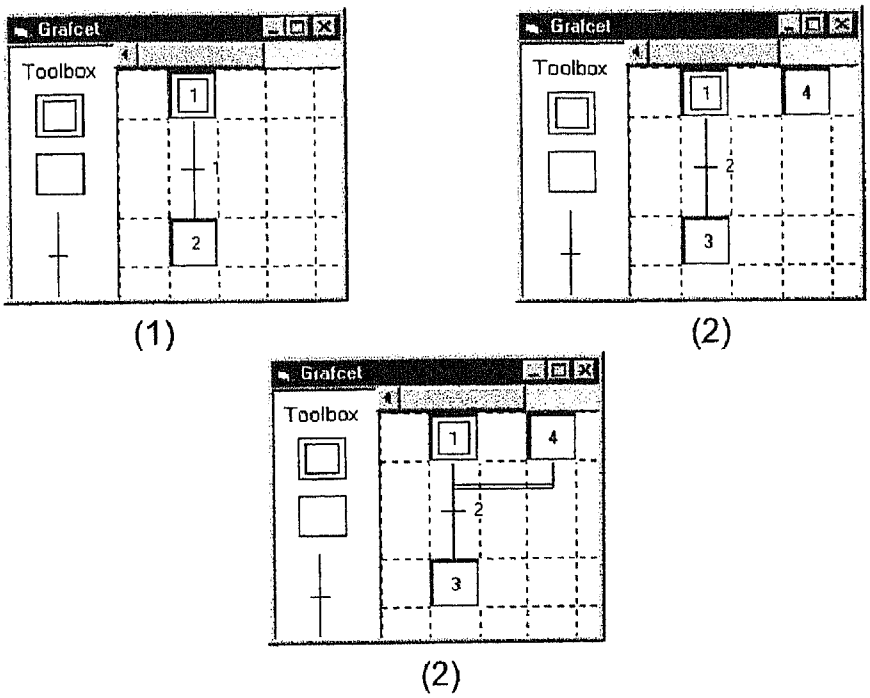


Figura 51 - GRAFCET: "AND" convergente

Outra das questão a considerar foi o tamanho máximo da página de desenho, uma vez que constitui uma das limitações ao tamanho do grafo a realizar. Para evitar que ficássemos limitados ao espaço visível imposto pelo ecrã do monitor, foi definida uma página de desenho maior e adicionada a possibilidade de se efectuar o seu varrimento na vertical e horizontal através das duas barras de varrimento representadas na figura 52.

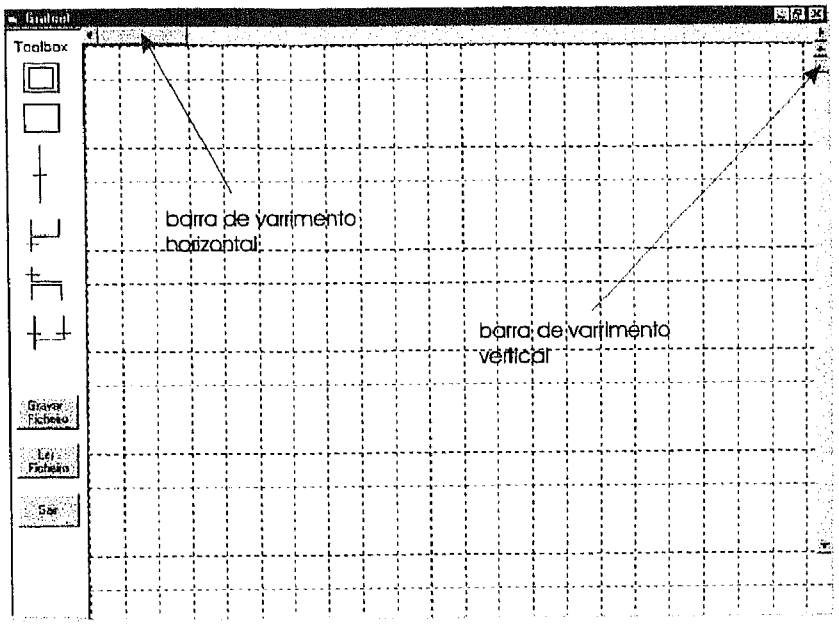


Figura 52 - GRAFCET: Varrimento vertical/horizontal

3.2.3 Edição de acções e receptividades

A definição de acções e receptividades é preponderante para interpretação do grafo realizado. A opção adoptada para a edição de acções e receptividades consiste na utilização de um editor de texto integrado no próprio editor de GRAFCET.

A edição gráfica do programa é independente da definição das acções e receptividades, no entanto é necessário que exista uma correspondência directa etapa/acção e transição/receptividade de modo a poder-se efectuar a interpretação do grafo. Deste modo, para diminuir o esforço de edição ao utilizador, sempre que se insira um símbolo referente a uma etapa ou transição no editor gráfico, surge automaticamente no editor de texto, intrínseco ao editor de GRAFCET, uma zona reservada à definição da respectiva acção ou receptividade identificada pela tipo de símbolo e numerador, conforme se representa na figura 53.

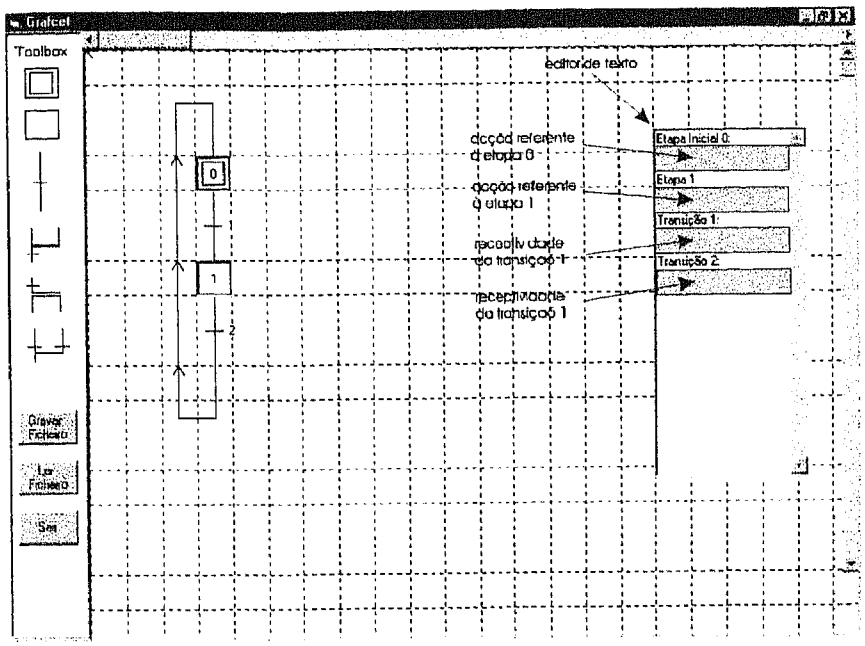


Figura 53 – GRAFCET: Edição de acções e receptividades

O utilizador terá apenas que introduzir o código do programa para a definição das etapas e receptividades. Nesta definição é utilizada a linguagem de programação usada no SimAP, de forma a uniformizar a linguagem utilizada.

Considere-se a título o exemplo as acções e receptividades definidas para o modelo gráfico representado na figura 54.

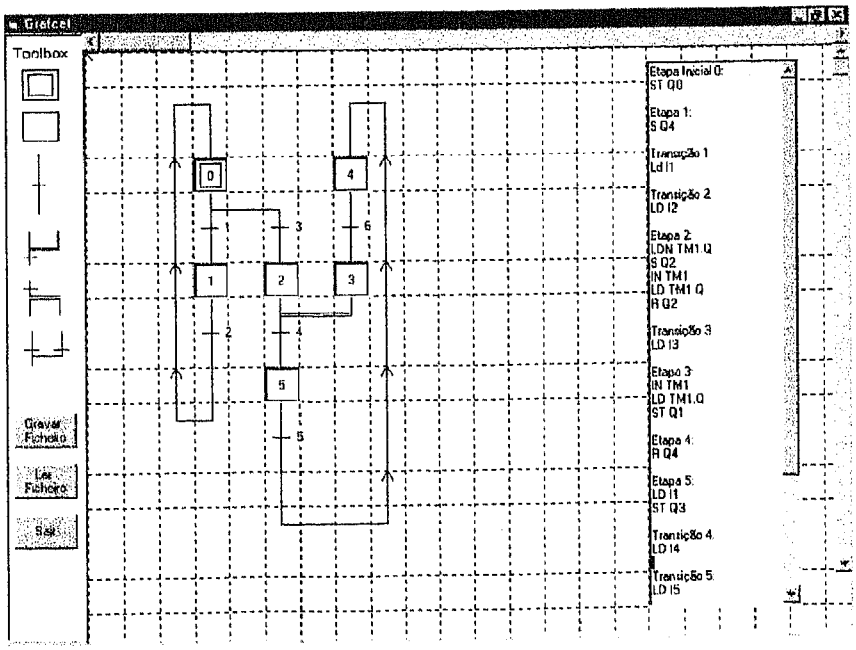


Figura 54 – GRAFCET: Edição de acções e receptividades (exemplo)

Para cada etapa no gráfico é definida uma acção. No exemplo que se apresenta são implementadas os diversos tipo de acções descritos no item anterior.

Na etapa inicial 0 é definida uma acção continua, ou seja, enquanto a etapa for activa a saída Q0 mantém-se actuada. Na etapa 1 e 4 define-se uma acção memorizada, neste caso, a saída 4 é actuada quando a etapa 1 for activa e mantém-se actuada até à activação da etapa 4, instante em que saída é desactivada. Na etapa 2 é definida uma acção condicional com atraso, a saída 1 é actuada assim que a etapa for activa e é desactivada passado um intervalo de tempo definido pelo temporizador 1. Na etapa 3 é definida uma acção condicional atrasada, quando a etapa for activa, apenas passado um instante de tempo definido pelo temporizador 1, é que a saída 1 é actuada. Na etapa 5, implementa-se uma acção do tipo condicional simples, em que se define que além da etapa ter que estar activa é necessária a activação da entrada 1. Relativamente às transições são definidas as receptividades, que no caso apresentado apenas dependem das entradas do SimAP, ou seja, para que as transições 1, 2, 3, 4, 5 sejam transpostas é necessário que as entradas, respectivamente I1, I2, I3, I4, e I5 sejam activadas.

O editor de GRAFCET possui ainda a versatilidade de em qualquer instante se poder alterar/corrigir o modelo realizado, com a vantagem de sempre que uma acção ou transição seja apagada, a respectiva definição da acção ou receptividade também o seja de forma automática, evitando-se que por esquecimento do utilizador seja impossibilitada a execução do programa, devido à complexidade do código entretanto gerado.

O modelo poderá também ser gravado em disco, e reutilizado em futuras edições.

3.3 Descrição da implementação

3.3.1 Representação interna do GRAFCET

O módulo de edição GRAFCET tem por objectivo, não só modelar o programa de forma gráfica, mas também permitir gerar, de forma automática, a partir do grafo implementado, o programa para o SimAP.

A melhor forma de se concretizar este objectivo será através da criação de uma estrutura interna de dados, representativa das etapas, transições e ligações etapa/transições, de suporte à representação gráfica editada. Para o efeito foram criados 2 tabelas de dados, uma designada por "tabela de registos", que armazena o tipo (etapa ou transição) e as coordenadas de todos os símbolos no desenho, e outra designada por "tabela_etapa_transição", que armazena os dados relativos às ligações efectuadas entre os referidos símbolos, bem como o tipo de ligação realizada, nomeadamente sequências, "AND" e "OR".

Comecemos por analisar a "tabela de registos". Como já verificamos, quando abordamos a edição da estrutura do GRAFCET, a página de desenho é dividida em células para a edição dos diversos símbolos disponíveis. Cada uma destas células foi identificada por duas coordenadas, x,y, de acordo a distribuição representada na figura 55.

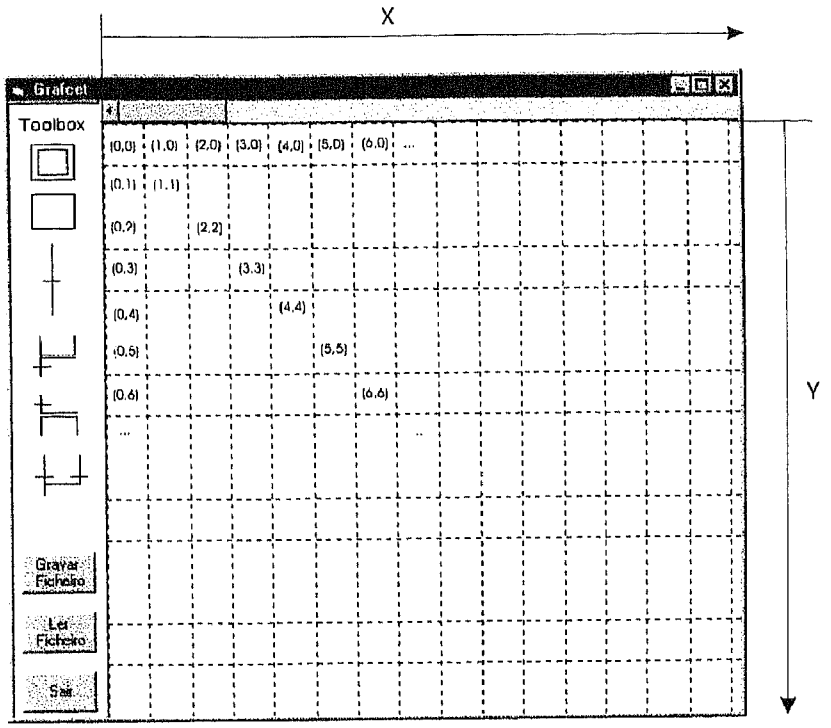


Figura 55 - GRAFCET: Identificação das células de desenho

Desta forma, qualquer símbolo introduzido na página de desenho pode ser facilmente localizado e destinguido dos restantes símbolos.

Sempre que um símbolo ocupa uma das células, na "tabela de registos" é criada uma nova posição do tipo "tabela_registos (x,y,z)", em que x e y representam respectivamente a coordenada horizontal e vertical da célula e z o valor 0, 1 e 2 ou 0 e 1, consoante o símbolo a guardar se trate, respectivamente de uma etapa ou transição. No caso do símbolo a guardar ser uma etapa, a informação armazenada na tabela será:

Tabela_registos (x,y,**0**) = tipo de registo

Tabela_registos (x,y,**1**) = caracter separador de campo de dados (99)

Tabela_registos (x,y,**2**) = numero da etapa

na qual,

Tipo de registo	
0	Etapa inicial

1	Etapa simples
2	Transição

Se tipo de registo = 2 (transição), obteremos:

Tabela_registos (x,y,**0**) = tipo de registo

Tabela_registos (x,y,**1**) = nº da transição

Consideremos a título de exemplo o conjunto de símbolos representados na figura 56.

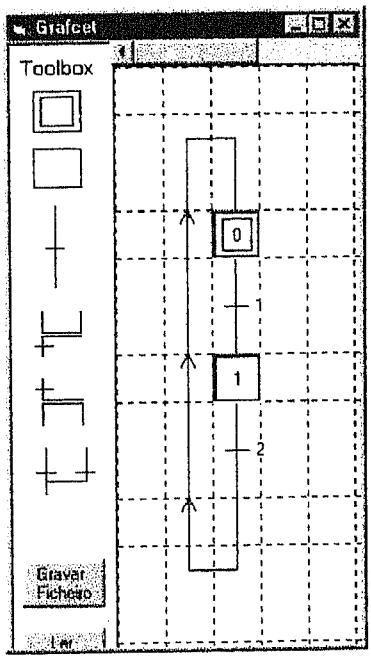


Figura 56 – GRAFCET: Ligação sequencial (exemplo 1)

Neste caso, a estrutura de dados resultante para cada símbolo seria:

Etapa inicial 0:

Tabela_registos (2,3,0) = 0; Tabela_registos (2,3,1) = 99; Tabela_registos (2,3,2) = 0

Etapa 1:

Tabela_registos (2,6,0)=1; Tabela_registos (2,6,1)=99; Tabela_registos (2,6,2)=1

Transição 1:

Tabela_registos (2,4,0) = 2; Tabela_registos (2,4,1) = 2

Transição 1:

Tabela_registos (2,5,0) = 2; Tabela_registos (2,5,1) = 2

Transição 2:

Tabela_registos (2,7,0) = 2 Tabela_registos (2,7,1) = 2

Transição 2:

Tabela_registos (2,8,0) = 2 Tabela_registos (2,8,1) = 2

Podemos verificar que informação referente a uma mesma transição é duplicada em duas células adjacentes. Isto deve-se ao facto do tamanho considerado para as células, se reportar ao tamanho de uma etapa, e por este motivo, uma transição, na realidade, ocupar duas células.

Note-se que, não se efectua qualquer referência às ligações entre etapas. Para isso, foi criada a já referida "tabela_etapa_transição". A estrutura de dados associada a esta tabela é mais complexa, devido à necessidade de se guardar um volume de informação maior, que está directamente relacionada com o tipo de ligação efectuada entre etapas. No caso de uma ligação do tipo sequencial, é apenas necessário guardar os dados referentes à etapa origem, etapa destino, o numero da transição que interliga as etapas e as coordenadas dos pontos referentes aos segmentos de recta utilizados no desenho da transição. Estas coordenadas terão especial interesse, como veremos mais tarde, na eventualidade de se pretender apagar/corrigir o grafo implementado. Se, por outro lado, a ligação entre etapas for do tipo "OR" ou "AND", além da informação anterior é também necessário guardar o tipo de ligação utilizada, nomeadamente se se trata de um "OR" convergente, "OR" divergente, "AND" convergente ou "AND" divergente, e ainda as coordenadas do conjunto de pontos utilizadas no desenho do respectivo símbolo.

A estrutura de dados obtida para a ligação sequencial representada na figura 57, seria a que se apresenta na tabela 28:

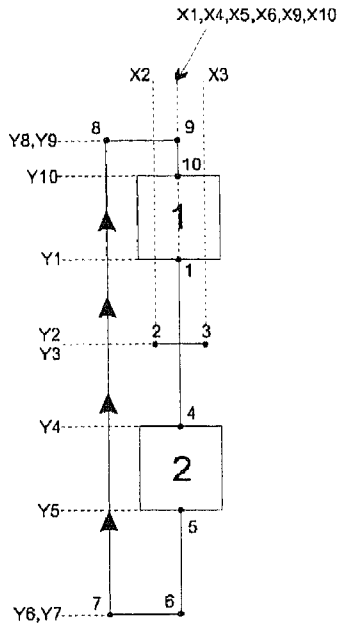


Figura 57 - GRAFCET: Ligação sequencial (exemplo 2)

Tabela 28 - GRAFCET: Estrutura interna de dados

Etapa origem	nº transição de saída	Coordenadas								Separador de dados	Etapa destino
0	1	X1	Y1	X2	Y2	X3	Y3	X4	Y4	99	1
1	1	X1	Y1	X2	Y2	X3	Y3	X4	Y4	99	99
1	2	X5	Y5	X6	Y6	...	...	X10	Y10	99	0
0	2	X5	Y5	X6	Y6	...	...	X10	Y10	99	99

As duas primeiras linhas da tabela correspondem à ligação entre a etapa 0 e 1. Na primeira linha está definida, a etapa origem da transição, neste caso a "0", a transição de saída dessa etapa, a "1", as coordenadas dos pontos correspondentes aos segmentos de recta utilizados no desenho da respectiva transição, um valor correspondente a um separador de dados, utilizado apenas para facilitar a posterior interpretação da tabela e no final a etapa destino da transição, a "1". A segunda linha é um pouco a repetição da anterior considerando-se a etapa 1 como origem, será como definir a ligação ao contrário, na qual se atribui à etapa final um código "99" que permite efectuar a distinção das duas linhas.

A informação, aparentemente redundante, referente à segunda linha da tabela irá revelar-se especialmente útil para se apagarem símbolos no desenho, assunto que será apresentado posteriormente. Para já, não nos esqueçamos, interessa-nos apenas analisar a estrutura de dados resultante da representação gráfica.

Na terceira e quarta linha, os dados armazenados são idênticos, mas agora referentes à ligação entre as etapa 5 e 6.

No caso das ligações "AND" e "OR" representadas respectivamente na figura 58 e 59, à estrutura de dados resultante é acrescentado o tipo de ligação realizada, conforme se apresentam respectivamente na tabela 29 e 30.

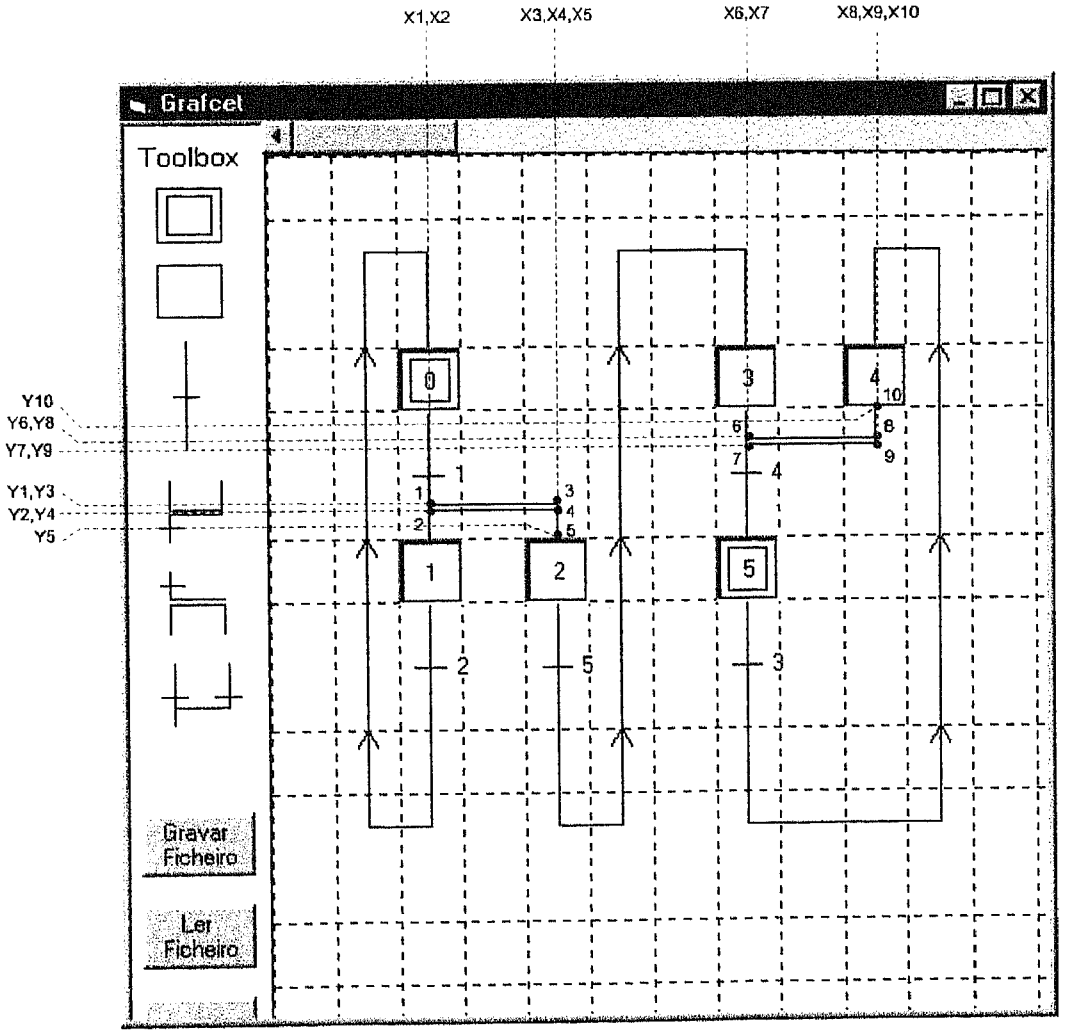


Figura 58 GRAFCET: Ligações tipo "AND"

Tabela 29 – GRAFCET: Estrutura interna de dados da ligação "AND"

Etapa origem	nº transição a que liga	Coordenadas										Separador de dados	Tipo ligação	Etapa destino
• "AND" Divergente														
2	1	X1	Y1	X3	Y3	X2	Y2	X4	Y4	X5	Y5	99	2	1
• "AND" Convergente														
4	4	X6	Y6	X7	Y7	X9	Y9	X8	Y8	X10	Y10	99	3	99

Nas quais,

Tipo de Ligação	
1	And Convergente
2	And Divergente
3	Or Convergente

Na tabela acima, representam-se exclusivamente as ligações "AND" divergente entre as etapas "1" e "2" e "AND" convergente entre as etapas "3" e "4". No entanto, para que seja possível efectuar estas ligações é necessário que previamente já exista uma transição entre etapas, de acordo com o representado na figura 58, entre as etapas "0" e"1" e etapas "3" e "5", das quais resulta toda uma estrutura de dados semelhante à descrita na ligação sequencial. Isto devido ao facto da nova etapa com a qual é realizada a operação lógica "AND" ficar ligada à referida transição.

Analisemos a tabela 29, na primeira linha, referente ao "AND" divergente, a etapa inicial é a "2". Uma vez que se pretende realizar o "AND" com a etapa "1", a nova etapa fica ligada, conforme se refere na tabela, à já existente transição de entrada "1" da etapa anterior ("1"). Em seguida são guardados todos as coordenadas do pontos utilizados no desenho do símbolo, o separador de dados, "99", o tipo de ligação efectuada, representada numericamente por "2" e no final o código "99" que indica o fim da linha.

A estrutura de dados referente ao "AND" convergente é em tudo semelhante ao descrito, conforme se pode verificar através da segunda linha da tabela.

Considere-se agora as ligações tipo "OR":

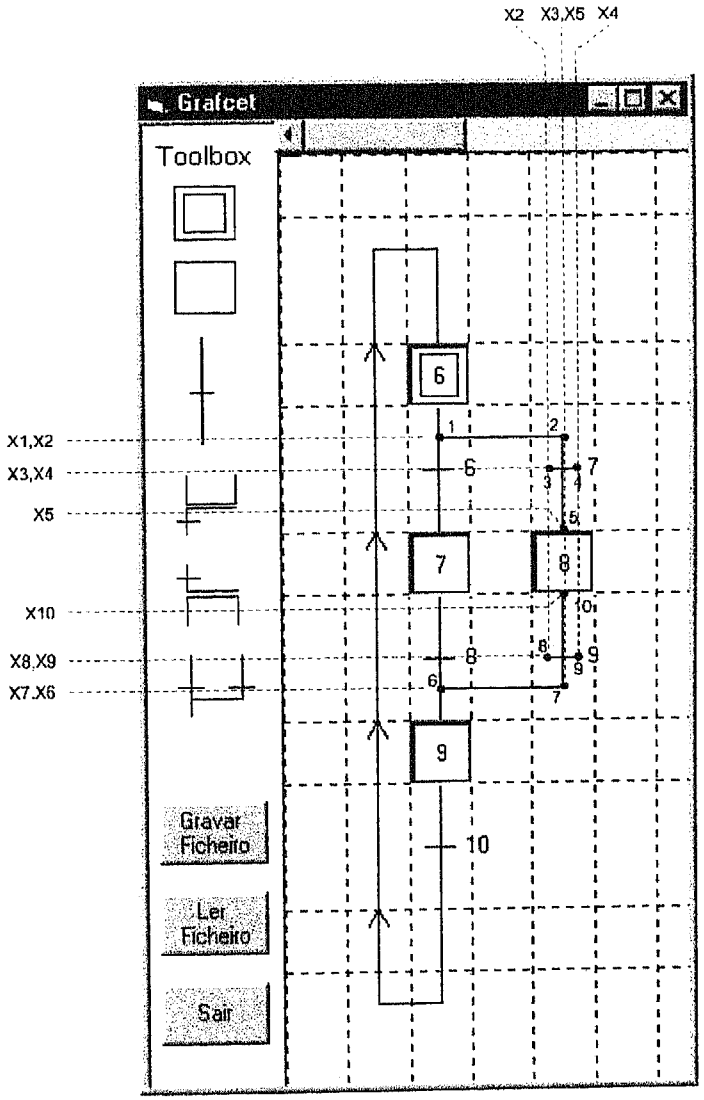


Figura 59 - GRAFCET: Ligação tipo "OR"

Tabela 30 - GRAFCET: Estrutura interna de dados da ligação "OR"

Etapa origem	nº transição a que liga	Coordenadas							Separador de dados	Tipo de ligação	Separa. Dados	nº transição saída	Fim dados
• "OR" Convergente													
8	8	X10	Y10	...	X6	Y6	X9	Y9	99	3	99	9	99
• "OR" Divergente													
6	7	X1	Y1	...	X5	Y5	X4	Y4	99	8	-	-	-
8	7	X1	Y1	...	X5	Y5	X4	Y4	99	99	-	-	-

Como se pode verificar através da tabela 30, a estrutura de dados associada à ligação "OR" tem algumas particularidades. No "Or" divergente, primeira linha, além dos campos de dados já referidos para a ligação "AND", acrescenta-se o numero a sua transição de saída. Esta necessidade deve-se ao facto, de neste tipo de ligações surgirem mais um conjunto de transições que necessitam de ser identificadas aquando da interpretação do programa.

Na ligação tipo "OR" divergente, tudo se passa como se na mesma etapa, "6", fossem realizadas "duas" ligações sequenciais, uma referente à ligação entre as etapas "6" e "7" e outra referente à ligação entre as etapas "6" e "8". A informação, referentes a este tipo de ligação, são representados na segunda e terceira linha da tabela e são em tudo idênticos à já apresentada ligação sequencial. Na segunda linha é definida a etapa origem, "6", o numero da transição de saída, "7", as coordenadas dos pontos representativos do grafismo do símbolo e finalmente a etapa destino, "8". Na terceira linha é representada a mesma informação, mas considerando a ligação efectuada da etapa "8" para a etapa "6".

3.3.2 Gravação/Reedição do GRAFCET

É necessário não descuidar que a representação do GRAFCET em memória, descrita no item anterior, mantém-se válida apenas durante a execução do GRAFCET, assim que esta aplicação for encerrada toda a informação será perdida. Para evitar que isto aconteça, e que um dado modelo possa ser utilizado em futuras aplicações do GRAFCET, é necessário guardar-se em disco toda a informação referente aos modelos realizados.

O método adoptado consistiu em guardar num ficheiro em disco, toda a informação referente à estrutura de dados resultante da edição de um modelo gráfico, nomeadamente das tabelas, acções e receptividades.

O ficheiro guarda o conteúdo da tabela_registos, que contém a informação relativa ao conteúdo de todas as células do desenho, o conteúdo da tabela_etapa_transição, com a descrição das ligações entre etapas, e o conteúdo do editor de GRAFCET, com as definições das acções e receptividades.

Consideremos a título de exemplo o modelo representada na figura 60.

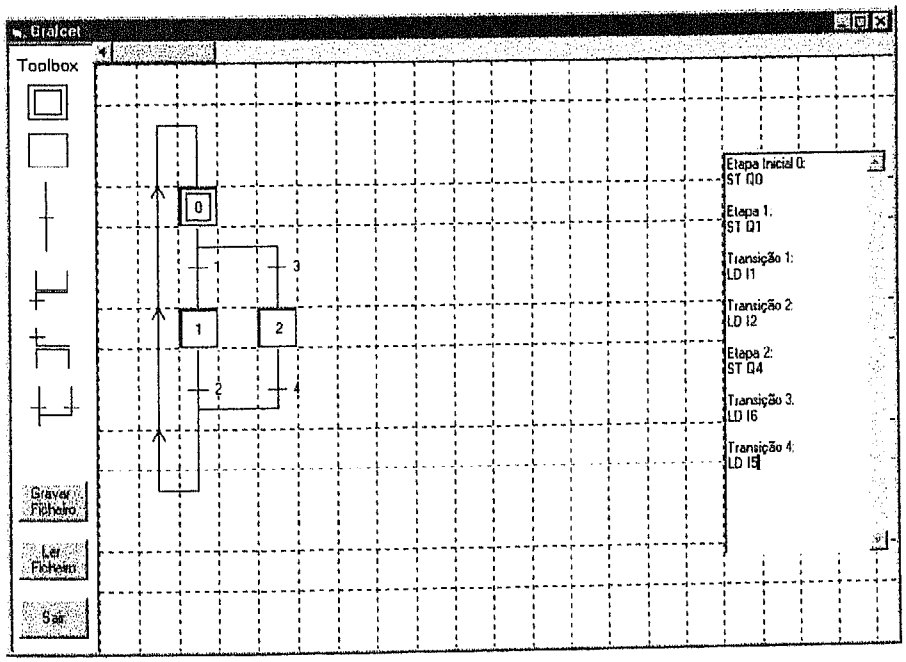


Figura 60 GRAFCET: Exemplo

A estrutura interna de dados em memória resultante seria:

Tabela_registos

Etapa 0: tabela_registos (2,3,0) = 0; tabela_registos (2,3,1) = 99; tabela_registos (2,3,2) = 0 Etapa 1: tabela_registos (2,6,0) = 1; tabela_registos (2,6,1) = 99; tabela_registos (2,6,2) = 1

Tabela_etapa_transição

Etapa origem	nº transição de saída	Coordenadas								Separador de dados	Etapa destino
• Ligação da etapa 0 à etapa 1											
0	1	1250	1750	1250	3000	1150	1350	-	-	99	1
1	1	1250	1750	1250	3000	1150	1350	-	-	99	99
• Ligação da etapa 1 à etapa 0											
1	2	1250	3500	1250	4750	750	4750	750	...	99	0
0	2	1250	3500	1250	4750	750	4750	750	...	99	99
• Ligação da etapa 0 à etapa 2											
0	4	1250	2150	2250	2150	2250	3000	2150	...	99	2
2	4	1250	2150	2250	2150	2250	3000	2150	...	99	0

Etapa origem	nº transição a que liga	Coordenadas					Separador de dados	Tipo de ligação	Separa. Dados	nº transição saída	Fim dados
• Ligação OR convergente da etapa 1 com a etapa 2											
2	2	1250	1750	1250	3000	...	99	3	99	4	99

O ficheiro guardado em disco, cujo parte do conteúdo se representa na tabela 31, resulta da consulta da estrutura de dados em memória. Sempre que se guardar um modelo, no ficheiro são criadas 3 zonas de dados, uma referente ao conteúdo da tabela etapa_transição, outra referente ao conteúdo da tabela de registos e por fim outra referente ao conteúdo do editor.

Tabela 31 - GRAFCET: Ficheiro guardado em disco

"**tb_et_tr**"

" 0 1 1250 2000 1250 2250 1250 2250 1250 2250 1250 2250 1250 3000 1125 2500 1375 2500 99 1 0"

" 1 1 1250 2000 1250 2250 1250 2250 1250 2250 1250 2250 1250 3000 1125 2500 1375 2500 99 99 0"

" 1 2 1250 3500 1250 3750 1250 3750 1250 3750 1250 3750 1250 4500 1125 4000 1375 4000 1250 4500 ... 99 0"

" 0 2 1250 3500 1250 3750 1250 3750 1250 3750 1250 3750 1250 4500 1125 4000 1375 4000 1250 4500 ... 99 99 0"

" 0 3 1250 2000 1250 2250 1250 2250 2250 2250 2250 2250 2250 3000 2125 2500 2375 2500 99 2 0"

" 2 3 1250 2000 1250 2250 1250 2250 2250 2250 2250 2250 2250 3000 2125 2500 2375 2500 99 99 0"

" 2 2 1250 4250 2250 4250 2250 4250 2250 3500 2125 4000 2375 4000 99 3 99 4 99 0"

"**tb_registos**"

" **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** ...

" 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 ...

" **99 0 0** 99 0 0 **99 0 0 0 99 0 2 1 0 2 1 0 1 99 1 2 2 0 2 2 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0** ...

" 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 **99 0 0** 99 0 0 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 **2 3 0 2 3 0 1 99 2 2 4 0 2 4 0** 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 ...

" 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 99 0 0 ...

...

"**Cont_et_tr**"

"7"

"Editor"

"Etapa Inicial 0:

ST Q0

```
Etapa 1:
ST Q1
Transição 1:
LD I1
Transição 2:
LD I2
Etapa 2:
ST Q4
Transição 3:
LD I6
Transição 4:
LD I5
"
```

Cada uma das células do gráfico é representada por 3 variáveis, tabela_registos (x,y,0), tabela_registos (x,y,1) e tabela_registos (x,y,2). Se a célula estiver vazia, cada um dos valores anteriores é respectivamente 99, 0, 0. Na tabela 31, a zona correspondente a "tb_registos", pode ser facilmente interpretada se considerarmos que cada 3 valores corresponde a uma célula.

A zona referente ao "editor" é uma copia do conteúdo do editor de texto do GRAFCET. Outro dos valores guardados corresponde ao conteúdo do cont_et_tr (contador_etapas_transições), utilizado no programa para identificar o numero de linhas da tabela_etapa_transição, de modo a possibilitar futuras manipulações desta tabela.

Depois do ficheiro ser guardado, o programa pode ser encerrado sem se perder quaisquer dados referentes ao grafo realizado.

Em futuras execuções do editor de GRAFCET, o ficheiro pode ser novamente aberto de modo a reconstituir-se o grafo guardado sob a forma de texto.

Quando se efectua abertura do ficheiro, a estrutura de dados em memória é novamente reconstituída através dos dados armazenados. Cada uma das tabelas é facilmente reconstituída, através da leitura da respectiva zona de dados no ficheiro. As acções e receptividade são lidas directamente para o editor de texto de GRAFCET.

O desenho de cada um dos símbolos e ligações são obtidos através da leitura da tabela_de_registos e da tabela_etapa_transição, logo após a leitura do ficheiro em disco. Futuras alterações do programa, nomeadamente a inserção de novos símbolos, resultam no conjunto de procedimentos em tudo idêntico ao descrito no item anterior.

3.3.3 Edição de símbolos

Na edição do modelo gráfico do programa é importante que seja possível efectuarem-se alterações/correcções do modelo, caso contrário eventuais modificações do programa ou enganos de edição, obrigaria à criação de um novo modelo perdendo-se todo o trabalho até então realizado.

Para se apagar um símbolo ou conjunto de símbolos é necessário efectuar a actualização da página de desenho e eliminar qualquer referência na estrutura de dados criada internamente, nomeadamente na "tabela_de_registos" e "tabela_etapa_transição".

Na eliminação de um símbolo, é necessário tomar em consideração, não só o símbolo em causa, mas também o conjunto de símbolos que lhe estão associados. Por exemplo, quando se apaga uma etapa pertencente a uma ligação sequencial, é também necessário apagarem-se as respectivas transições de entrada e saída, uma vez que, como já verificamos, em termos de edição gráfica da estrutura das etapas e transições, não podem existir transições sem que seja definida uma etapa origem e destino, caso contrário seria necessário adoptar um mecanismo de actualização da estrutura de dados interna, que além de eliminar a informação correspondente aos símbolos apagados, efectuasse a correspondência dos dados remanescentes com futuros símbolos e ligações, obtendo-se como resultando uma estrutura de dados demasiado complexa. Consideremos a título de exemplo a ligação sequencial representada na figura 61, bem como a estrutura de dados interna resultante, representada na tabela 32 e 33, correspondentes respectivamente à "tabela_de_registos" e "tabela_etapa_transição".

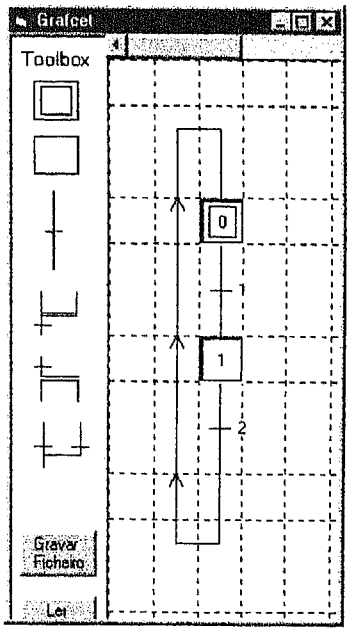


Figura 61 - Ligação sequencial (exemplo)

Tabela 32 - Estrutura de dados da "tabela_de_registos"

Etapa 0:
Tabela_registos (2,3,0) = 0; Tabela_registos (2,3,1) = 99; Tabela_registos (2,3,2) = 0;
Etapa 1:
Tabela_registos (2,6,0) = 1; Tabela_registos (2,6,1) = 99; Tabela_registos (2,6,2) = 1;
Transição 1:
Tabela_registos (2,4,0) = 2; Tabela_registos (2,5,1) = 2
Transição 2:
Tabela_registos (2,7,0) = 2; Tabela_registos (2,8,1) = 2

Tabela 33 - Estrutura de dados da "tabela_transições"

Etapa origem	nº transição de saída	Coordenadas								Separador de dados	Etapa destino
0	1	1250	1750	1250	3000	1150	1350	-	-	99	1
1	1	1250	1750	1250	3000	1150	1350	-	-	99	99
1	2	1250	3500	1250	4750	750	4750	750	...	99	0
0	2	1250	3500	1250	4750	750	4750	750	...	99	99

Ao se apagar a etapa 1, de acordo com o que foi referido, serão também apagadas as transições 1 e 2. Em termos da estrutura interna de dados, a informação correspondente a estes símbolos será eliminada, obtendo-se como resultado a actualização da "tabela_de_registos" e "tabela_etapa_transição", de acordo com o representado respectivamente na tabela 34 e 35.

Tabela 34 - Tabela_de_registos

Etapa 0:

Tabela_registos (2,3,0) = 99; Tabela_registos (2,3,1) = 0; Tabela_registos (2,3,2) = 0

Etapa 1:

Tabela_registos (2,6,0) = 1; Tabela_registos (2,6,0) = 99; Tabela_registos (2,6,2) = 1;

Transição 1: Tabela_registos (2,4,0) = 99 Tabela_registos (2,5,1) = 0

Transição 2: Tabela_registos (2,7,0) = 99 Tabela_registos (2,8,1) = 0

Tabela 35 - Tabela_etapa_transição

Etapa origem	nº transição de saída	Coordenadas								Separador de dados	Etapa destino
99	-	-	-	-	-	-	-	-	-	-	-
99	-	-	-	-	-	-	-	-	-	-	-
99	-	-	-	-	-	-	-	-	-	-	-
99	-	-	-	-	-	-	-	-	-	-	-

Conforme se verifica na tabela 34, é eliminada qualquer referência a símbolos nas posições (2,3), (2,4), (2,5), (2,7) e (2,8), correspondentes anteriormente à etapa 1, transição 1 e transição 2. O caracter "99" é utilizado apenas como identificador de linha "vazia" para posteriores manipulações da tabela.

Em termos de edição gráfica é efectuada a actualização de toda a página de desenho através da consulta da estrutura interna de dados, ou seja, serão

representados os símbolos indicados na "tabela_de_registos" e desenhadas todas as ligações indicadas na "tabela_etapa_transição". No exemplo em causa, devido à actualização das referidas tabelas apenas será representada a etapa 1, uma vez que a única informação válida encontrada é:

Etapa 1:
Tabela_registos (2, 6, 0)=1; Tabela_registos (2, 6, 0)=1; Tabela_registos (2, 6, 0)=1

Outra hipótese considerada, seria a de actualizar apenas parte da página apagada, no entanto torna-se mais simples efectuar a leitura de toda a estrutura interna de dados, do que efectuar a leitura dos dados referentes à informação já apagada.

O mecanismo de actualização da estrutura interna de dados consiste no seguinte, sempre que é seleccionada a etapa a apagar é chamada uma função em que se passa como parâmetros as coordenadas X e Y da referida etapa. Através da consulta da "tabela_de_registos" nomeadamente da sua posição "tabela(X, Y, 0)=0" verifica-se o tipo de símbolo armazenado, ("0"- etapa inicial). Em seguida efectua-se a procura na "tabela_etapa_transição" de qualquer referência a este símbolo de modo a constatar-se a existência de eventuais ligações, nomeadamente de transições de entrada e saída, que como já verificamos terão que ser apagadas.

O mecanismo de procura pode tornar-se mais complexo se, por exemplo, uma das transições a apagar for utilizada para a representação de uma ligação tipo "AND" de duas etapas, conforme se representa na figura 62

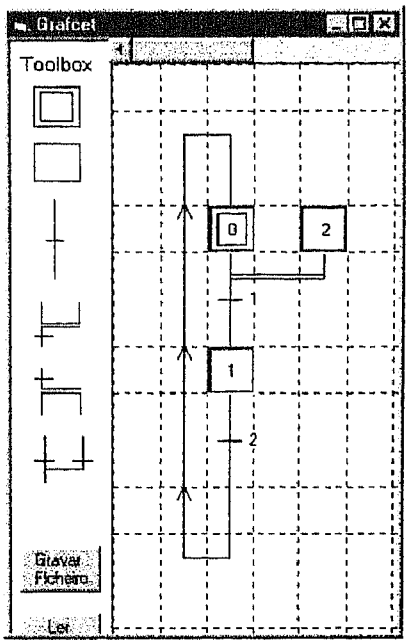
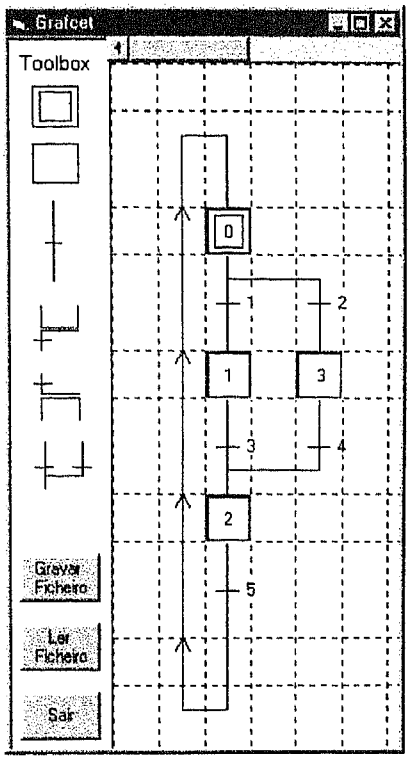


Figura 62 - Ligação "AND convergente" de duas etapas

Na realidade, quando se apaga uma etapa e em seguida se efectua a procura das transições de entrada e saída dessa etapa, é realizada também a procura de eventuais tipos de ligação realizadas com essas transições. No caso de existirem será tudo apagado. No exemplo da figura 62, quando se apaga a etapa 0, serão apagadas as respectivas transições 1 e 2 e consequentemente a ligação "AND" da etapa 0 com a 2, uma vez que para se manter necessitava da transição 1. Ficam apenas representadas as etapas 1 e 2. Associada a esta actualização gráfica está a correspondente actualização da estrutura interna de dados, em tudo idêntico ao anteriormente descrito. Note-se que se for apagada a etapa 2, conjuntamente com ela apenas será apagada a ligação "AND". A transição 1 mantém-se uma vez que a representação gráfica deste símbolo não depende da referida ligação.

Se fosse realizada uma ligação "OR" o processo de actualização gráfica e dados seria realizada da mesma forma. Consideremos a figura 62.



Segundo o que se acabou de referir, se for apagada a etapa 0, serão apagadas as transições 1, 2 e 5. Se for apagada a etapa 2, serão apagadas as transições 5, 3 e 4. Se for apagada a etapa 1, serão apagadas as transições 1, 2, 3 e 4. No entanto se for apagada a etapa 3, apenas serão apagadas as transições 2 e 4. Note-se que a actualização gráfica origina sempre a actualização da estrutura interna de dados.

Consideremos agora a eliminação de uma transição. Ao contrário do que acontecia anteriormente, em que o apagar de uma etapa implicaria a necessidade de se apagar eventuais transições que a elas estivessem ligadas, o apagar de uma transição nunca origina a eliminação de uma etapa. Consideremos a titulo de exemplo a ligação sequencial representada na figura 63.

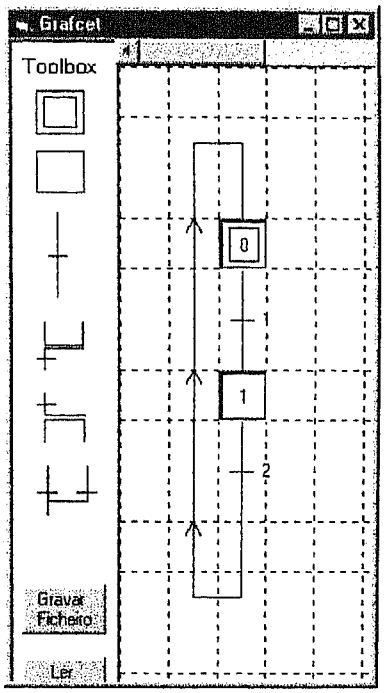


Figura 63 - GRAFCET: Ligação sequencial (exemplo)

Quando se pretender apagar a transição 1 o resultado final consistirá apenas na eliminação deste símbolo, uma vez que em termos de edição gráfica nenhum dos outros símbolos depende desta transição.

Se esta transição for utilizada na representação gráfica de um "AND" ou "OR", tudo se passa como já anteriormente foi referido, é necessário efectuar a procura das ligações realizadas com esta transição de modo a que estas também sejam apagadas.

No exemplo representado na figura 64, se a transição 6 ou 7 fosse apagada, seria também eliminado respectivamente o "AND" divergente e convergente da etapa 5 com a etapa 7 e 8. Mantendo-se as etapas e efectuando-se a actualização da estrutura interna de dados.

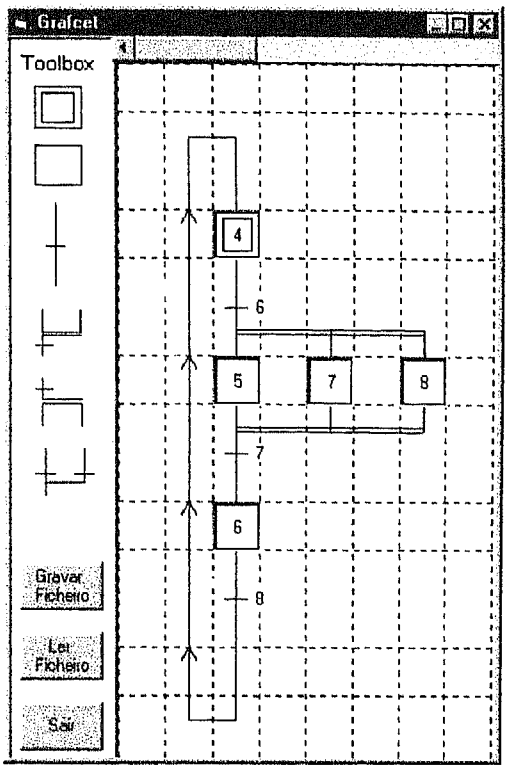


Figura 64 – Ligação tipo "AND" convergente e divergente

3.4 Geração do programa para o SimAP

A geração do programa para o SIMAP resulta da interpretação do modelo gráfico obtido do editor de GRAFCET. Para a obtenção do programa existem dois métodos a considerar:

- O método assíncrono, mais simples de implementar, mas que no entanto não resolve eventuais situações de conflito nem de reactivação de etapas que possam ocorrer.
- O método síncrono, mais complexo, mas aplicável em qualquer caso.

A implementação de qualquer um dos métodos passa por duas fases. Na primeira é determinado o estado actual do sistema, na segunda é determinado o estado actual das saídas.
O estado actual do sistema é memorizado através de um conjunto de flags auxiliares, M0...M99, pertencentes ao vector de registos em memória descrito no capítulo anterior, de forma a que:

Se a etapa Ei estiver activa Mi = 1

Se a etapa Ei estiver desactiva Mi = 0

3.4.1 Método assíncrono

Este método é aplicável desde que no GRAFCET não existam estados de conflitos nem de reactivação de etapas, nestes casos as regras de evolução do GRAFCET não são respeitadas, nomeadamente no que se refere ao seguinte:

- No caso de uma etapa ser activada e desactivada simultaneamente (situação de conflito) então essa etapa deverá permanecer activa.
- No caso de existirem transições disparáveis simultaneamente (reactivação de etapas), serem disparadas em simultâneo.

Considerando estes aspectos, a aplicação deste método terá que resultar do compromisso entre flexibilidade de utilização, relacionado com a necessidade de se restringir a ocorrência dos conflitos referidos, e a simplicidade de implementação como se constata através do algoritmo a seguir descrito:

Algoritmo de implementação do método assíncrono

1º determinação do estado do sistema

- Para cada transição Ti do GRAFCET

  Se Ti está disparável então desactiva-se as etapas de entrada (Mi) de Ti

  E activa-se as eatpas de saída de Ti

2º Determinação do valor das saídas

- Para cada etapa Ei do GRAFCET, executam-se as acções que lhe estão associadas

Consideremos a título de exemplo o grafo representado na figura 65.

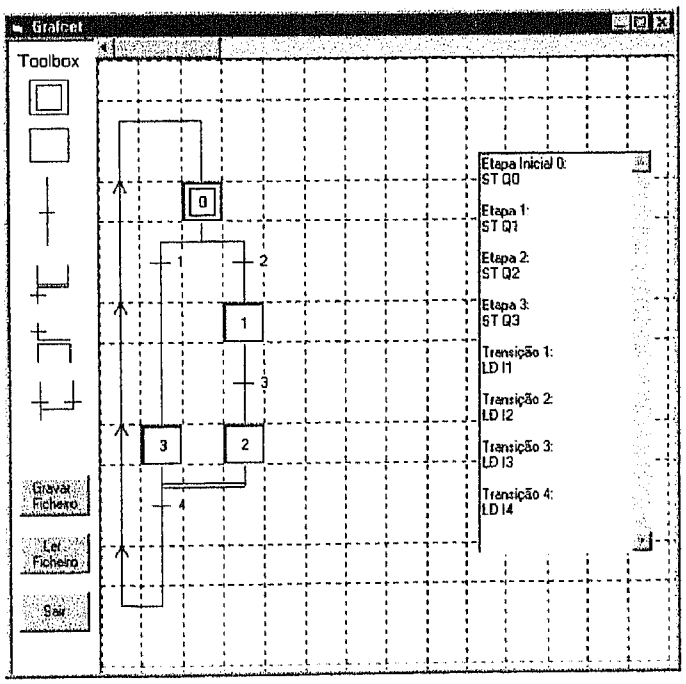


Figura 65 - Interpretação do GARFCET: Exemplo

De acordo com o método descrito o programa gerado seria:

- 1º Determinação do estado do sistema

Para a transição 1 LD M0 AND I1 R M0 S M3 Para a transição 2 LD M0 AND I2 R M0 S M1	Para a transição 3 LD M1 AND I3 R M1 S M2 Para a transição 4 LD M2 AND M3 AND I4 R M3 R M2 S M0

- 2º Determinação do estado das saídas

```
LD M0
ST Q0

LD M1
ST Q1

LD M2
ST Q2

LD M3
ST Q3
```

Perante um estado de conflito, originado por duas transições simultaneamente disparáveis a partilharem a mesma etapa de entrada, conforme se representa na figura 66, em que a etapa "0" está activa e as transições 1 e 2 estão validadas.

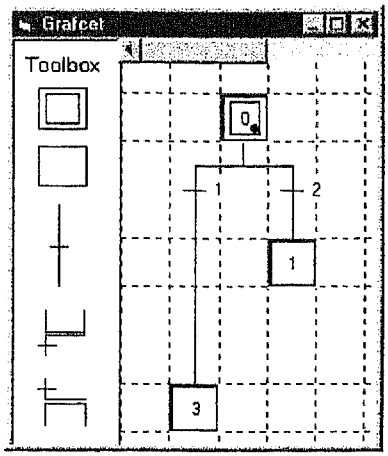


Figura 66 - GRAFCET: Estado de conflito

Neste caso ocorreria um erro, isto porque, admitindo que inicialmente estaríamos com M0=1, M1=0 e M3=0, de acordo com as regras de evolução do GRAFCET, o estado seguinte deveria ser M0=0, M1=1 e M3=1 e não realidade, considerando o excerto de programa que se segue, não o é, senão vejamos:

Para a transição 1 LD M0 AND I1 R M0 S M3	| M0 = 1; M1 = 0; M3 = 0 | M0 = 0; M1 = 0; M3=1
Para a transição 2 LD M0 AND I2 R M0 S M1	| M0 = 0; M1 = 0; M3=1 **| M0 = 0; M1 = 0; M3=1**

Obteríamos como resultado final M0=0, M1=0 e M3=1, o que está errado

Outra fonte de erro, seria a ocorrência da activação e desactivação simultânea da mesma etapa, conforme se representa na figura 67, em que temos M0=1, M1=1, M2=0 e as receptividades das transições 2 e 3 validadas:

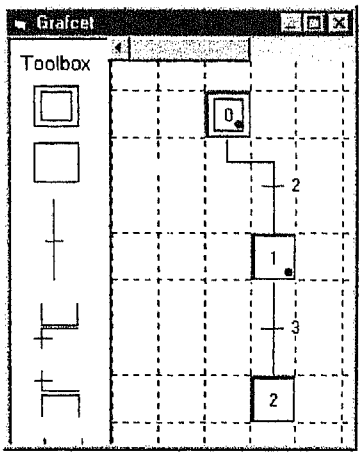


Figura 67 - GRAFCET: Reactivação de etapas

Neste caso, de acordo com as regras do GRAFCET, deveríamos obter M0=0, M1=1 e M2=1, no entanto, considerando o excerto do programa a seguir descrito:

Para a transição 1	
LD M0	| M0 = 1; M1 = 0; M2 = 0
AND I1	
R M0	
S M3	| M0 = 0; M1 = 1; M2=0
Para a transição 2	
LD M0	| M0 = 0; M1 = 1; M2=0
AND I2	
R M0	
S M1	**| M0 = 0; M1 = 1; M2=0**
Para a transição 3	
LD M1	**| M0 = 0; M1 = 1; M2=0**
AND I3	
R M1	
S M2	**| M0 = 0; M1 = 0; M2=1**

Obteríamos como resultado final M0=0; M1= 0 e M2=1, o que estaria errado.

Apesar da simplicidade deste método é necessária ter em consideração a possibilidade da ocorrência de erros.
Para ultrapassar estes problemas deve-se utilizar o método síncrono a seguir descrito.

3.4.2 Método Síncrono

Apesar deste método dar origem a programas mais extensos, possui a vantagem de garantir que as regras de evolução são sempre respeitadas.

Para ultrapassar os referidos estados de conflito, este método começa por determinar todas as transições disparáveis, guardando o estado de cada uma delas em flags auxiliares, M100...M199, pertencentes ao vector de registos em memória descrito no capítulo anterior, de forma a que,

Se Ti está disparável, M'i=1
Se Ti não está disparável, M'i=0

No que se refere à reactivação de etapas, as regras do GRAFCET definem que a activação tem prioridade sobre a desactivação, deste modo, é suficiente que a activação das etapas de saída, das transições disparáveis, sujam após as instruções de desactivação das etapas de entrada, conforme se descreve no algoritmo de implementação a seguir representado.

O algoritmo de implementação do método síncrono:

1º - Determinação do estado do sistema

- Para cada transição, Ti, do GRAFCET:

  Se Ti está disparável

  então M'i = 1

  senão M'i = 0

- Para cada transição, Ti, do GRAFCET:

  Se M'i = 1

  então desactivar etapas de entrada Ti

- Para cada transição, Ti, do GRAFCET:

  Se M'i = 1

  então activar etapas de entrada Ti

2º - Determinar o valor das saídas

- Para cada etapa E do GRAFCET

  Executar as acções associadas a E

Neste caso, considerando novamente o exemplo representado na figura 65, o programa resultante seria:

1º - Determinação das etapas disparáveis

```
LD M0
AND I1
ST M21

LD M0
AND I2
ST M22

LD M1
AND I3
ST M23

LD M3
AND M2
AND I4
ST M24
```

2º - Desactivação das etapas de entrada

```
LD M21
R M0

LD M22
R M0

LD M23
R M1

LD M24
R M3
```

3º - Activação das etapas de entrada

```
LD M21
S M3

LD M22
S M1

LD M23
S M2

LD M24
S M0
```

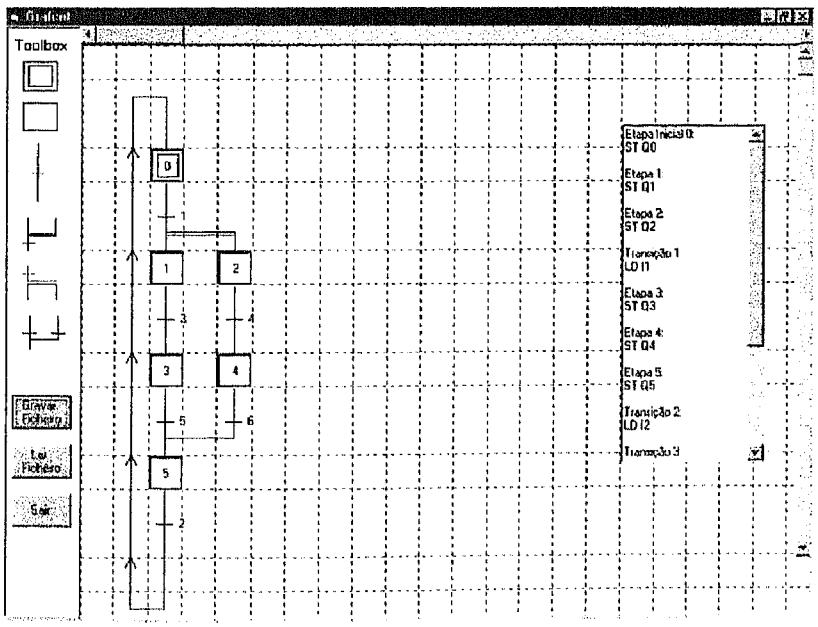
4º - Determinação das saídas

```
LD M0
ST Q0

LD M1
ST Q1

LD M2
ST Q2

LD M3
ST Q3
```

Evitando-se desta forma a ocorrência de erros.

Tomando em consideração o que foi referido e atendendo ao facto da necessidade de se solucionar quaisquer estados do sistema, nomeadamente os de conflito e reactivação de etapas, optou-se pela utilização do método síncrono para a geração do programa para o SIMAP.

Apesar do programa para o SIMAP, resultar da implementação directa do algoritmo do referido método, existem ainda algumas considerações a abordar. Consideremos o modelo GRAFCET representado na figura seguinte.



Através do qual se gera o respectivo programa para o SIMAP, a seguir representado.

Programa para o SIMAP

1º Determinação das etapas iniciais.	3º Determinação das etapas disparáveis.	4º Desactivação das etapas de entrada.	5º Activação das etapas de entrada.	6º Determinação do estado das saídas.
Inicio P0 LD S0 S M0 R M1 R M2 R M3 R M4 R M5 R S0	LD M0 AND I1 ST M100 ST M101 LD M1 AND I3 ST M102 LD M2 AND I4 ST M103 LD M3 AND I5 ST M104 LD M4 AND I6 ST M105 LD M5 AND I2 ST M106	LD M100 R M0 LD M102 R M1 LD M103 R M2 LD M104 R M3 LD M105 R M4 LD M106 R M5	LD M100 S M1 LD M101 S M2 LD M102 S M3 LD M103 S M4 LD M104 S M5 LD M105 S M5 LD M106 S M0	L0: LD M0 JMPCN L1 ST Q0 L1: LD M1 JMPCN L2 ST Q1 L2: LD M2 JMPCN L3 ST Q2 L3: LD M3 JMPCN L4 ST Q3 L4: LD M4 JMPCN L5 ST Q4 L5: LD M5 JMPCN L99 ST Q5 L99: Fim P0
2º Tratamento de acções memorizadas. LDN F0 R Q0 LDN F1 R Q1 LDN F2 R Q2 LDN F3 R Q3 LDN F4 R Q4 LDN F5 R Q5 LDN F6 R Q6 LDN F7 R Q7				

Podemos verificar que antes da aplicação, propriamente dita, do algoritmo referente ao método síncrono (3º - Determinação das etapas disparáveis), existem duas etapas, a primeira referente à "Determinação das etapas iniciais", a segunda

referente ao "Tratamento de acções memorizadas". Analisemos de forma particular a necessidade de cada uma delas.

1º Determinação das etapas iniciais

```
LD S0
S M0
R M1
R M2
R M3
R M4
R M5
R S0
```

Esta etapa permite que no primeiro ciclo de execução do programa, identificado pelo teste do bit de sistema S0, sejam definidas quais as etapas iniciais do sistema. Para o efeito atribui-se o valor lógico "1" às flags auxiliares representativas das etapas iniciais e o valor lógico "0" às restantes etapas. Estabelecendo-se deste modo o ponto de partida do sistema.

2º Processamento de acções memorizadas

```
LDN F0
R Q0
LDN F1
R Q1
LDN F2
R Q2
LDN F3
R Q3
LDN F4
R Q4
LDN F5
R Q5
LDN F6
R Q6
LDN F7
R Q7
```

Esta parte do programa é utilizada para garantir que são respeitadas as regras do GRAFCET, referentes às acções contínuas e acções memorizadas, uma vez que o código das acções pertencentes a uma dada etapa só é processado se esta estiver activa, conforme se pode verificar na parte do programa referente a "Determinação do estado das saídas". Para o efeito são utilizadas um conjunto de flags auxiliares, F0...F7, referentes respectivamente às saídas, Q0...Q7, pertencentes ao vector de registos em memória descrito no capítulo anterior. Estas flags são colocadas a valor lógico "1" sempre que for realizado o "set" da respectiva saída e colocadas a valor lógico "0" quando se efectua o "reset".

Desta forma, no novo ciclo do programa, enquanto não se realize o "reset" das saídas às quais anteriormente se tenha feito o "set", estas mantêm-se activas. As saídas pertencentes a acções contínuas, começam por ser colocadas a "0" e voltarão a "1" se a respectiva etapa se mantiver a activa. Note-se que estas variações de valor lógico durante a execução do ciclo do programa, são realizadas a nível das flags internas. Apenas no fim do ciclo é que haverá repercussões nas saídas.
Consideremos o exemplo o modelo representado na figura 68.

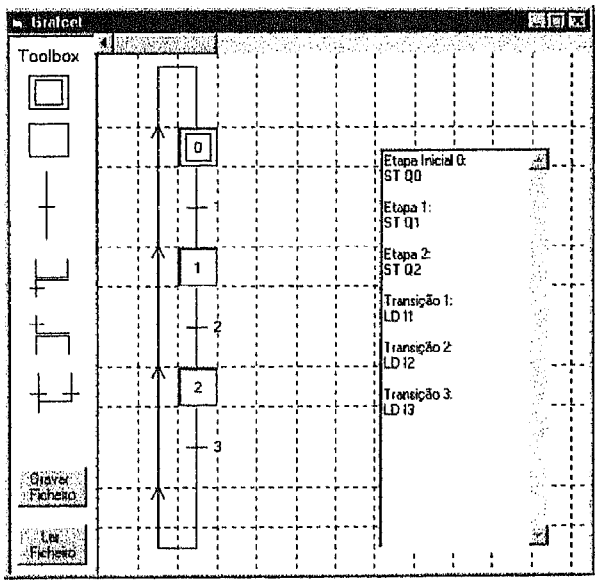


Figura 68 - Modelo GRAFCET

A partir do qual se obtém o programa a seguir representado:

1º	3º	4º	5º
Inicio P0	LD M0	LD M100	L0:
LD S0	AND I1	R M0	LD M0
S M0	ST M100		JMPCN L1
R M1		LD M101	ST Q0
R M2	LD M1		
R S0	AND I2	R M1	L1:
	ST M101		LD M1
2º		LD M102	JMPCN L2
LDN F0	LD M2	R M2	ST Q1
R Q0	AND I3		
LDN F1	ST M102	LD M100	L2:
R Q1		S M1	LD M2
LDN F2			JMPCN L99
R Q2		LD M101	ST Q2
LDN F3		S M2	
R Q3			L99:
LDN F4		LD M102	
R Q4		S M0	Fim P0
LDN F5			
R Q5			
LDN F6			
R Q6			
LDN F7			
R Q7			

Admitindo que inicialmente o estado do sistema é, M0=1, M1=0 e M2=0, teremos **Q0=1, Q1=0** e Q2=0. Assim que a "transição 1" for transposta e for iniciado um novo ciclo, de acordo com o bloco do programa referente a "tratamento de acções memorizadas" as flags internas referentes às saídas Q0... Q3, são colocadas a valor lógico "0". Sendo agora o estado do sistema, M0=0, M1=1 e M2=0, assim que as saídas forem processadas, e uma vez que apenas são processadas aquelas referentes a etapas activas, teremos, **Q0=0, Q1=1** e Q2=0.

Considerando agora as mesmas condições iniciais, mas omitido a parte do programa referente a "Processamento de acções memorizadas", acontecerá que, após a "transição 1" ser transposta e se dar início a um novo ciclo do programa as flags internas referentes a cada uma das saídas manter-se-ão em **Q0=1, Q2=0** e Q3=0. No entanto, uma vez que o estado do sistema passa a ser M0=0, M1=1 e M2=0, assim que

as saídas forem processadas, e uma vez que apenas é processado o código referente às etapas activas, obteremos **Q0=1, Q1=1** e Q2=0. Note-se que apesar da "etapa 0" estar inactiva, uma vez que o código referente às suas acções não foi processado, a respectiva saída, Q0, mantém o nível lógico "1", o que está errado uma vez que se trata de uma acção contínua.

Se se alterasse o programa, de forma a que à etapa M0 correspondesse uma acção memorizada tipo "S Q0" em vez da anterior "ST Q0", quando esta acção fosse processada a respectiva flag auxiliar "F0" seria colocada a nível lógico "1". No início do ciclo seguinte assim que fossem "processadas as acções memorizadas" uma vez que F0=1, Q0 manter-se-ia a nível lógico "1". Neste caso, o estado seguinte das saídas do sistema seria Q0=1, Q1=1 e Q2=0, tal como pretendido.

Capítulo 4

Simuladores de Sistemas Físicos

4.1 Apresentação

O simulador de sistema físico, constitui o elo final da cadeia de aplicações, integradas na ferramenta de apoio ao desenvolvimento de sistemas de automação industrial, que tem por objectivo, permitir ao utilizador simular o controlo de um automatismo real.

A simulação do sistema físico é obtido através da animação do conjunto de objectos constituintes do simulador, cujo comportamento depende da variável que lhe é associada.

Neste módulo foram desenvolvidos um conjunto de simuladores representativos de sistemas reais susceptíveis de serem controlados por um autómato programável. Pretende-se deste modo, dotar a ferramenta desenvolvida com um conjunto de módulos de software representativos de sistemas automatizados, a serem utilizados pelo aluno para o teste e validação dos seus programas, previamente editados em lista de instruções ou em GRAFCET e em execução no Simap.

Efectuemos então a apresentação e análise funcional de cada um dos simuladores desenvolvidos.

4.2 Análise funcional

4.2.1 Parque de Estacionamento

No parque de estacionamento, cuja interface com o utilizador se representa na figura 69, podem-se adoptar um enorme variedade de soluções de controlo que estão dependentes do tipo de funcionamento desejado. Independentemente da opção adoptada, esta passa sempre pelo controlo do conjunto de mecanismos constituintes do sistema. No caso que se apresenta, temos que efectuar o controlo de duas cancelas, para a entrada e saída de veículos, o semáforo para indicação de parque cheio, o movimento dos veículos e a leitura de seis detectores de presença.

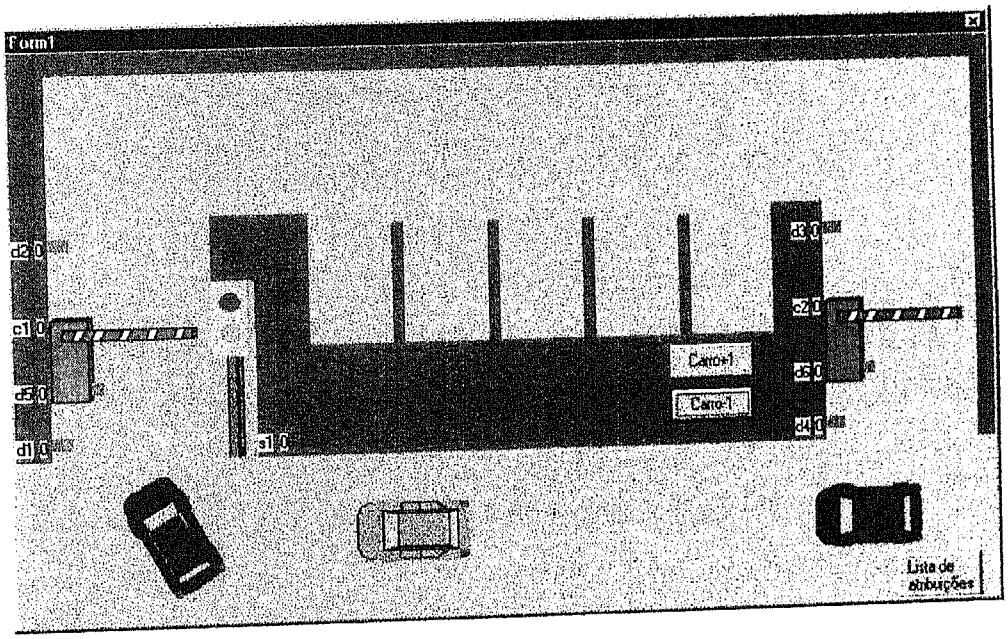


Figura 69 - Parque de estacionamento: Interface com o utilizador

Para que o sistema se torne o mais semelhante possível a um sistema real, foi necessário definir a animação de cada um dos objectos constituintes e definir para cada um deles a variável que irá permitir efectuar o seu controlo.

Comecemos por considerar o movimento dos veículos. A opção adoptada permite que o utilizador efectue o controlo do movimento de cada um dos veículos

do sistema. Foram definidos tantos veículos quanto o numero de lugares existentes no parque de estacionamento, seis. Outra das opções seria a de deslocar os veículos de forma automática sem a intervenção do utilizador, no entanto o movimento controlado pelo utilizador tem a vantagem de possibilitar efectuar uma análise mais cuidada do comportamento do sistema, nomeadamente a nível da actuação dos detectores. Para os veículos não foi necessário definir nenhuma variável de controlo, uma vez que estes não vão interagir directamente com o Simap. Desta forma o utilizador poderá deslocar de forma independente cada um dos veículos, através das cancelas e zonas estacionamento.

No desenho do parque de estacionamento foram definidas zonas exclusivas para a entrada e saída de veículos. Em cada uma destas zonas foi colocada uma cancela, cuja abertura é controlada respectivamente pelas variáveis lógicas "C1" e "C2". Quando colocadas a valor lógico "1" as cancelas abrem, quando colocadas a valor lógico "0" fecham. Em cada uma desta zonas foram também colocados três sensores de presença, às quais se associaram as variáveis "d1", "d2" e "d5" para a entrada e "d3", "d4" e "d6" para a saída. Sempre que um veículo estiver nas mediações de um destes sensores a respectiva variável é colocada a valor lógico "1", caso contrário manter-se-á com o valor lógico "0". Sempre que um detector for actuado é também dada uma indicação visual ao utilizador através da mudança de cor do sensor, conforme se representa na figura 70.

A utilização de sensor "d1" tem por objectivo detectar a chegada de um veículo, de modo a que se proceda à abertura da cancela "c1". O sensor "d5", detecta a presença do veículo a passar pela cancela, de modo a que esta não se feche aquando da passagem do veículo. O sensor "d2", permite detectar que o veículo efectivou a sua entrada no parque, pelo que se pode proceder ao fecho da cancela. Os detectores colocados na saída terão função semelhante.

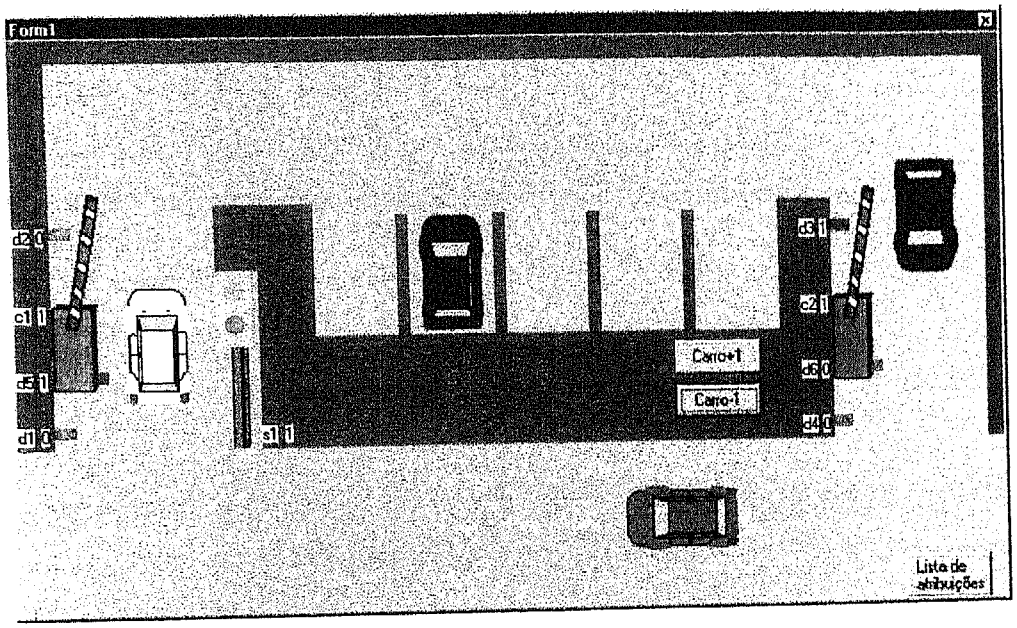


Figura 70 - Parque de estacionamento: Análise funcional

Ao semáforo foi associada a variável "S1", que quando colocada a valor lógico "1" altera o semáforo para verde (parque livre), quando colocada a valor lógico "0" altera o semáforo para vermelho (parque ocupado).

Será através da ligação lógica deste conjunto de variáveis ao Simap, que é efectuado o controlo do parque de estacionamento. O Simap efectua a leitura de cada um dos detectores e accionará os actuadores do sistema de acordo com o programa que estiver em execução.

As variáveis que o programa terá a considerar são definidas através de uma lista de atribuições disponível no simulador do sistema físico, que deverão ser consultadas pelo utilizador aquando da elaboração do programa, de modo a que possa realizar uma correspondência adequada entre entradas/saídas do Simap e os detectores/actuadores do simulador.

4.2.2 Célula de fabrico

Na célula de fabrico, cuja interface com o utilizador se representa na figura 71, pretende-se efectuar o controlo do robot, do tapete de entrada, do tapete de saída, das máquinas e a leitura de um conjunto de sensores para a detecção do tipo de peça, posição do robot.

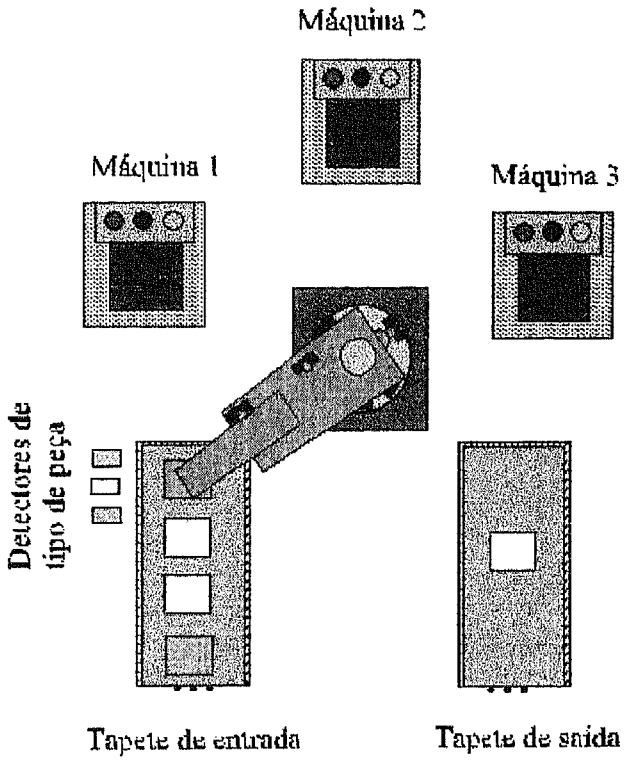


Figura 71 - Célula de fabrico: Interface com o utilizador

Admitindo que o tapete de entrada fornece 3 tipos de peças distintas, uma das soluções de controlo passíveis de serem implementadas, seria a seguinte. Cada um dos detectores de tipo de peça permite indicar ao Simap, em cada instante de tempo, o tipo de peça que chega. Assim que for detectada nova peça, o Simap desloca o robot até ao tapete de entrada. Em seguida, o robot pega na peça, recolhe o braço e, em função do detector de tipo de peça actuado, desloca-se à máquina 1, 2 ou 3, conforme se representa na figura 72.

Aí chegado, estica o braço, larga a peça, em seguida recolhe o braço e a maquina é colocada em funcionamento. Assim que a peça for maquinada, o robot

retira a peça da maquina e coloca-a no tapete de saída. Quando surgir nova peça no tapete de entrada, o processo repete-se.

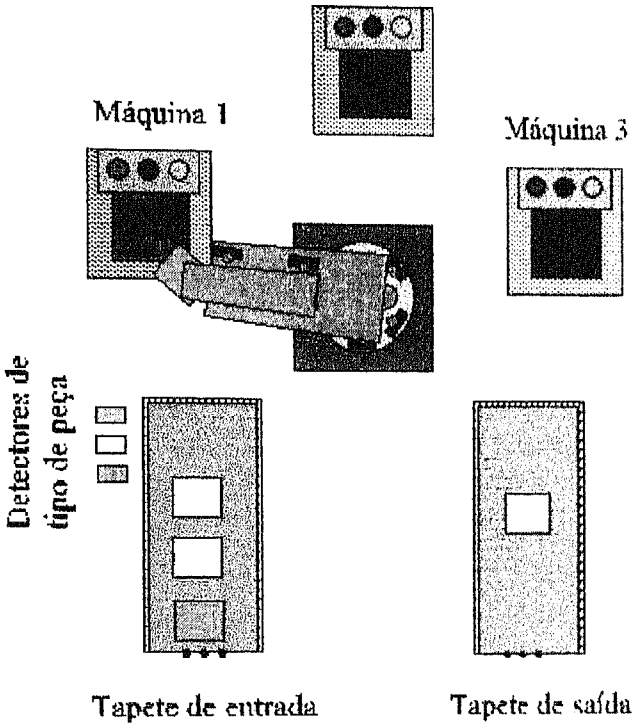


Figura 72 - Célula de fabrico: Análise funcional

Note-se que, se antes de uma peça ser maquinada, surgir uma nova peça no tapete de entrada, o robot deverá encaminhá-la para a respectiva máquina, a não ser que esta esteja ocupada.

Como se pode constatar, teremos que definir uma grande variedade de variáveis de controlo para cada um dos objectos do sistema, nomeadamente para, o controlo do tapete de entrada, o controlo do tapete de saída, o controlo do movimento rotativo do robot, o controlo do movimento do braço do robot, o controlo do "gripper" do robot, o controlo ON/OFF das máquinas 1, 2 e 3, e um conjunto de variáveis para a leitura dos sensores de tipo de peça, sensores de posição de rotação do robot e sensores de posição do braço do robot. Naturalmente que, tal como aconteceu para o simulador anterior, este conjunto de variáveis do sistema irá

interagir com as entradas/saídas do Simap, de modo a efectuar-se o controlo adequado de todo o sistema.

4.3 Descrição da implementação

A implementação dos simuladores de processos físicos prende-se essencialmente com dois factores, a implementação dos objectos que constituem o sistema e a execução, propriamente dita, da aplicação simulador.
A implementação dos objectos, é obtida à custa da criação de um conjunto de classes de objectos, para os quais se definem as propriedades e métodos que regem o comportamento da classe, e a partir dos quais derivam todos os objectos do sistema.

Para a execução da aplicação temos a considerar duas abordagens, cuja apresentação será realizada considerando a título de exemplo, o comando de uma das cancelas representadas no simulador "Parque de estacionamento" já descrito em temos funcionais.
Admita-se que o sinal de saída Q, comanda o estado da cancela, de forma a que: Q=0, cancela fechada, Q=1, cancela aberta.

Na primeira abordagem, consoante o valor Q=0 ou Q=1 a cancela gráfica está respectivamente na posição aberta ou fechada, conforme se representa na figura 73, sendo a transição realizada de forma "instantânea".

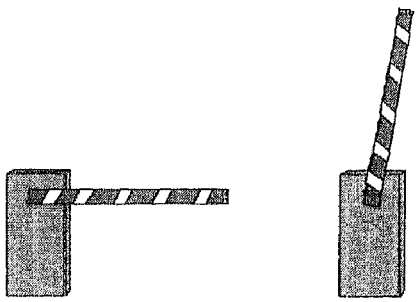


Figura 73 - Execução da aplicação simulador: Abertura da cancela

Neste caso, o objecto associado tem apenas dois estados, conforme se representa na figura 74.

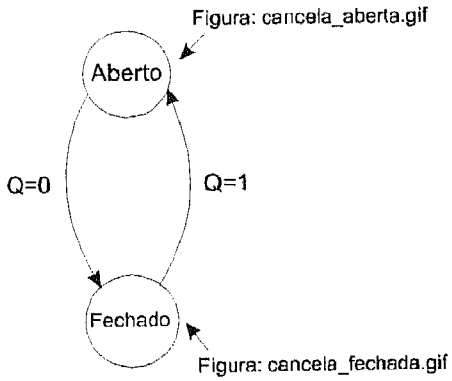


Figura 74 - Execução da aplicação simulador: Diagrama de estados da 1ª abordagem

Na segunda abordagem, a transição 0-1 no sinal de saída do AP dá origem a um movimento da cancela de tempo não nulo, conforme se representa na figura 75.

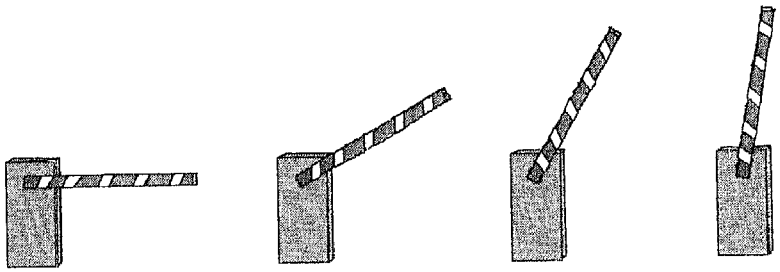


Figura 75 - Execução da aplicação simulador: Abertura da cancela

Agora, o objecto tem o diagrama de estados, representado na figura 76.

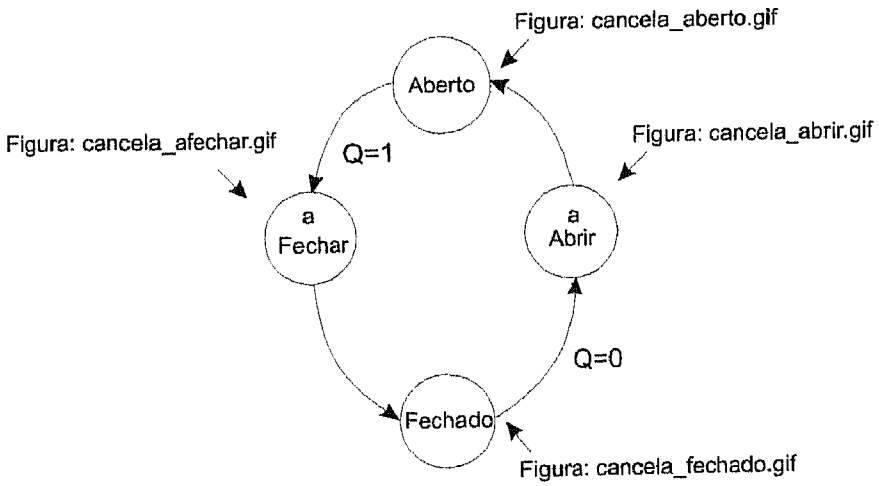


Figura 76 - Execução da aplicação: Diagrama de estados da 2ª abordagem

A primeira a bordagem está relacionada com a execução de um conjunto de acções como resposta a um ou mais conjuntos de eventos do sistema. Neste caso, num dado instante de tempo, é executada, de início a fim, apenas uma acção, no caso que se apresenta quando surgisse o evento referente à passagem de Q=1 para 0, seria executado apenas o método referente ao fecho da cancela, só após a execução deste método é que eventualmente, se desencadearia a execução de outro método ou se esperaria por novo evento.

A segunda abordagem está associada à execução cíclica do conjunto de acções ou programas que regem o funcionamento de todo o sistema. Deste modo, considerando o exemplo anterior, durante o movimento da cancela podem ocorrer outros eventos a actuar sobre outros objectos. Neste caso, os métodos associados a cada objecto do sistema são executados de forma cíclica, e têm por função, não a execução início-fim de um acção, mas sim a verificação e validação do estado seguinte do objecto.

A escolha da abordagem mais adequada depende do tipo de simulador utilizado. Consideremos a título de exemplo a implementação referente ao parque de estacionamento a seguir descrito.

4.3.1 Descrição da implementação do "Parque de Estacionamento"

Para este caso em concreto, são definidas quatro classes de objectos, a classe veículos, a classe semáforos, a classe cancelas e a classe sensores. Para cada uma destas classes são definidas um conjunto de propriedades e métodos a partir das quais se regem todos os objectos pertencentes à classe.

Para a elaboração destas propriedades e métodos, foi necessário definir previamente o comportamento pretendido para cada um dos objectos da classe. Comecemos por analisar a classe veículos.

Classe veículos

Ora, o comportamento do veículo prende-se apenas com o seu movimento, desta forma criou-se o método "carro.exe" que permite que sempre que o utilizador pressione o botão do lado direito do rato, o veículo seleccionado, se desloque ou não, dependendo da posição anterior, para a posição actual do cursor. O código associado a este método efectua a leitura da posição actual do veículo seleccionado, e através de um mecanismo "Select Case" determina se a nova posição pretendida é válida, em caso afirmativo, o veículo é deslocado para as novas coordenadas e a respectiva imagem visualizada. A restrição do movimento do veículo é apenas efectuada de modo a que sequência de deslocamentos seja realizada de forma ordenada.

Para a visualização de cada veículo foram definidas oito tipos de figuras guardadas em disco, conforme se representa na figura 77.

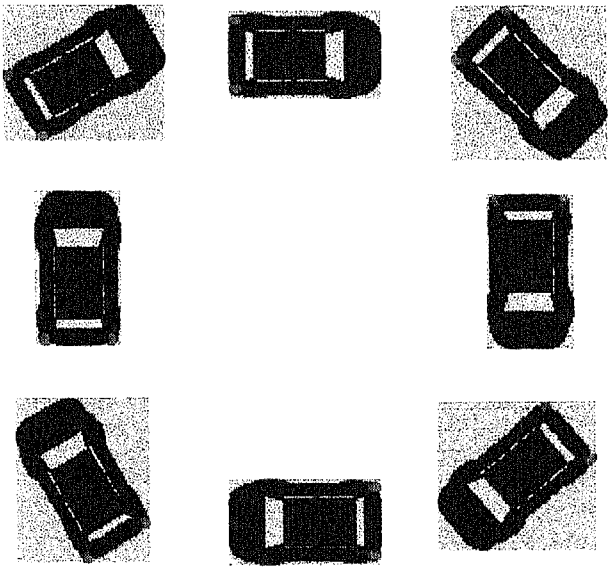


Figura 77 - Simulador "Parque de estacionamento": Veículos

A razão pela qual se necessita deste conjunto de figuras, deve-se ao facto do Visual Basic não possuir instruções de rotação. Deste modo, sempre que for necessário visualizar a posições do veículo como aquelas representadas na figura 78, é necessário a carregar do disco novas imagens.

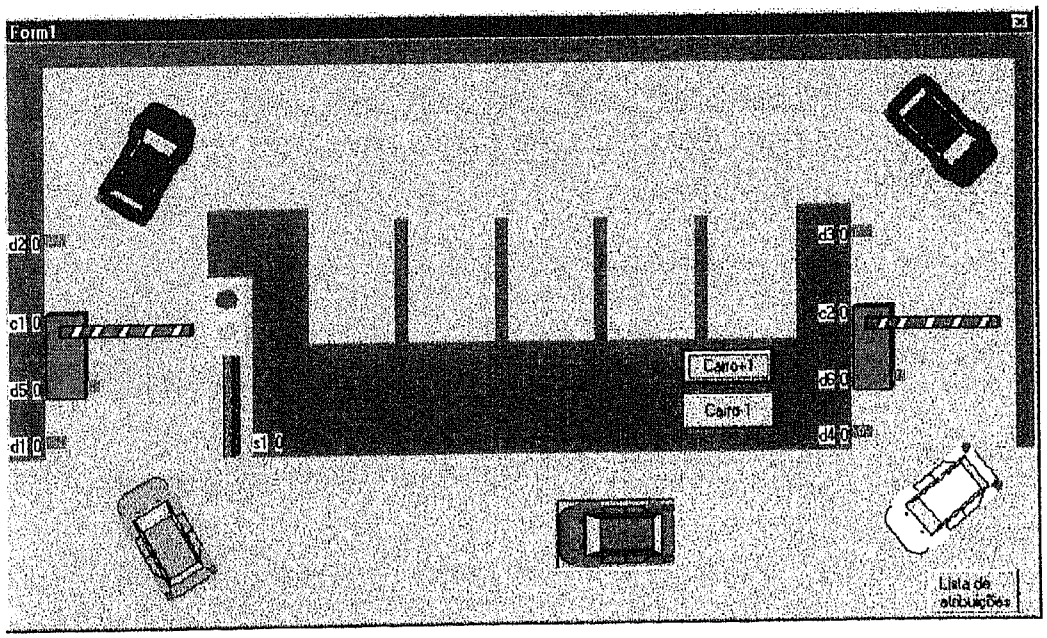


Figura 78 – Parque de estacionamento

Classe semáforos

O semáforo terá apenas que mudar de cor de acordo com o estado da sua variável. Nesta classe foi definida a propriedade "semaforos.**sinal**" e o método "semaforos.**exe**". A O método verifica apenas o estado da propriedade, se esta estiver a valor lógico "1", o semáforo é colocado em verde, caso contrário é colocado a vermelho. O valor da propriedade é imposto pela variável que lhe está associada. A colocação do semáforo em verde e vermelho, consiste na leitura de duas figuras distintas armazenadas em disco, conforme se representa na figura 79.

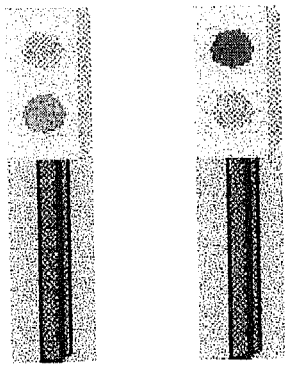


Figura 79 Semáforos

Classe cancelas

A cancela terá que efectuar o movimento de abertura e fecho. Para o efeito foi considerada a propriedade "cancelas.**abrir**" e o método "cancelas.**exe**". O método tem por objectivo verificar o estado da propriedade, se "cancelas.abrir" assumir o valor lógico "1" a cancela abre, caso contrário fecha. O valor da propriedade é imposto pela variável que lhe está associada . O movimento de abertura e fecho, é obtido à custa da leitura de uma sequência de imagens pré-definidas em disco, conforme se representa na figura 80.

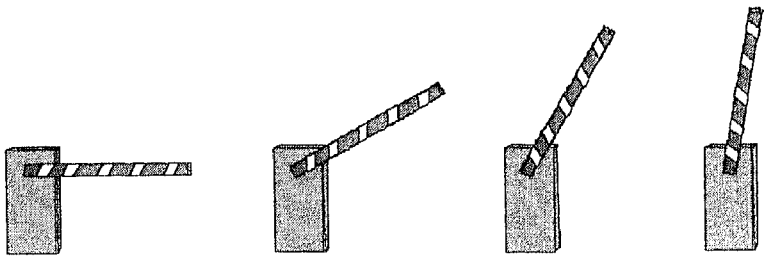


Figura 80 Cancelas

Classe sensores

Cada um dos sensores do sistema, terá que detectar a presença de veículo na sua proximidade e fornecer essa indicação ao Simap. Nesta classe foram definidas as propriedades "sensores.**estado**", "sensores.x", "sensores.y" e o método "sensores.**exe**". A propriedade "x" e "y" define a posição do veículo no desenho. O método "sensores.exe", verifica a existência de veículo na proximidade das coordenadas do sensor, e em caso afirmativo coloca a propriedade "sensores.estado" a valor lógico "1". A variável de ligação ao Simap, correspondente a cada um dos sensores, assume o valor desta propriedade.

Para a execução da aplicação, é utilizada a primeira abordagem, descrita no item anterior, ou seja, a execução de cada um dos métodos dos objectos, depende da existência de um evento, que neste caso se refere à actualização da nova posição do veículo definida pelo utilizador. Assim que este evento surgir desencadeia a execução de cada um dos métodos(.exe) dos objectos definidos. Note-se que a execução de cada método é realizado na perspectiva início-fim, apenas se dará o início à execução de um novo método quando o anterior tiver finalizado a sua execução.

Analisemos os blocos de código de programa, representados na tabela 36 e 37. A primeira referente às acções despoletadas por o evento, resultante da actualização da posição do veículo, a segunda referente ao evento resultante da actualização das saídas do Simap.

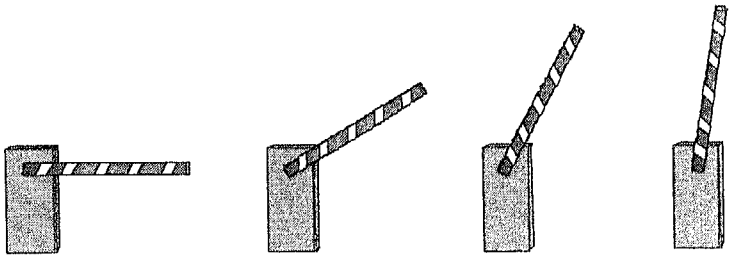


Figura 80 Cancelas

Classe sensores

Cada um dos sensores do sistema, terá que detectar a presença de veículo na sua proximidade e fornecer essa indicação ao Simap. Nesta classe foram definidas as propriedades "sensores.**estado**", "sensores.x", "sensores.y" e o método "sensores.**exe**". A propriedade "x" e "y" define a posição do veículo no desenho. O método "sensores.exe", verifica a existência de veículo na proximidade das coordenadas do sensor, e em caso afirmativo coloca a propriedade "sensores.estado" a valor lógico "1". A variável de ligação ao Simap, correspondente a cada um dos sensores, assume o valor desta propriedade.

Para a execução da aplicação, é utilizada a primeira abordagem, descrita no item anterior, ou seja, a execução de cada um dos métodos dos objectos, depende da existência de um evento, que neste caso se refere à actualização da nova posição do veículo definida pelo utilizador. Assim que este evento surgir desencadeia a execução de cada um dos métodos(.exe) dos objectos definidos. Note-se que a execução de cada método é realizado na perspectiva início-fim, apenas se dará o início à execução de um novo método quando o anterior tiver finalizado a sua execução.

Analisemos os blocos de código de programa, representados na tabela 36 e 37. A primeira referente às acções despoletadas por o evento, resultante da actualização da posição do veículo, a segunda referente ao evento resultante da actualização das saídas do Simap.

Comecemos pela tabela 36, neste caso verificamos que, quando surgir o evento, são executados, de forma consecutiva e de uma só vez, os métodos carroX.exe referentes a cada veículo, de modo a que o veículo seleccionado se desloque para a nova posição. Em seguida, são executados os métodos sensor_X.exe, nos quais cada um dos detectores de presença verifica se existe algum veículo na sua proximidade. Finalmente, são actualizadas as entradas do Simap, com a informação anterior recolhida por cada detector, En(X) = sensor_X.estado.

Tabela 36 - Parque de estacionamento: Execução da aplicação

```
Private Sub Form_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
Carro_1.Exe X, Y                    ' execução dos métodos referentes a cada veículo
Carro_2.Exe X, Y
carro_3.Exe X, Y
carro_4.Exe X, Y
carro_5.Exe X, Y

sensor_1.Exe                        ' execução dos métodos referentes a cada veículo
sensor_2.Exe
sensor_3.Exe
sensor_4.Exe
sensor_5.Exe
sensor_6.Exe

En(0).Caption = sensor_1.estado     ' Actualização das variáveis de entrada do Simap
En(1).Caption = sensor_2.estado     ' leitura dod detectores de presença do sistema
En(2).Caption = sensor_3.estado
En(3).Caption = sensor_4.estado
En(4).Caption = sensor_5.estado
En(5).Caption = sensor_6.estado

End Sub
```

Na tabela 37, o evento em causa, é a notificação à aplicação simulador, da existência de uma actualização das saídas do Simap, para a actuação de uma cancela. Este evento, despoleta um mecanismo do tipo "Select Case", que verifica qual a saída em causa, em seguida actualiza a variável de controlo da cancela, cancela_X.abrir =

Val(Sd(0).Caption), e executa o respectivo método, cancela_X.exe, que tem como função actualizar a imagem referente à nova posição de aberto ou fechado.

Tabela 37 - Parque de estacionamento: Notificação de actualização de variáveis

```
Private Sub Sd_LinkNotify(Index As Integer)
  Select Case Index
    Case 0
      Sd(0).LinkRequest                    ' actualização das variáveis de controlo
                                           ' da cancela 1
      cancela_1.abrir = Val(Sd(0).Caption) ' proveniente da variável de ligação à saída do
                                           ' Simap.
      cancela_1.Exec

    Case 1
      Sd(1).LinkRequest                    ' controlo da cancela 2
      cancela_2.abrir = Val(Sd(1).Caption)
      cancela_2.Exec

    Case 2
      Sd(2).LinkRequest                    ' controlo do semáforo
      semaforo.sinal = Val(Sd(2).Caption)
      semaforo.Exec

    End Select
End Sub
```

Na execução da aplicação também poderia ser utilizada a segunda abordagem, neste caso, os métodos "exe" de cada um dos objectos são realizados de forma cíclica. A ocorrência do evento referente à actualização do veículo apenas origina a sua mudança de posição e não despoleta quaisquer outros métodos, uma vez que estes já se encontram em execução.

Este tipo de simulador, permite que a sua execução seja realizada utilizando qualquer uma das abordagens descritas anteriormente. No entanto, se fosse utilizado um simulador no qual houvesse a necessidade de um movimento continuo, por exemplo, o de um comboio, neste caso não poderíamos esperar pela ocorrência de um evento para efectuar a referida deslocação, e por isso teríamos que optar, obrigatoriamente, pela segunda bordagem.

Capítulo 5

Conclusões e Perspectivas de desenvolvimento

O trabalho realizado no âmbito da presente dissertação, teve como objectivo a realização de um conjunto integrado de ferramentas, constituído por:

- Editor de GRAFCET
- Simulador de autómato programável
- Simuladores de sistemas físicos

A utilização deste conjunto de ferramentas, permitiu desenvolver uma aplicação que possibilita a simulação em PC, de todas as etapas de desenvolvimento de um sistema automatizado, desde a edição do programa em lista de instruções ou em GRAFCET até à simulação do controlo do sistema físico.

No desenvolvimento do simulador do autómato programável, foi definida uma interface que permite, além da execução do programa, acompanhar a sua evolução por visualização directa do conjunto de entradas, saídas, flags auxiliares, contadores e temporizadores, disponíveis na interface ou através de funcionalidades adicionais tais como o traçado de diagramas temporais on-line e off-line.

Na implementação do simulador, foi necessário definir o set de instruções a utilizar. Tratando-se de um sistema didáctico, a ser utilizado pela Faculdade de Engenharia, optou-se por adoptar a mesma linguagem utilizada pelos autómatos em uso na FEUP, linguagem "PL7 da Telemecanique", de forma a que o aluno, numa fase de aprendizagem, não tenha necessidade de aprender dois tipos de linguagem distintos, uma para a programação dos autómatos e outra para a utilização do simulador.

Para que simulador do autómato programável, execute o programa pretendido de forma mais eficiente, foi criado um bloco de compilação que converte o código fonte, proveniente dos editores de texto ou GRAFCET, em código executável, de estrutura mais simples e consequentemente de execução mais rápida.

Outro dos aspectos a tomar em considerar, foi a filosofia de funcionamento do autómato programável, nomeadamente quanto à execução cíclica do programa, tendo sido para o efeito utilizado o ciclo síncrono para entradas e saídas.

O desenvolvimento do editor de GRAFCET, levou à necessidade de definição da interface com o utilizador, da simbologia a utilizar, da estrutura interna de dados, em memória inerente edição gráfica, e em disco para se guardar o modelo realizado e possibilitar a sua utilização em futuras edições.

Outro dos aspectos a considerar, foi a necessidade de interpretação do grafo para a geração automática do programa para o simulador do autómato programável. Houve a considerar dois métodos, o assíncrono e o síncrono. Uma vez que no primeiro método podem ocorrer erros perante situações de conflito e de reactivação de etapas, foi adoptado o método síncrono de modo a salvaguardada a ocorrência de qualquer situação.

Por fim, foram desenvolvidos simuladores de sistemas físicos, constituídos por um conjunto de objectos passíveis de animação, no sentido de tornar a simulação o mais realista possível. Para a implementação dos simuladores, foram criadas classes de objectos, caracterizadas por um conjunto de propriedades e métodos, dos quais derivam todos os objectos do simulador

O objectivo final, é efectuar a interacção desta aplicação, com o simulador do autómato programável, de forma a efectuar o teste e validação do programa de controlo realizado. A interacção entre aplicações é realizada via DDE e consiste na ligação lógica entre as variáveis de controlo, associadas aos detectores/actuadores do simulador do sistema físico e as entradas/saídas do simulador do autómato programável. Em termos funcionais tudo se passa como se fosse realizada uma ligação física entre os dois sistemas.

Em termos de perspectivas de desenvolvimento e aplicação, acredita-se que estas ferramentas podem ser um auxilio valioso no apoio ao ensino de sistemas de

automação industrial, no entanto, falta ainda testa-las na prática de modo a puder-se constatar a necessidade de futuros desenvolvimentos. De qualquer modo, desde já, é possível identificar alguns:

- Implementação das aplicações numa ferramenta mais eficiente, possivelmente, Visual C++.
- Aumentar a cobertura do "set" de instruções
- Aumentar a cobertura das funcionalidades do GRAFCET, nomeadamente quanto à possibilidade de implementação macro etapas e transmissões.
- Desenvolvimento de novos simuladores e criação de uma biblioteca de classes, de onde poderiam ser criados os objectos utilizados nos novos simuladores.
- Comunicação com o autómato programável, de modo a executar o programa no próprio autómato ou simular a sua execução no PC. Seria quase como uma integração desta ferramenta com o software de desenvolvimento do autómato programável.

Bibliografia

1 - Using VisualBasic 5.0

M. McKelvy, R. Martinsem, J. Webb

QUE Corporation 1995

2 - Doing Objects in Microsoft Visual Basic 5.0

Deborah Kurata

Ziff-Davis Press, 1997

3 - From Grafcet to Petri nets

R. David. H. Alla

Prentice-Hall, 1994

4 - Compreender, matrîser e appliquer le Grafcet

M. Blanchard

Cepadues Éditions, 1982

5 - Le Grfcet: de nouveaux concepts

Grupe Équipment de Production Automatisé

Cepadues Éditions, 1985

6 - Les automatismes programmables

D. Bouteille

Cepadues Éditions, 1987

7 - Computer automated manufacturing

J. Powers JR.

McGraww-Hill Book Company, 1987

8 - Introduction to programmable logic controllers
G. Dunning
Delmar Publishers, 1998

9 - Sistemas de Concepção de Automatismos em GRAFCET
Raul Oliveira, dissertação de mestrado
FEUP, 1991

10 - Apontamentos da disciplina Automação Industrial
J. Faria
FEUP, 1998