

Implementação da estruturação automática de programas PLC de controlo de transportadores seguindo a metodologia do grupo Körber na Consoveyo S.A.

Paulo André Vaz Lopes

Dissertação de Mestrado

Orientador na FEUP: Professor Doutor Germano Correia dos Santos Veiga

Orientador na Consoveyo S.A.: Engenheiro António Lopes



Mestrado Integrado em Engenharia Mecânica

Setembro 2020

*“Success is not final;
failure is not fatal:
It is the courage to continue that counts”*

Winston S. Churchill

Resumo

O objetivo desta dissertação de mestrado, elaborada em conjunto com o departamento de Automação da Consoveyo S.A., foi estudar uma nova metodologia a ser adotada na empresa para a programação de sistemas periféricos, que se baseia na geração automática de código para controladores lógicos programáveis (PLC) utilizados no controlo destes equipamentos.

Com a implementação desta nova metodologia na Consoveyo S.A., a empresa pretende diminuir o tempo de programação manual, reduzir a duração da integração destes sistemas em obra e melhorar a produtividade e rendimento da empresa como um todo. Estes objetivos são alcançáveis uma vez que, atualmente, a empresa não possui um *standard* bem definido para este processo.

No estudo desta metodologia foram utilizadas ferramentas de programação de projetos de automação, como o TIA Portal e o software desenvolvido pelo grupo Körber, Engineering Editor. Foi também utilizado o Visio como interface com o utilizador para a definição do sistema de periféricos a ser colocado no armazém automático.

Na parte prática desta dissertação foi desenvolvida uma aplicação .NET, inspirada na metodologia estudada, que gera automaticamente código PLC para um cenário virtual de simulação 3D, também gerado automaticamente pela aplicação.

Foi usado o TIA Portal como ferramenta principal para programação de PLCs, e exploradas as funcionalidades do software TIA Portal Openness com o propósito de permitir a geração automática de código. Para a reprodução do sistema periférico num simulador, foi utilizado o software Factory IO.

A edição de documentos XML foi um processo crucial durante toda a elaboração da aplicação, uma vez que são este tipo de ficheiros que permitem a geração dos dois componentes referidos.

O possível impacto desta aplicação num ambiente industrial passa pela existência de mais um ponto de verificação da programação através do teste da mesma num simulador. Atualmente a empresa não recorre a este género de programas pois seria necessário um elevado período de tempo para a construção manual de um armazém automático no simulador.

Abstract

The purpose of this master's dissertation, elaborated with the Automation department of Consoveyo S.A., was to study the new methodology to be implemented on the company to program peripheral systems, based on automatic code generation for programmable logic controllers (PLC) used for the control of this equipments.

With the implementation of this new methodology on Consoveyo S.A., the company wants to reduce the time of manual programming, shorten the integration time of these systems on-site and improve the productivity and efficiency of the company as a whole. These goals are reachable once, nowadays, the company does not have a well-established standard for this process.

During the study of this methodology were used automation project programming tools, such as TIA Portal and the software developed by the Körber group, the Engineering Editor. The software Visio was used as an interface with the user to create the peripheral system to be installed in the automated warehouse.

On the practical approach of this dissertation, it was developed a .NET app, inspired on the studied methodology, that automatically generates PLC code for a 3D virtual scenario, also automatically generated by the app.

TIA Portal was used as the main tool to program PLCs, and the functionalities of TIA Portal Openness were explored for allowing the automatic code generation. For the recreation of the peripheral system on the simulator, the software Factory IO was used.

The editing of XML documents was a crucial process throughout the development of the app since this type of file allows the generation of the two referred components.

The possible impact of this app on an industrial environment is the existence of another programming checkpoint by testing it on a simulator. Nowadays the company does not use this kind of software because it would consume a large amount of time with the manual construction of the system in the simulator.

Agradecimentos

Gostaria de demonstrar os meus profundos agradecimentos a todos que fizeram parte desta etapa importante na minha formação profissional, em particular:

Ao Professor Doutor Germano Manuel Correia dos Santos Veiga por todo o acompanhamento, partilha de conhecimento e ajuda prestada ao longo da elaboração desta dissertação.

Ao Engenheiro António Lopes por me ter dado a oportunidade de integrar a equipa de Automação da Consoveyo S.A. e pela disponibilidade demonstrada ao longo do estágio curricular.

À Consoveyo S.A. pela oportunidade de realizar este projeto em conjunto com a empresa. O contacto com a realidade industrial e empresarial durante o estágio curricular é algo que dou muito valor e que certamente será uma mais valia no futuro.

Aos meus pais, Paulo Lopes e Maria Vaz Alves, pela força dada durante esta etapa, por sempre me motivarem a dar tudo, pela confiança depositada e por me terem possibilitado as melhores condições durante todo o meu percurso académico.

Às minhas irmãs, Sofia e Carolina Lopes, pelo acompanhamento, constante incentivo e paciência ao longo destes 5 anos.

À minha namorada, Carolina Couto, pelo carinho, apoio incondicional, amizade, por estar sempre lá nos bons e maus momentos e por sempre me encorajar a dar meu melhor, especialmente durante a realização desta dissertação.

Ao meu grupo de amigos da faculdade, por todos os momentos partilhados durante estes 5 anos. Foi um percurso inesquecível, do qual vou levar muitas amizades e para sempre lembrar com muita saudade.

À minha avó, Carmo Alves, por sempre ter sido uma segunda mãe, por ter rezado sempre por mim e por cuidar sempre de mim incondicionalmente. De momento atravessa uma fase difícil e espero que recupere rapidamente. As melhoras avó.

Por último, queria dedicar esta dissertação ao meu avô Vasco Alves. Gostaria muito que estivesse comigo no terminar desta etapa tão importante ao tornar-me engenheiro, e sei que estaria orgulhoso. Abraço avô.

Índice

1	Introdução	1
1.1	Enquadramento e motivação da dissertação	1
1.2	Objetivos da dissertação	2
1.3	Método seguido na dissertação	3
1.4	Estrutura da dissertação	4
2	Estado da Arte	5
2.1	Estado da Arte Geral	5
2.1.1	Tecnologia PLC	5
2.1.2	Aspetos da programação repetitiva de sistemas baseados em PLCs	8
2.1.3	Geração automática de código	9
2.2	Estado da Arte Industrial	16
2.2.1	Engineering Editor	16
2.2.2	Inputs	17
2.2.3	Outputs	23
3	Metodologia praticada na Consoveyo S.A.	32
3.1	Descrição geral do sistema	32
3.2	Princípio de funcionamento dos transportadores	35
3.2.1	Composição dos transportadores	35
3.2.2	Modos de funcionamento	37
3.3	Ferramentas e Metodologias	39
3.3.1	AutoCAD	39

3.3.2	Excel	40
3.3.3	Siemens TIA Portal e SIMATIC STEP 7	40
3.3.4	Eplan	41
3.3.5	<i>Warehouse Management System</i> (WMS) e <i>Material Flow Control</i> (MFC)	42
3.4	Protocolos de comunicação utilizados	42
3.4.1	PROFINET	42
3.4.2	Rede AS-i (<i>Actuator Sensor - Interface</i>)	43
4	Abordagem prática - Geração de código PLC	44
4.1	Ferramentas e metodologias utilizadas	44
4.1.1	Sharp 7	45
4.1.2	TIA Portal Openness	47
4.1.3	Ficheiros XML	48
4.1.4	TIA Portal	50
4.2	Objetivo da aplicação	51
4.3	Estrutura da aplicação	52
4.4	Construção do layout em Visio	54
4.5	Interface de controlo de ordens	60
4.6	Geração automática de código para PLC	62
4.6.1	Interface da aplicação	63
4.6.2	Geração de ficheiros XML	64
4.6.3	Manipulação de ficheiros XML	66
4.6.4	Importação dos ficheiros XML para o TIA Portal	81
4.6.5	<i>Function Blocks</i> desenvolvidos no TIA Portal	82
5	Abordagem prática - Geração do ambiente de simulação	95
5.1	Ferramentas e metodologias utilizadas	95
5.1.1	Ficheiros XML	95
5.1.2	Factory IO	96

5.2	Geração automática do cenário de simulação no Factory IO	96
5.2.1	Manipulação dos ficheiros XML	97
5.2.2	Ficheiro XML original	98
5.2.3	Colocação dos equipamentos no ambiente virtual	98
5.2.4	Associação de <i>tags</i> e variáveis	103
6	Conclusões e perspectivas de trabalhos futuros	106
6.1	Conclusões	106
6.2	Trabalhos Futuros	107

Siglas

AGV - Automated Guided Vehicle

API - Application Programming Interface

CAD - Computer-Aided Design

DB - Database Block

EE - Engineering Editor

FB - Funcion Block

GSD - General Station Description

HMI - Human-Machine Interface

IEC - International Electrotechnical Commission

LD - Ladder Diagrams

MFC - Material Flow Control

PLC - Programmable Logic Controller

SCADA - Supervisory Control And Data Acquisition

SCL - Structured Control Language

UDT - User Data Type

XML - Extensible Markup Language

WMS - Warehouse Management System

Índice de Figuras

1.1	Armazém automático da empresa Asia Pacific Food Industry [1]	2
2.1	Modelo de Software pela norma IEC 61131-3 [2]	8
2.2	Interface da aplicação desenvolvida	14
2.3	Ambiente de simulação modelo criado para verificação do código gerado . .	15
2.4	Esquema representativo do funcionamento do Engineering Editor	17
2.5	Biblioteca de blocos no Visio (<i>documentação interna Aberle</i>)	18
2.6	Legenda dos elementos existentes numa posição	18
2.7	Exemplo de parte de um layout concebido no Visio	19
2.8	Ficheiro Eplan (.sdf), que vai ser inserido no Engineering Editor, visto em modo de texto	20
2.9	Lista de equipamentos a serem utilizados numa instalação	21
2.10	Árvore de hardware e ligações no TIA	22
2.11	Exemplo de ficheiro de texto com informação do hardware, gerado do TIA Portal	23
2.12	Funções já existentes de controlo de transportadores	24
2.13	Nomenclatura blocos de funções	25
2.14	Exemplo de uma configuração de 1 posição para outra	25
2.15	<i>Function block</i> FE1001 no TIA Portal	26
2.16	DB.Position vista no TIA Portal	27
2.17	UDT_DriveFlag vista no TIA Portal	28
2.18	Gráficos temporais das variáveis <i>Forward</i> , <i>Forward_Discharge</i> e <i>TakeOver</i> durante a passagem de uma paleta	28
2.19	UDT_DriveTypes vista no TIA Portal	29

2.20	UDT_Drive vista no TIA Portal	30
2.21	Aplicação OpennessDemo	31
2.22	Exemplo de um ficheiro Excel gerado correspondente às tabelas de monitorização	31
3.1	Layout geral de um sistema concebido pela Consoveyo S.A.	33
3.2	Layout - Zona 1	33
3.3	Layout - Zona 2	34
3.4	Layout - Zona 3	35
3.5	Interior de um transportador de correntes, desenvolvido pela Consoveyo S.A.	36
3.6	Transportador de rolos desenvolvido pela Consoveyo S.A.	37
3.7	Interior de uma mesa elevatória, desenvolvida pela Consoveyo S.A.	37
3.8	Diagrama de estados possíveis em modo automático	38
3.9	Ecrã de controlo dos transportadores da consola de operação	39
3.10	Código esqueleto gerado pelas macros no Excel	40
3.11	Exemplo de uma UDT no TIA Portal, com a informação referente à configuração de um transportador	41
4.1	Compatibilidade de funções com PLCs	45
4.2	DB a alterar no TIA Portal	45
4.3	Criação do servidor no software <i>NetToPLCsim</i>	46
4.4	Interface de alteração de variáveis no TIA Portal	46
4.5	Interface da aplicação <i>StartOpenness</i>	47
4.6	Interface da aplicação <i>OpennessDemo</i>	48
4.7	<i>Schema</i> do ficheiro XML <i>Orders</i>	49
4.8	Exemplo de um ficheiro XML	50
4.9	Layout criado em Visio	51
4.10	Diagrama representativo do funcionamento da aplicação	52
4.11	Ficheiro XML correspondente ao desenho criado em Visio, extraído do ficheiro de formato <i>.vsdx</i>	53
4.12	<i>Stencil</i> criado para a construção do layout em Visio	54

4.13	Atribuição das variáveis na <i>shape data</i> de um bloco	55
4.14	Bloco <i>Start</i> no Visio (esquerda) e <i>Emitter</i> no Factory IO (direita)	56
4.15	Informação a inserir no bloco <i>End</i>	56
4.16	Bloco <i>End</i> no Visio (esquerda) e <i>Remove</i> no Factory IO (direita)	56
4.17	<i>Shape Data</i> de um <i>Roller conveyor</i>	57
4.18	Bloco <i>Roller conveyor</i> no Visio (esquerda) e no Factory IO (direita)	57
4.19	<i>Shape Data</i> de uma <i>Transfer table</i>	58
4.20	Bloco <i>Transfer table</i> no Visio (esquerda) e no Factory IO (direita)	58
4.21	<i>Shape Data</i> de uma <i>Rotative table</i>	59
4.22	Bloco <i>Rotative table</i> no Visio (esquerda) e no Factory IO (direita)	59
4.23	Informação associada a um <i>Dynamic connector</i> , representada no ficheiro XML	60
4.24	Interface da aplicação de controlo de ordens	61
4.25	Aspeto das ordens no ficheiro XML de armazenamento	61
4.26	Global DB com a informação referente às ordens	62
4.27	Exemplo de um bloco gerado para um determinado equipamento no layout criado	63
4.28	Interface .NET da aplicação desenvolvida no Visual Studio	64
4.29	Elementos principais dos ficheiros XML provenientes do TIA Portal	65
4.30	Exemplo dos elementos principais e o <i>CodeBlock</i> de cada tipo de ficheiro. 1) <i>Global DB</i> ; 2) <i>Tag Table</i> ; 3) <i>Instance DB</i> ; 4) <i>OB</i>	65
4.31	Ficheiro XML criado com o código da figura 4.2	67
4.32	Ferramenta do Visual Studio que permite a geração de <i>schemas</i>	68
4.33	Exemplo de um ficheiro XML de uma <i>Tag Table</i> sem informação	68
4.34	Parâmetros para definição de uma variável numa <i>tag table</i>	69
4.35	Ficheiro XML correspondente a uma <i>Instance DB</i>	71
4.36	Variáveis existentes na <i>Instance DB</i> para controlo de um transportador de rolos	71
4.37	Exemplo da <i>Global_DB_RC</i> gerada, vista no TIA Portal	72
4.38	Parte do ficheiro XML correspondente à <i>Global_DB_RC</i>	73

4.39	Zona do documento XML onde se encontram definidos os IDs de cada elemento da <i>Global_DB_RC</i>	73
4.40	<i>Global_DB_Zones</i> vista no TIA Portal	74
4.41	Definição dos <i>Equipment_ID</i> no documento XML da <i>Global_DB_Zones</i>	74
4.42	Dois primeiros IDs no ficheiro XML do OB	75
4.43	Network com a lógica de controlo do <i>emitter</i> no TIA Portal	75
4.44	Codificação no ficheiro XML do controlo do <i>emitter</i>	75
4.45	Estrutura do ficheiro XML de um OB	76
4.46	Elementos no ficheiro XML correspondentes a várias <i>networks</i>	77
4.47	Diagrama de funcionamento do método <i>EnterNetwork</i>	77
4.48	Elementos no ficheiro XML correspondentes à definição das variáveis	78
4.49	Lógica para escolha dos métodos de colocação de variáveis	78
4.50	Processo de colocação de informação de uma variável no <i>function block</i> . . .	79
4.51	Elementos no ficheiro XML correspondentes à informação do <i>function block</i> e <i>instance DB</i>	80
4.52	Lógica da colocação de informação acerca do <i>function block</i> e <i>instance DB</i> .	80
4.53	Estrutura do elemento <i>Wires</i>	80
4.54	Lógica da colocação de informação do elemento <i>Wires</i>	81
4.55	Informação contida no elemento <i>ObjectList</i> de uma <i>network</i>	81
4.56	Exemplo de variáveis locais de um <i>function block</i>	82
4.57	Exemplo da programação de um <i>function block</i>	83
4.58	Exemplo de um <i>FB_RC</i> instanciado, visto no TIA Portal	84
4.59	Grafcet funcional do <i>FB_RC</i>	85
4.60	Estrutura auxiliar (contador) do grafcet do <i>FB_RC</i>	86
4.61	Lógica da ativação da variável <i>Mem_Hold</i>	86
4.62	Exemplo de um <i>FB_TT</i> instanciado, visto no TIA Portal	89
4.63	Grafcet funcional do <i>FB_TT</i>	90
4.64	Estruturas auxiliares do bloco <i>FB_TT</i>	91
4.65	Exemplo de um <i>FB_RT</i> instanciado, vista no TIA Portal	92
4.66	Grafcet funcional do <i>FB_RT</i>	93

4.67	Estruturas auxiliares do bloco FB_RT	94
5.1	Sistema periférico criado no Visio e respetivo cenário virtual	96
5.2	Estrutura XML de um cenário de Factory IO vista no <i>Bloco de notas</i>	97
5.3	Significado de alguns elementos do ficheiro XML	98
5.4	Equipamentos existentes num cenário virtual	99
5.5	Exemplo do elemento correspondente à colocação de um transportador de rolos	99
5.6	Significado dos elementos filho do elemento <i>Proxy</i>	100
5.7	Lógica da colocação de equipamentos no ambiente virtual	101
5.8	Significado dos restantes elementos de um objeto	103
5.9	Elemento <i>Drivers</i> e respetivos elementos filhos	103
5.10	Informação correspondente às ligações entre <i>tags</i> do Factory IO e variáveis do TIA Portal, vista no Factory IO	104
5.11	Informação correspondente às ligações entre <i>tags</i> do Factory IO e variáveis do TIA Portal, vista no ficheiro XML	104

Índice de Tranches de Código

4.1	Exemplo de um <i>XPath</i> usado na aplicação desenvolvida para chegar a um elemento	66
4.2	Código C# usado na criação de um ficheiro XML	66
4.3	Programação PLC para encontrar a zona de destino consoante o código RFID	84
4.4	Programação em SCL que associa a direção que uma palete deve seguir com base no destino da mesma	87
4.5	Programação em SCL que grava o destino de cada palete que vai passar por uma determinada mesa de transferência ou mesa rotativa	87

Introdução

As exigências do mundo industrial têm vindo a aumentar, e a área de automação não é exceção. Estas exigências estão associadas com o aumento da qualidade dos produtos e à diminuição do tempo que demoram até ao cliente. No desenvolvimento de armazéns automáticos existem equipamentos e partes de uma instalação que se vão repetir em várias obras. Portanto a geração automática de código para estas situações é um grande avanço na agilização e produtividade de todo o processo de desenvolvimento destes sistemas.

A presente dissertação surge neste contexto: Implementação da estruturação automática de programas PLC de controlo de transportadores seguindo a metodologia do grupo Körber na Consoveyo S.A.. Foi desenvolvida no âmbito da unidade curricular de Dissertação do curso Mestrado Integrado em Engenharia Mecânica, ramo de Automação, da Faculdade de Engenharia da Universidade do Porto. Realizada durante o segundo semestre do ano letivo 2019/2020, com colaboração da empresa Consoveyo S.A., orientada pelo Professor Doutor Germano Manuel Correia dos Santos Veiga e o Engenheiro António Lopes da Consoveyo S.A..

1.1 Enquadramento e motivação da dissertação

Os armazéns automáticos são soluções de gestão de stock e produtos cada vez mais utilizados hoje em dia, em vários tipos de indústrias, como forma de manter a competitividade industrial. Devido à crescente exigência de maior eficiência e menores custos, muitas fábricas encontram nos armazéns automáticos a resposta a esta exigência, graças à enorme fiabilidade e precisão associadas a este tipo de solução.

Na Consoveyo S.A. são desenvolvidas soluções para armazéns automáticos constituídos por vários elementos, entre eles os transportadores, sobre os quais a presente dissertação se foca. Na figura 1.1 temos um exemplo de um armazém automático.

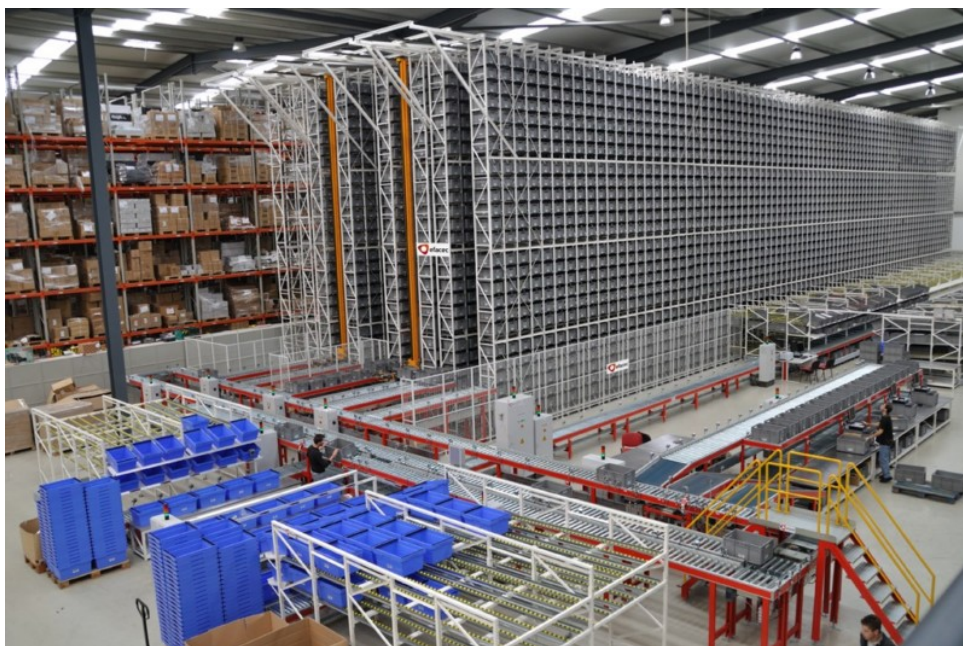


Figura 1.1: Armazém automático da empresa Asia Pacific Food Industry [1]

Os transportadores são equipamentos chave que permitem um transporte eficiente e preciso de produtos entre as várias áreas operacionais que constituem um armazém automático e são também os componentes de eleição de interação com outros equipamentos, permitindo que estes realizem corretamente a sua tarefa, como controlo dimensional da carga, verificação de paletes e interface com envelopadoras, etiquetadoras e robôs. Dependendo do que está a ser transportado, desde cargas leves a pesadas, entre as diferentes variações deste equipamento existe um apropriado para realizar esse transporte.

Além dos transportadores, outro equipamento encontrado na maioria dos armazéns automáticos são os transelevadores. Os transelevadores são máquinas rápidas e com elevada exatidão usadas para armazenar as cargas no armazém em si. Normalmente, estes recolhem uma paleta proveniente de um transportador e colocam-na no espaço destinado.

1.2 Objetivos da dissertação

A criação de um programa que permita a geração automática de código PLC representa um passo enorme na programação de sistemas automatizados que têm alta complexidade mas cujos componentes principais se repetem com alguma frequência, como é o caso dos sistemas desenvolvidos pela Consoveyo S.A..

Na presente dissertação foi definido como objetivo a criação de uma aplicação de geração automática de código para o PLC de controlo de um sistema periférico, composto por vários tipos de transportadores. Como interface com o utilizador foi escolhido o software Visio para conseguir uma perspetiva ilustrativa da disposição dos equipamentos. Para além da geração de código, outro objetivo definido foi a geração automática de um ambiente de simulação 3D para verificação do código gerado.

Uma aplicação com esta capacidade iria reduzir o tempo de programação manual, bem como os erros associados a esta, aumentando a eficiência de uma equipa, uma vez que as

tranches de código consideradas rotineiras iriam ser geradas automaticamente, permitindo ao programador concentrar-se em casos mais específicos onde é necessário código mais customizado.

Associado à geração automática de código encontra-se a padronização do mesmo, existindo assim apenas um modelo de programação na Consoveyo S.A.. Este modelo bem definido permite à empresa ter uma maneira mais estável de desenvolver projetos o que se traduz num comissionamento e manutenção mais fáceis e torna os cálculos de tempo e orçamento necessários mais precisos.

A geração automática de um ambiente de simulação virtual torna rentável o uso deste tipo de software ao existir um ponto extra de validação da solução encontrada sem ter que abdicar de tempo nem recursos humanos exagerados.

1.3 Método seguido na dissertação

A pesquisa bibliográfica foi dividida em duas secções principais: a bibliografia geral e a bibliografia industrial. Na primeira foi abordada a tecnologia PLC a flexibilização da programação dos mesmos. Esta recolha de informação facilitou a contextualização da segunda parte da pesquisa bibliográfica, onde é abordada a metodologia praticada pela empresa Aberle, empresa que integra o grupo Körber. Nesta secção encontra-se documentado o estudo da ferramenta Engineering Editor, criada pela Aberle, que serviu como ponto de partida para o desenvolvimento da aplicação criada no âmbito desta dissertação.

Posteriormente foram delineados todos os processos necessários para o desenvolvimento da aplicação e seleccionadas as ferramentas a usar, com base nas potencialidades de cada uma e a sua compatibilidade na Consoveyo S.A..

Para a interface com o utilizador foi escolhido o Visio por permitir uma elaboração gráfica do layout simples e intuitiva. Para a geração de código foi selecionado o TIA Portal Openness devido às suas funcionalidades e capacidade de interação com o TIA Portal, software utilizado na Consoveyo S.A. para programação de PLCs. Por último foi escolhido o Factory IO como simulador 3D para a verificação de código devido à familiaridade ganha durante o mestrado de Automação.

Para a geração automática tanto de código PLC como do ambiente de simulação 3D foram necessários conhecimentos sobre ficheiros XML (*Extensible Markup Language*) e manipulação dos mesmos. Foram também adquiridos conhecimentos sobre programação na linguagem C# para a elaboração da aplicação e edição dos documentos XML. Foi também necessária a familiarização com o software TIA Portal, para a execução da programação de controlo de PLCs.

Depois de traçado o plano e adquiridas as aptidões necessárias, procedeu-se à realização das tarefas na seguinte ordem:

1. Desenvolvimento dos blocos de controlo dos equipamentos no TIA Portal;
2. Criação da biblioteca ilustrativa de blocos no Visio, para construção do layout;
3. Desenvolvimento da aplicação C# para leitura e edição de documentos XML para:
 - (a) Extração de informação do layout no Visio;

- (b) Construção do programa PLC;
- (c) Interface de controlo de ordens/encomendas;
- (d) Geração do ambiente de simulação 3D, fiel ao layout construído em Visio.

1.4 Estrutura da dissertação

A presente dissertação encontra-se dividida em 5 capítulos, sendo um deles este capítulo introdutório.

Na primeira parte do capítulo 2 é feita uma revisão do estado da arte, onde se encontram conhecimentos teóricos sobre o tema, bem como o enquadramento industrial no qual a dissertação se encontra inserida. É abordada a tecnologia PLC e a flexibilização da correspondente programação, relacionando estes dois temas com os armazéns automáticos.

Na segunda parte do capítulo 2 é abordado a metodologia seguida atualmente por uma empresa do grupo Körber, a Aberle. Esta parte foca-se em todo o processo, desde a primeira formação do layout do armazém até ao código final gerado. É referido o software essencial desenvolvido para tornar este processo possível, e explicadas as interações com os restantes software necessários para a implementação de um projeto.

No capítulo 3 são descritos os sistemas desenvolvidos e a metodologia seguida atualmente na Consoveyo S.A.. São abordados também os transportadores utilizados e como funcionam quando se encontram inseridos num sistema periférico. Quanto à metodologia seguida, são enumeradas as ferramentas utilizadas e a respetiva função.

O capítulo 4, descreve a abordagem prática da dissertação, nomeadamente a aplicação desenvolvida para controlo e geração do ambiente simulação de um sistema periférico.

O capítulo 5 contém as conclusões retiradas desta dissertação e sugestões para projetos futuros.

2

Estado da Arte

Este capítulo encontra-se dividido em duas secções: o estado da arte geral e o estado da arte industrial.

No estado de arte geral é abordada a pesquisa bibliográfica realizada, relacionada com o tema da presente dissertação. No estado de arte industrial é abordada a metodologia existente para a programação de sistemas periféricos, criada pelo grupo Körber.

2.1 Estado da Arte Geral

Na presente secção é feita uma primeira abordagem aos armazéns automáticos, uma vez que o tema desta dissertação se debruça sobre os transportadores, um componente essencial em qualquer armazém automático. Posteriormente é feita uma pequena descrição da evolução da tecnologia PLC até aos dias de hoje, bem como a principal norma de referência destes equipamentos (Norma IEC 61131-3).

Por último expõem-se ferramentas já existentes que realizam algum tipo de geração automática de código e aborda-se a flexibilização da programação de PLCs, de uma perspetiva do tema desta dissertação.

2.1.1 Tecnologia PLC

Nesta subsecção são abordados os PLCs (*Programmable Logic Controller*), nomeadamente, a sua evolução, a norma internacional que rege estes dispositivos e as várias maneiras de programação dos mesmos, bem como os métodos e ferramentas para flexibilizar.

2.1.1.1 Evolução dos PLCs

O autor em [2] definiu os PLCs como equipamentos programáveis, destinados ao controlo industrial de aplicações. Inicialmente, estes equipamentos eram simples dispositivos lógicos, programáveis por eletricitistas, quer servissem para substituir soluções de controlo baseadas em relés.

Ainda em [2] é referida a evolução destes equipamentos, sendo introduzidos temporizadores, contadores, suporte de variáveis analógicas e a possibilidade de realização de operações matemáticas necessárias a algum controlo específico.

Hoje em dia, um PLC é um sistema computadorizado bastante potente, necessário em vastos sistemas industriais devido à sua enorme versatilidade de funções e capacidade de comunicação com outros sistemas. Outras características destes dispositivos, referidas pelo autor de [2], que justificam ainda mais o seu uso em ambientes industriais passa pela sua boa adaptação aos mesmos, onde conseguem ter um correto funcionamento apesar de ruídos eletromagnéticos, atmosferas sujas, vibrações consideráveis e temperaturas e humidades particularmente baixas.

Em [2] é descrita a constituição mais usual de um PLC:

- CPU: unidade central de processamento;
- Memórias RAM e ROM;
- Portas de entrada e saída (sinais digitais e analógicos) e portas de comunicação com outros sistemas;
- Outros atributos, entre eles:
 - Relógio de tempo-real;
 - Contadores em hardware;
 - Sistema operativo;
 - Barramentos internos.

2.1.1.2 Norma IEC 61131-3

A norma IEC 61131-3 foi desenvolvida pela Comissão Eletrotécnica Internacional (IEC) com o intuito de resolver problemas relacionados com a construção, programação e exploração de PLCs, passando a solução pela standardização de linguagens de programação e de outras funcionalidades referentes aos PLCs [2].

O objetivo desta normalização é facilitar o entendimento entre diferentes fabricantes de PLCs e entre programadores. Um exemplo prático deste cenário é a facilidade de comunicação entre alguém que esteja a trabalhar num projeto de automação e um programador de PLC, que se traduziria numa melhor compreensão do programa e eventuais alterações ao mesmo por parte de terceiros.

A normalização destes aspetos passou pela normalização faseada de [3]:

- Informação geral;
- Hardware e requisitos de teste;
- Linguagens de programação;
- Interface com o utilizador;
- Comunicações.

A programação dos primeiros PLCs era implementada em diagrama de contactos elétricos, sendo esta linguagem ainda hoje muito utilizada. Porém, dado a enorme evolução dos PLCs, as capacidades básicas desta linguagem não eram suficientes, obrigando ao desenvolvimento de novas linguagens [3], sendo que da última atualização das mesmas em 2013 resultaram:

- Textual:
 - Texto estruturado - ST;
 - Lista de instruções - IL;
- Gráfica
 - Ladder - LD;
 - Diagrama de blocos funcionais - FBD;
- Sequential Function Chart - SFC.

Segundos os autores em [3], o modelo de software na norma IEC 61131-3, ilustrado na figura 2.1, foi desenvolvido para ser compatível com controladores lógicos programáveis mais avançados, bem como sistemas onde existem vários CPUs. Nesse modelo existem vários conceitos usados para descrever os componentes desses sistemas.

A normalização especifica uma hierarquia inerente à estrutura de um programa. Dentro desta hierarquia existem elementos que estão presentes em todos os PLCs, apesar de alguns fabricantes usarem outros nomes para os caracterizarem. São esses elementos:

- Configuração: engloba tudo que é necessário à realização de uma aplicação de controlo num ou mais PLCs;
- Recurso: algo que efetua processamento de informação (CPU);
- Tarefa: mecanismo de controlo de programas;
- Programa: software de aplicação cuja execução é "lançável";
- Unidades de organização da programação: incluem programas, *function blocks* ou *functions* que podem ser reutilizados em forma de instâncias;
 - Programa: torna possível a reutilização de software a um nível lato. Por exemplo, permite criar um programa para controlar a posição de um motor e usá-lo em diversas aplicações;
 - *Function block*: têm um carácter sequencial e podem implementar algoritmos de controlo para situações correntes;
 - *Function*: têm cariz combinacional e são especialmente usadas para manipulação de dados;

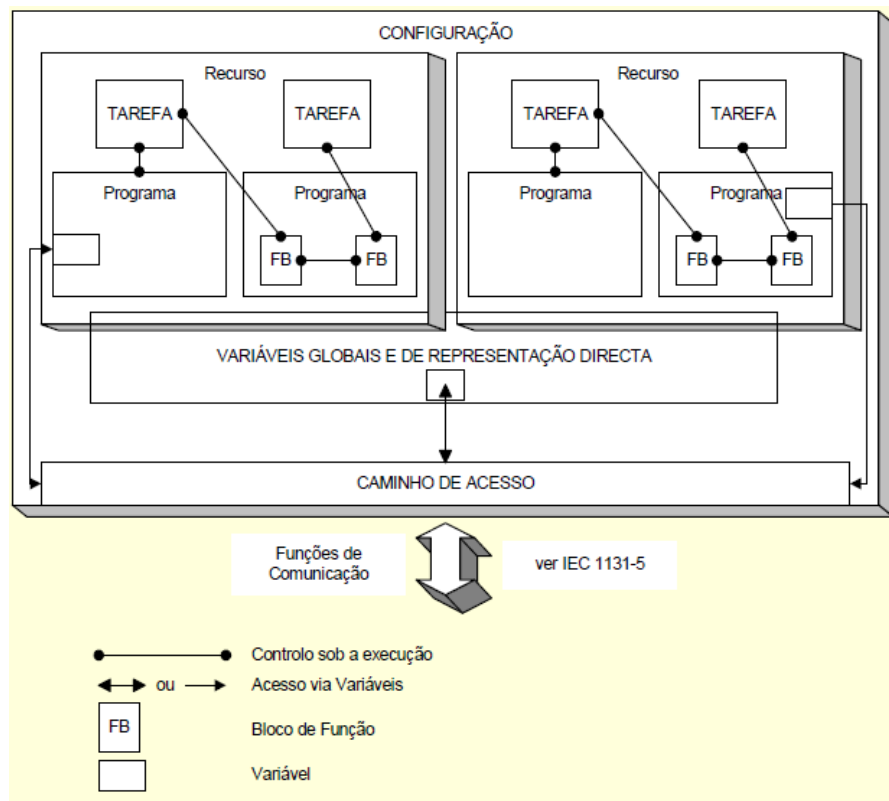


Figura 2.1: Modelo de Software pela norma IEC 61131-3 [2]

2.1.2 Aspectos da programação repetitiva de sistemas baseados em PLCs

Os sistemas de controlo industriais são um tipo especial de sistemas computadorizados que controlam dispositivos físicos através de comandos lógicos sequenciais. Além disso, normalmente têm que responder a eventos externos, provenientes do ambiente que controlam. Para que os requisitos operacionais de cada instalação industrial sejam cumpridos são normalmente usados PLCs que comunicam através de redes de campo industriais (*fieldbuses*) e dispositivos de I/O dedicados. Todos estes elementos executam as suas tarefas em harmonia, juntamente com standards no campo de aplicação, metodologias para definir o design de aplicação e ferramentas para auxiliar o ciclo de desenvolvimento. O objetivo é melhorar a produção e otimizar todos os processos, o que se traduz numa maior cadência e qualidade dos produtos, enquanto diminuem a mão de obra [4].

Como os autores de [5] referem, hoje em dia a complexidade industrial associada ao controlo de equipamentos aumenta constantemente, sendo por vezes necessário a implementação de sistemas com vários PLCs conectados. Além deste facto, os diferentes requisitos de segurança e fiabilidade fazem com que os programas tenham que ser elaborados com um maior rigor e cuidado. A validação dos programas é maioritariamente realizada através de testes e simulações.

Nos sistemas abordados nesta dissertação, os armazéns automáticos, a fiabilidade e segurança são dois dos aspetos mais importantes, conseguidos com o uso de PLCs. A maioria dos equipamentos existentes num armazém automático têm que apresentar um determinado funcionamento com base em informações ou sinais recebidos em tempo real, logo, é necessário usar um aparelho de controlo que permita este tipo de resposta imediata.

No que diz respeito à segurança, visto que os equipamentos e objetos transportados neste tipo de armazéns são de grande porte, é crucial que o funcionamento dos mesmos não ponha em perigo vidas humanas. Para isso, nos programas deste tipo de sistemas, tem que existir sistemas dedicados à gestão das segurança tais como PLC's de segurança, que garantem o correto funcionamento da máquina e possuem um conjunto de características mais estritas que um PLC convencional.

Como os autores em [6] constataam, todos os dias são lançadas novas soluções e funcionalidades associadas aos PLCs, com o objetivo de alcançar maiores níveis de competitividade. Quando uma empresa compra vários PLCs a diferentes fabricantes, tendo cada um o seu modo específico de funcionamento, os utilizadores deparam-se com um problema: a dificuldade de reutilizar programas, e partilhar programas entre os diferentes dispositivos. Uma solução para este problema seria ter tantos programadores especializados quanto PLCs diferentes, porém isso seria uma solução pouco viável. Outra solução, sendo esta mais duradoura, seria a criação de um formato inicial "neutro" que descrevesse a lógica a ser seguida por um controlador e a sua tradução para código standard PLC. Por outras palavras, a geração automática de código PLC partindo de informação lógica. Foi provado pelos autores em [7] que um programa escrito numa linguagem lógica pode ser diretamente traduzido em código destinado a realizar uma tarefa, para um modelo particular, e exportado para um controlador. Esta abordagem permite ao utilizador criar e gerir uma biblioteca de "objetos", definindo um conjunto de instâncias associadas a métodos, que podem ser utilizados em qualquer fase da programação. Quanto mais praticada esta abordagem, maior a biblioteca se torna, o que leva à contínua evolução do desenvolvimento de um software de programação.

Neste contexto, o mundo industrial que recorre à programação de PLCs para o controlo de diversos dispositivos necessita cada vez mais de uma solução para agilizar a programação e torná-la mais intuitiva e precisa. Nesta procura a geração automática de código surge como o principal candidato a resolver os problemas referidos. Importante referir que a geração automática de código para PLCs não é uma solução para todos os casos com uso a este tipo de aparelhos, uma vez que nem todos os equipamentos controlados por estes dispositivos são padronizados. Porém, em armazéns automáticos, nomeadamente os desenvolvidos pela Consoveyo S.A., em que os equipamentos utilizados não apresentam diferenças significativas a nível de programação de instalação para instalação, mas que possuem alguma complexidade, a geração automática de código apresenta vários benefícios. De seguida, estudamos de uma forma mais aprofundada esse tema, uma vez que a presente dissertação foi redigida à volta do mesmo.

2.1.3 Geração automática de código

Como foi referido na secção 2.1.2, uma possível abordagem à flexibilização da programação de PLCs é a geração automática de código. Vão surgindo cada vez mais estudos e artigos relacionados com este tema, porém não existe ainda nenhuma ferramenta de eleição que cumpra perfeitamente a função de gerar código PLC a partir de informação lógica.

O design e desenvolvimento de um programa PLC para controlar um armazém automático ou o fabrico flexível de algum produto é algo extremamente complexo. Apesar de ser relativamente fácil o desenvolvimento de um programa deste género para um sistema pequeno, à medida que o programa vai aumentando, a sua verificação torna-se cada vez mais difícil. Devido à dificuldade de testar extensa programação, convém basear as

chamadas de código em blocos de funções já devidamente testados. Um atraso ou erro na escrita/geração de código traduz-se, muitas vezes, na perda de receita ou até clientes [8].

Podemos identificar como as principais vantagens, apontadas pelos respetivos autores:

- Em [8]: A reutilização de código, usando "templates" previamente verificados, para diferentes fases da programação;
- Em [9]: Na mudança de layout numa fábrica (de uma linha de produção por exemplo), encurta o processo de programação, alterando para uma nova solução de uma maneira mais rápida, reduzindo os custos que acompanham essa mudança;
- Em [8]: Redução do tempo de desenvolvimento e aumento da produtividade através da modelação e compilação rápida;
- Em [8]: Redução do tempo de programação "manual" e dos erros adjacentes possivelmente introduzidos pelos programadores.

No mundo industrial temos um exemplo de geração automática de código. Ao contrário do que se verifica na programação de PLCs, a programação de máquinas industriais (CNC) é maioritariamente realizada através de geração automática de código G, usando componentes chamados pós-processadores.

O processo começa pela criação de um modelo, através de um programa de fabrico assistido por computador, da peça a maquinar (ficheiro CAD). Posteriormente é gerada a definição do movimento da ferramenta e especificações de maquinagem, através de um sistema CAM, chamado ficheiro CLDATA (*Cutter Location Data File*). De seguida entram em cena os pós-processadores, que são os equipamentos responsáveis por converter o programa de maquinagem na linguagem da própria máquina, em código G [10].

2.1.3.1 Condições a cumprir por código gerado de forma automática

Hoje em dia, o desenvolvimento de sistemas controlados por PLCs seguem a metodologia conhecida por "modelo V", abordada em [11], que consiste na análise dos requisitos, design do sistema, implementação, teste, verificação e validação. Este modelo ajuda muito na elaboração deste tipo de sistemas, porém, ainda tem algumas desvantagens dado que muitos dos erros são detetados apenas depois da implementação. Para alcançar uma maior eficácia, convém que a deteção de erros seja feita o mais cedo possível. Por outro lado, a verificação não é bem integrada neste processo, pois, mesmo que o design esteja bem estruturado, é difícil manter o produto final de acordo com o mesmo devido à implementação manual. Este obstáculo pode ser facilitado com a integração da instalação num programa de simulação 3D, porém, irão sempre existir situações não contempladas ou inesperadas na implementação do sistema real.

Todos estes fatores apontam para a necessidade de uma ferramenta que faça a implementação, teste e verificação de uma maneira combinada, de modo a tirar maior proveito. É também importante que esta ferramenta torne o código reutilizável para conseguir manter-se a par da crescente exigência do mercado [11].

De acordo com os autores em [11], no desenvolvimento de uma ferramenta de geração automática, a mesma deve seguir as seguintes premissas:

1. Modularidade: o modelo deve ser detalhado o suficiente para ser implementado, bem estruturado e organizado para que seja fácil a sua manutenção e reutilização;
2. Clareza: depois dos modelos estarem definidos, a sua simulação e verificação devem ser facilmente realizadas, para serem o mais eficientes possível;
3. Boa sintaxe: o código deve estar delineado para que, caso seja necessário realizar alguma alteração ou melhoria, o programador o faça sem problemas.

2.1.3.2 Metodologias existentes para a geração automática de código PLC

Na pesquisa bibliográfica de artigos referentes à geração automática de código, foram encontradas diversas metodologias que abordam este tema, porém, foram encontrados muito poucos artigos e fontes que explicassem como a geração em si acontecia.

Métodos Formais

O recurso aos métodos formais permite fazer a especificação formal, partindo da especificação informal, seguindo para a implementação e realização. O termo "informal" refere-se a qualquer coisa que não é baseado num princípio bem composto e semanticamente e sintaticamente bem definido. De um modo geral, a especificação informal consiste na descrição de um processo não controlado e nos requisitos para o sistema a ser controlado. Atualmente, na maior parte dos casos, o design de um sistema de controlo lógico começa com a especificação informal do problema de controlo. Posteriormente é feita a implementação direta do controlador através de programação do PLC para chegar à fase da realização e em seguida é feita a validação, através da comparação de resultados do controlador implementado com os da especificação informal do sistema [12].

Os autores em [12] defendem que a programação industrial de PLCs irá ser cada vez mais suportada por métodos formais devido à crescente complexidade dos problemas de controlo, a exigência de tempos de desenvolvimento reduzidos e ao ter que permitir a reutilização de blocos de software pré-existent, resultando numa abordagem formal à programação de PLCs. A exigência de soluções de alta qualidade e a aplicação de PLCs em processos onde a segurança é um fator crítico, necessitam de procedimentos de verificação e validação, principalmente ao nível de ambiguidade e coerência, a partir de métodos formais.

Engenharia Orientada a Objetos

A Engenharia Orientada a Objetos (MDE - *Model-Driven Engineering*) é uma metodologia de desenvolvimento de sistemas que permite a construção de uma aplicação, ou parte dela, partindo de modelos. Para poder usar modelos para a geração de código ou documentação, para posteriormente estas gerações serem validadas, simuladas ou verificadas, é necessário alterar os modelos existentes de uma maneira correta. A transformação de modelos é ainda usada para definir como é feita a transformação entre diferentes linguagens, como por exemplo, entre linguagens do utilizador para linguagens de verificação, definidas em DSML (*Domain Specific Modeling Languages*), um domínio usado para representar as diferentes linguagens usadas durante o desenvolvimento de um sistema [13].

A garantia de uma transformação de linguagens bem sucedida é assegurada pelo uso de hierarquia de metamodelos na representação de linguagens DSML. A OMG (*Object Management Group*) definiu uma estrutura piramidal para a modelação da hierarquia de metamodelos: a variável *real word* no nível M0, os modelos que a representam no nível M1, os metamodelos que definem os modelos no nível M2 e no topo da pirâmide, no nível M3, um metamodelo único, MOF (*Meta-Object Facility*), que se define a si próprio. Um metamodelo é um modelo que descreve os elementos que permite a definição das linguagens de modelação [13].

Virtual Engineering

Os autores em [14] defendem que ao introduzir métodos de engenharia inteligentes e tecnologias que permitem a realização de processos de fabrico dinâmicos e de alto nível de reconfiguração, uma empresa consegue responder rapidamente a alterações de situações industriais.

O propósito de uma ferramenta de engenharia virtual é criar modelos digitais para sistemas de manufatura para auxiliar os processos reais. Tipicamente, os modelos virtuais consistem em três aspetos principais: mecânico, elétrico e controlo.

Em [14] é investigado uma metodologia para o desenvolvimento de um modelo digital de montagem de uma célula de combustível, e posteriormente usa esses mesmo modelo para gerar código PLC. O objetivo é demonstrar as capacidades de ferramentas de engenharia virtual para gerar automaticamente e eficientemente código para controlo de aplicações industriais.

2.1.3.3 Exemplos e casos de estudo de geração automática de código PLC

Durante a pesquisa bibliográfica referente à geração automática de código PLC, foram encontradas diversas ferramentas que executam a tarefa, de formas diferentes, precisando cada uma de tipos de informação específicas para a geração.

PLCopen XML Standard

PLCopen XML Standard é uma versão padrão de troca de informação adotada pelo grupo AutomationML com o objetivo de criar um formato de ficheiros XML que contenha toda a informação referente a uma determinada linha de produção ou armazém, como por exemplo geometria 3D, informação cinemática, desdrição de processos e informação lógica [15].

Existem abordagens para alcançar a geração de código através de modelos, mas estas usam formatos de trocas de informação privilegiados, únicos para um determinado software de modelação e fornecedores de PLCs. Devido a este facto, as suas aplicações são limitadas. O objetivo com esta versão é criar standard "neutro" que não tenha as limitações referidas [15].

AutoGen

Software desenvolvido pela EATOPS (*Ergonomics Applied to Operations*), empresa de engenharia especializada no desenvolvimento de sistemas e centros de controlo. Comercializa tecnologia RIVOPS (*Remote Intuitive Visual Operations*), uma combinação de software e hardware ergonómicos [16].

De acordo com a informação encontrada em [16], o AutoGen gera, automaticamente, os controlos e processos lógicos de gestão, onde a "entrada" é uma lista de equipamentos, usando uma nomenclatura consistente em todo o código-fonte. Esta geração é feita através de *python scripts*, permitindo à fábrica a automação dos componentes chave, através da linha de produção, bem como a manutenção do software, tendo em conta que hoje em dia praticamente todos os PLCs integrados em ambientes industriais são compatíveis com o uso de linguagem Python para gerar código. Como em todos os softwares de geração automática de código, a principal vantagem deste é a produtividade, sendo possível gerar o código em segundos, tarefa que demoraria dias/semanas quando feito manualmente.

Simulink PLC Coder

Este software gera código PLC compatível com a norma IEC 61131-3, em texto estruturado (ST) ou ladder (LD), através da leitura de modelos Simulink, gráficos de Stateflow e funções de MATLAB. O código é gerado no formato PLCopen XML, porém são suportados outros formatos, através do uso de IDEs (*Integrated Development Environments*), como por exemplo 3S-Smart Software Solutions CODESYS, Rockwell Automation Studio 5000, Siemens TIA Portal e Omron Sysmac Studio. Como resultado, é permitido ao utilizador compilar e implementar a sua aplicação em vários PLCs ou PACs (*Programmable Automation Controller*). O Simulink PLC Coder gera ainda testes que permitem ao utilizador verificar o seu programa usando simuladores, e, posteriormente, elabora relatórios com métricas de códigos estáticas e rastreio bidimensional entre modelo e código [17].

TIA Portal Openness

O TIA Portal Openness é uma extensão existente nos pacotes de instalação do software TIA Portal. Segundo a informação existente em [18] esta ferramenta possibilita a integração do TIA Portal no desenvolvimento de programas, com o intuito de automatizar tarefas. Permite ainda usar ferramentas externas de desenvolvimento de ambientes para criar aplicações, como por exemplo, um gerador de código para imagens numa HMI e blocos de software de PLC.

Para utilizar esta ferramenta é necessário, ao utilizador, estar familiarizado com o software TIA Portal e as suas funcionalidades, o ambiente de programação do Microsoft Visual Studio e ter ainda conhecimentos básicos de programação (C# / VB.net e .Net). Entre as imensas funções existentes nesta ferramenta, na documentação da Siemens em [19], destacam-se as seguintes:

- Importação e exportação de projetos através de ficheiros XML, tabelas de variáveis e blocos de funções;

- Comunicação com o TIA Portal;
- Execução de processamento de informação bruta e rotinas de testes, enquanto o utilizador configura os programas;
- Configuração de HMIs.

2.1.3.4 Desenvolvimento de uma aplicação para geração automática de código para o PLC de controlo de um shuttle car

Dissertação de mestrado desenvolvida no Mestrado Integrado de Engenharia Mecânica, ramo de Automação, na Faculdade de Engenharia da Universidade do Porto [20]. O tema da presente dissertação aproxima-se ao da mencionada pelo facto de ambas terem sido desenvolvidas na mesma empresa e abordarem o mesmo conteúdo, a geração automática de código PLC. Outro aspeto semelhante são as ferramentas usadas, nomeadamente O TIA Portal e o TIA Portal Openness. Isto deve-se ao facto do TIA Portal ser o software utilizado na Consoveyo S.A. na programação de PLCs, e o TIA Portal Openness ser a ferramenta de geração automática de código que consegue comunicar com o TIA Portal.

Nesta dissertação é abordada uma aplicação .NET desenvolvida em C# que através da manipulação de ficheiros XML é capaz de automaticamente gerar código para um PLC de controlo de um *shuttle car*, equipamento que pode integrar um armazém automático. Um *shuttle car* tem como finalidade transferir paletes entre diferentes linhas de transportadores.

Na interface com o utilizador, figura 2.2, este pode escolher a quantidade de entradas e saídas bem como as suas localizações ao longo do espaço abrangido pelo *shuttle car*. Com base nos dados inseridos, aplicação edita ficheiros XML pré-existentis, conseguidos através da API Demo Openness. Depois de editados estes são importados no software TIA Portal.

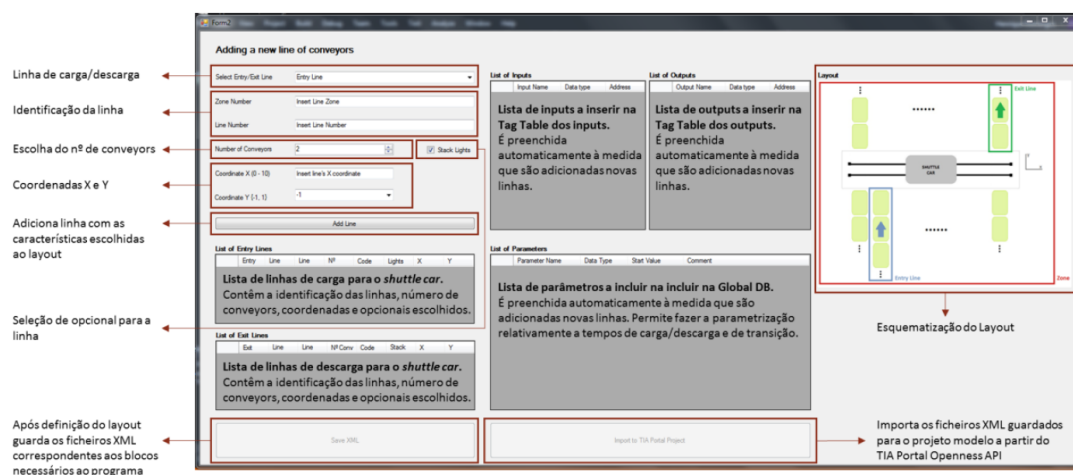


Figura 2.2: Interface da aplicação desenvolvida

Para a manipulação de ficheiros XML foi escolhida a alternativa *XmlSerialize* por permitir utilizar os XML *schemas* fornecidos pela Siemens e o acesso às diversas áreas dos documentos, através da serialização e deserialização de objetos. Posteriormente foram

geradas automaticamente as classes necessárias para a edição dos ficheiros a partir do *Developer Command Prompt* (ferramenta do Microsoft Visual Studio), usando os respetivos XML *schemas*. O resultado deste processo é um ficheiro .cs, utilizável no Visual Studio, com um conjunto de classes que permitem o acesso às diferentes áreas dos documentos XML correspondentes aos diversos blocos no TIA Portal (DB, OB, FB e *Tag tables*).

No código gerado existem vários módulos para o controlo das várias partes do sistema:

- Ordens: módulo para leitura das ordens de carga/descarga a partir de uma listagem;
- Modo de funcionamento:
 - Modo Automático: módulo para interpretação e execução automática da ordem recebida;
 - Modo Manual: módulo de código para movimentação manual do *shuttle car*;
- Linhas de carga e descarga: módulo para verificação da disponibilidade das linhas para receção de cargas e analisar a presença de cargas a transportar;
- Transportadores: caso existam transportadores iniciais, intermédios ou finais, é inserido um módulo (diferente dependendo da zona onde se encontram) para o controlo automático dos transportadores dessa zona.

A importação dos ficheiros XML, depois de editados, para o TIA Portal foi efetuada ao utilizar métodos provenientes da API Demo Openness, mais propriamente da classe *TiaOpennessHelper*. O processo de importação executa as seguintes ações: iniciação de uma instância do TIA Portal, abertura do projeto modelo e importação dos ficheiros XML editados.

Para testar o código gerado e visualizar o comportamento do sistema foi desenvolvido um ambiente modelo de simulação no Factory IO (software de simulação 3D), presente na figura 2.3. Devido às limitações deste simulador, foi usado um transelevador de garfos, uma vez que apresenta um princípio de funcionamento muito semelhante ao *shuttle car*. Neste cenário encontram-se diversas linhas de carga e descarga, compostas por diferentes quantidades de transportadores, dispostas ao longo do curso do *shuttle car*.

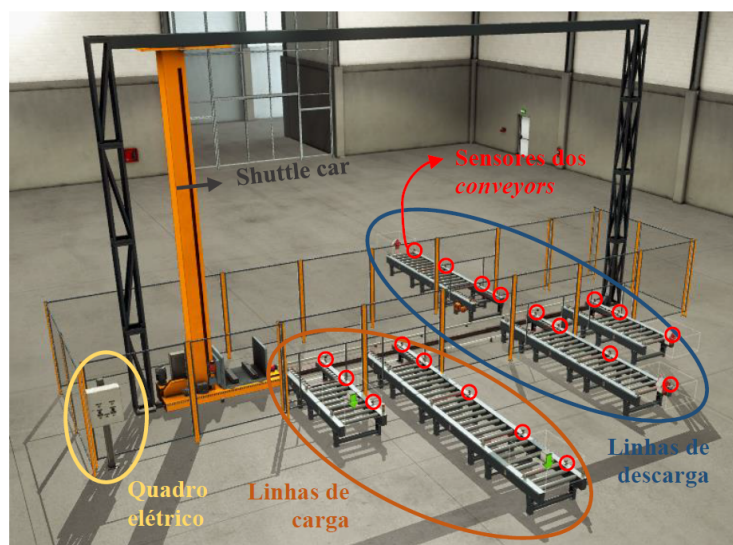


Figura 2.3: Ambiente de simulação modelo criado para verificação do código gerado

2.2 Estado da Arte Industrial

Seguindo o tema desta dissertação, iremos agora abordar a forma de trabalhar do grupo Körber. Esta metodologia foi acompanhada através da empresa Aberle onde já se encontra implementada, com o objetivo de a pôr em prática na Consoveyo S.A..

São expostas as diferentes ferramentas usadas ao longo do processo e foca-se principalmente na interface com o utilizador e como o código gerado se encontra estruturado.

2.2.1 Engineering Editor

Na programação de armazéns automáticos, como em quase todos os tipos de programação, existem diferentes maneiras de realizar a mesma tarefa. Portanto, nesta secção iremos abordar os softwares e metodologias que a Aberle segue.

Os engenheiros da Aberle desenvolveram uma aplicação de geração automática de código, chamada Engineering Editor, que tem como inputs três ficheiros, e gera quatro ficheiros:

- Ficheiros de entrada:
 - Excel gerado do Visio;
 - Ficheiro gerado do Eplan (formato .SDF);
 - Ficheiro de texto com a configuração do hardware da instalação, gerado no TIA Portal;
- Ficheiros gerados:
 - Conexão com etiquetadora para a impressão das etiquetas de identificação;
 - Ficheiro em formato Excel com as posições dos transportadores, sensores e aviso de alarmes, para ser usado no SCADA (WinCC);
 - Ficheiro XML com o código de controlo dos transportadores para ser importado para o TIA Portal;
 - Excel com tabelas de monitorização para facilitarem o comissionamento.

Este programa, usando os ficheiros mencionados, vai gerar cerca de 70% do código necessário para o controlo de uma instalação, incluindo a comunicação com outros softwares como o MFC (*Material Flow Control*). Para além disso, gera ainda mais componentes necessários num armazém automático, como etiquetas dos equipamentos e um ficheiro WinCC com a listagem de todos os equipamentos. No esquema da figura 2.4 está representado este processo.

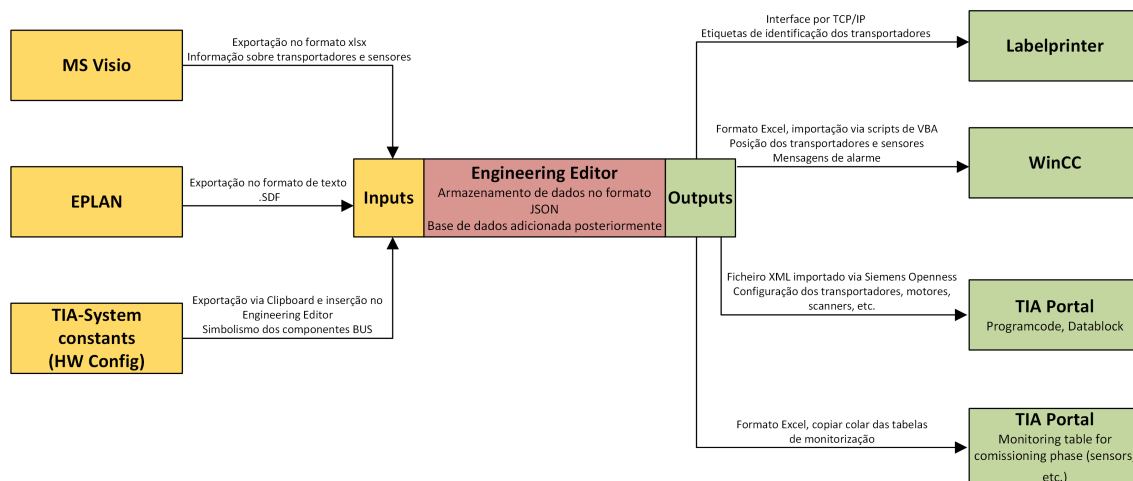


Figura 2.4: Esquema representativo do funcionamento do Engineering Editor

No Engineering Editor o utilizador consegue configurar valores padrão de variáveis, e os nomes de drives, transportadores e dispositivos externos. Esta aplicação, ao "ler" o Excel gerado no Visio, sugere ao utilizador, automaticamente, funções de funcionamento e passagem de paletes entre transportadores, quando esta passagem é simples. Quando as passagens são mais complexas, como por exemplo, no caso de um transportador passar paletes para outros três diferentes transportadores, o Engineering Editor pode também sugerir ao utilizador algumas funções existentes ou o utilizador terá que programar manualmente. A maior parte das vezes, devido à complexidade e aos "casos especiais" de certas zonas, o programador opta pela programação manual para permitir uma maior flexibilidade.

Para a configuração completa no Engineering Editor é necessário associar os símbolos criados no Eplan aos sensores configurados no Visio, e para isso é necessário os nomes coincidirem, caso contrário o software não faz a associação automaticamente, terá que ser feito manualmente.

2.2.2 Inputs

Neste ponto iremos abordar os programas que geram os ficheiros que vão ser inseridos no Engineering Editor. Iremos perceber como funcionam e o seu papel no standard do grupo Körber e na geração automática de código.

2.2.2.1 Visio

O software Microsoft Visio é utilizado para a construção de diversos tipos de diagramas (organogramas, fluxogramas, diagramas de rede, etc.). Este software oferece recursos automatizados para diagramas mais avançados, além da funcionalidade de integração com informação proveniente de outras aplicações [21].

Na Aberle, depois do primeiro contacto com o cliente, o primeiro layout do armazém automático vai ser concebido neste software. Neste layout devem estar devidamente representados todos os pormenores existentes na instalação, ou seja, a partir deste layout um programador deve conseguir perceber tudo o que existe numa instalação, e como funciona.

O layout é concebido usando uma biblioteca de blocos representativos de diferentes soluções de transportadores (criada pela Aberle, figura 2.5), combinando-os para formar sistemas automatizados. Nesses blocos é também possível adicionar ou retirar sensores, scanners, batentes e outros componentes, aumentar ou diminuir o comprimento dos transportadores e alterar a sua solução construtiva para correntes ou rolos.



Figura 2.5: Biblioteca de blocos no Visio (*documentação interna Aberle*)

No Visio, cada retângulo que se encontra numerado indica uma posição (figura 2.6). As posições, ao longo de todo o sistema, são os locais onde uma paleta pode parar, onde existe uma interface com outro equipamento ou onde existe um drive, e a maior parte destas posições está acompanhada de uma fotocélula. Todos os transportadores que são inseridos no layout têm pelo menos uma posição. O caminho que cada paleta deve ou pode seguir quando se encontra numa posição é representado por uma seta, que liga uma posição a outra.

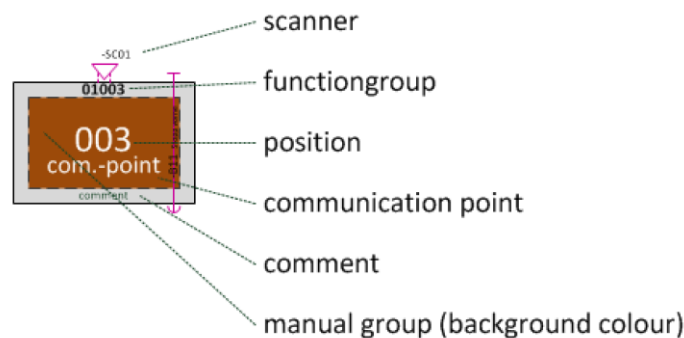


Figura 2.6: Legenda dos elementos existentes numa posição

As funções de cada componente existente numa posição são:

- *Scanner*: indica a existência de um leitor de barras ou outro tipo de scanner que dá conhecimento ao sistema do conteúdo existente numa determinada paleta;
- *Function group*: é um número de 5 dígitos onde os primeiros 2 são o *unit group* (número que vai determinar os blocos a serem exportados) e os 3 restantes são a posição do bloco. Este número serve para associar outros equipamentos a esta posição, como botões e sensores;
- *Position*: número da posição no layout;
- *Communication point*: este campo encontra-se preenchido quando numa determinada posição existe troca de informação com outro software ou PLC, como por exemplo o MFC;
- *Comment*: campo para deixar um documentário que, por definição, preenche-se com o nome do transportador (por exemplo RCV 10.0103);
- *Manual group*: campo a preencher com um número entre 1 e 50 que indica a alimentação dos transportadores, que visualmente se traduz em 50 cores diferentes. Posições/transportadores com o mesmo *manual group* têm a mesma corrente de alimentação.

Na figura 2.7 temos parte de um layout de um armazém automático, configurado no Visio.

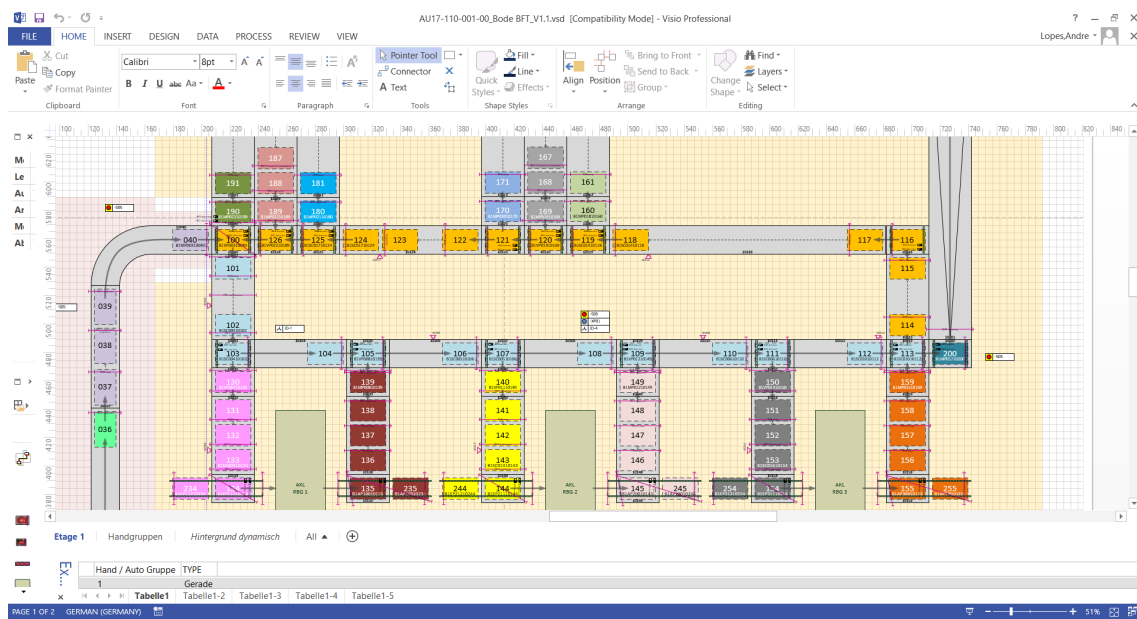


Figura 2.7: Exemplo de parte de um layout concebido no Visio

Depois do layout estar completo é feita uma exportação para um ficheiro Excel. Essa exportação é feita utilizando código VBA no software Visio, associada a botão existente no layout, que vai exportar todos os blocos que tenham o mesmo *unit number* que esse botão, e que também permite verificar se todas as conexões estão feitas e se não existem mais que uma posição com o mesmo número, validando assim o desenho. Nesse ficheiro

Excel irá estar toda a informação referente ao ficheiro Visio, em células. Cada página irá corresponder a um tipo de bloco, e o número de linhas dessa página irá ser o número de vezes que esse bloco foi usado, bem como toda a informação referente ao respetivo bloco (sensores, dimensões, soluções construtivas, etc.).

2.2.2.2 Eplan

Software utilizado no auxílio a projetos de automação, nomeadamente no planeamento, gestão e desenvolvimento dos mesmos. Além disso possibilita a elaboração de relatórios e facilita o entendimento de todos os detalhes inerentes ao projeto de automação ao longo das diferentes fases do mesmo. Resumidamente, organiza a informação e gera relatórios detalhados com base em esquemas elétricos. Possibilita ainda a troca de dados através de interface com outros software [22].

O software Eplan no standard da Körber serve para fazer a arquitetura dos esquemas elétricos de toda a instalação, e também para nomear todos os cabos de alimentação de drives. Os nomes dos símbolos e os seus endereços têm que estar bem definidos, para estarem em conformidade com os restantes ficheiros que vão servir como inputs do Engineering Editor. Depois dos ficheiros provenientes do Visio e do Eplan serem importados no Engineering Editor, este software irá associar automaticamente os símbolos do Eplan aos sensores e botões colocados no Visio, desde que a lógica de nomenclatura tenha sido a mesma, ou seja, desde que tenham os mesmos nomes. Caso contrário, como dito na subsecção 2.2.2.1, esse processo terá que ser feito manualmente.

O papel do Eplan no standard da Körber é muito semelhante ao standard atual da Consoveyo S.A., com a vantagem de, devido ao emparelhamento com o Engineering Editor, passar a haver uma dupla verificação do projeto elétrico, uma vez que estes dois software têm que estar em sintonia.

SCP23_ADDR_AtribList_20200513.sdf - Bloco de notas

Ficheiro Editar Formatar Ver Ajuda

==KLS.CS23=PS1.MC01++MC0	I	0.0	BOOL	feedback socket cabinet
==KLS.CS23=PS1.MC01++MC0	I	0.1	BOOL	feedback 400V AC energy meter
==KLS.CS23=PS1.MC01++MC0	I	0.2	BOOL	AS-i Power Supply CB
==KLS.CS11=PS1.UPS++UPS+	I	0.3	BOOL	Sitop fuse +SPS-F71 O.K.
I0.4	I	0.4	BOOL	Sitop fuse +SPS-F71 O.K.
I0.5	I	0.5	BOOL	Sitop fuse +SPS-F72 O.K.
==KLS.CS23=CCV23.0305++M	I	0.6	BOOL	Reset Button OB3
	I	0.7	BOOL	
==KLS.CS23=PD1.MC01++MC0	I	1.0	BOOL	Three Phase Supply 400V AC Drives Area 2101
==KLS.CS23=PD1.MC01++MC0	I	1.1	BOOL	Three Phase Supply 400V AC Drives Area 2102
==KLS.CS23=PD1.MC01++MC0	I	1.2	BOOL	Three Phase Supply 400V AC Drives Area 2103
==KLS.CS23=PD1.MC01++MC0	I	1.3	BOOL	Three Phase Supply 400V AC Drives Area 2104
I1.4	I	1.4	BOOL	Operationg State Battery On
I1.5	I	1.5	BOOL	UPS Alarm
I1.6	I	1.6	BOOL	UPS < 85%
==KLS.CS23=RCV23.0109++M	I	1.7	BOOL	Reset Button OB2
SCP23-Temp_OK	I	20.0	BOOL	Cabinet temperature O.K.
==KLS.CS23=PLC.DOOR++MC0	I	20.1	BOOL	System Start/Restart
I20.2	I	20.2	BOOL	Spare
==KLS.CS23=PLC.DOOR++MC0	I	20.3	BOOL	Control Unit Off
I20.4	I	20.4	BOOL	Spare
I20.5	I	20.5	BOOL	Spare
I20.6	I	20.6	BOOL	Spare
==KLS.CS23=PLC.MC01++MC0	I	20.7	BOOL	ASi network 1 CB OK
==KLS.CS23=PLC.MC01++MC0	I	21.0	BOOL	ASi network 2 CB OK
==KLS.CS11=PS1.UPS++UPS+	I	21.1	BOOL	Sitop fuse +SPS-F71 O.K.
==KLS.CS23=EXT+++Kxx:x	I	21.2	BOOL	Manual - Move CW
==KLS.CS23=EXT+++Kxx:x	I	21.3	BOOL	Manual - Move CCW
==KLS.CS23=EXT+++Kxx:x	I	21.4	BOOL	Manual - Move Low Speed

Figura 2.8: Ficheiro Eplan (.sdf), que vai ser inserido no Engineering Editor, visto em modo de texto

Na figura 2.8 temos o ficheiro de texto, exportado do Eplan, que irá posteriormente servir como *input* do Engineering Editor.

2.2.2.3 TIA-System Constants

O TIA-System Constants diz respeito às constantes globais, automaticamente criadas no TIA Portal, que são necessárias ao funcionamento de um programa. Estas componentes servem, por exemplo, para identificar objetos de hardware num projeto de automação.

Na standardização do grupo Körber, usam o TIA Portal para a criação e identificação de todo o hardware que irá ser usado na instalação, como drives, PLCs, HMIs e outros equipamentos. Depois de todo esse hardware ter sido gerado e identificado, é exportado um ficheiro de texto (com o aspeto do da figura 2.11) que vai ser inserido no Engineering Editor. A definição do hardware no TIA Portal tem que estar em conformidade com os restantes ficheiros que também vão servir de input, para uma posterior correta geração de código.

O hardware que costuma ser utilizado em todos os armazéns automáticos são PLCs, HMIs e drives, para comandar os transportadores e todas as suas variações. O primeiro passo na configuração do hardware no TIA Portal é saber que equipamentos vão ser utilizados. Essa definição do hardware é feita, na primeira fase do projeto, depois do primeiro contacto com o cliente, quando a empresa está a estudar as soluções construtivas para o armazém. Sabendo que cada transportador possui um drive, consegue-se já ter uma ideia de quais são os equipamentos necessários para o funcionamento de uma instalação. Portanto, a lista de hardware a utilizar (figura 2.9) é feita numa fase inicial, sendo atualizada à medida que se avança no projeto, caso seja necessário.

Level	Type	Description	Panel	Power line	Sequence	N°	ID	N° Pallet	Pallet type	Weight / Pa	Speed (m/s)	Time	Selected Po	Current	Index	Zone	Contactor	Power Circ	Switch Circ	Resistor	Endereço	PLC Byte	Current (A) without LTB	Current (A) with tipo X
10	RCV	Roller conveyor	23	01	01	23.0101	RCV23.0101	1	EUR/OWP	1150	0.3	2.00	0.37	1.30	30	2301	2301	2301		2301	1		1.3	0
11	RCV	Roller conveyor	23	01	02	23.0102	RCV23.0102	1	EUR/OWP	1150	0.3	0.00	0.37	1.30	2	2301	2301	2301		2301	2		1.3	0
12	RTB	Rotating table	23	01	03	23.0103	RTB23.0103	1	EUR/OWP	1000	—	0.00	0.37	1.30	1	2301	2301	2301		2301	3		1.3	0
13	RCV	Roller conveyor	23	01	04	23.0104	RCV23.0104	2	EUR/OWP	1000	0.3	10.00	0.55	1.60	27	2301	2301	2301		2301	4		1.6	0
14	RCV	Roller conveyor	23	01	05	23.0105	RCV23.0105	2	EUR/OWP	1000	0.3	10.00	0.55	1.60	26	2301	2301	2301		2301	5		1.6	0
15	RCV	Roller conveyor	23	01	06	23.0106	RCV23.0106	2	EUR/OWP	1000	0.3	10.00	0.55	1.60	25	2301	2301	2301		2301	6		1.6	0
16	RCV	Roller conveyor	23	01	07	23.0107	RCV23.0107	2	EUR/OWP	1000	0.3	10.00	0.55	1.60	24	2301	2301	2301		2301	7		1.6	0
17	RCV	Roller conveyor	23	01	08	23.0108	RCV23.0108	2	EUR/OWP	1000	0.3	10.00	0.55	1.60	23	2301	2301	2301		2301	8		1.6	0
18	RCV	Roller conveyor	23	01	09	23.0109	RCV23.0109	2	EUR/OWP	1000	0.3	10.00	0.55	1.60	22	2301	2301	2301		2301	9		1.6	0
19	RCV	Roller conveyor	23	02	01	23.0201	RCV23.0201	2	EUR/OWP	1000	0.3	10.00	0.55	1.60	21	2301	2301	2302		2301	10		0	0
20	RCV	Roller conveyor	23	02	02	23.0202	RCV23.0202	1	EUR/OWP	1000	—	0.00	0.37	1.30	4	2302	2302	2302		2301	11		0	0
21	LTB	Lifting table	23	02	03	23.0203	LTB23.0203	1	EUR/OWP	1000	—	0.00	0.37	1.30	5	2302	2302	2302		2301	12		0	0
22	CCV	Chain conveyor	23	02	04	23.0204	CCV23.0204	1	EUR/OWP	1000	0.25	9.60	0.55	1.60	3	2302	2302	2302		2301	13		0	0
23	RCV	Roller conveyor	23	02	05	23.0205	RCV23.0205	1	EUR/OWP	1000	0.3	5.00	0.37	1.30	19	2302	2302	2302		2301	14		0	0
24	RCV	Roller conveyor	23	02	06	23.0206	RCV23.0206	1	EUR/OWP	1000	—	0.00	0.37	1.30	6	2302	2302	2302		2301	15		0	0
25	LTB	Lifting table	23	02	07	23.0207	LTB23.0207	1	EUR/OWP	1000	—	0.00	0.37	1.30	7	2302	2302	2302		2301	16		0	0
26	CCV	Chain conveyor	23	02	08	23.0208	CCV23.0208	1	EUR/OWP	1000	0.25	9.60	0.55	1.60	8	2302	2302	2302		2301	17		0	0
27	CCV	Chain conveyor	23	03	02	23.0302	CCV23.0302	2	EUR/OWP	1000	0.25	11.20	1.10	2.40	14	2303	2303	2303		2303	1		0	0
28	CCV	Chain conveyor	23	03	03	23.0303	CCV23.0303	2	EUR/OWP	1000	0.25	11.20	1.10	2.40	15	2303	2303	2303		2303	2		0	0
29	CCV	Chain conveyor	23	03	04	23.0304	CCV23.0304	2	EUR/OWP	1000	0.25	11.20	1.10	2.40	16	2303	2303	2303		2303	3		0	0

Figura 2.9: Lista de equipamentos a serem utilizados numa instalação

Posteriormente, já com a lista de equipamentos, é feita a configuração do hardware no TIA Portal. Para isso é necessário importar para o software os ficheiros GSD de cada equipamento. Um ficheiro GSD (*Generic Station Description*) é um ficheiro de texto ASCII que funciona como um "bilhete de identidade" de cada equipamento. Possui as funcionalidades gerais e específicas de comunicação do equipamento, bem como a identificação de cada dispositivo, os parâmetros ajustáveis, os tipos de variáveis desses parâmetros e os seus valores limite. Estes ficheiros podem ser encontrados no site do fabricante de cada equipamento, ou, caso o equipamento seja da Siemens, o seu ficheiro GSD já se encontra

no catálogo do TIA Portal.

Depois de cada equipamento já ter o seu ficheiro GSD associado no TIA Portal é necessária fazer pertencer cada GSD a uma rede, ao atribuir a cada GSD um endereço na rede. Essa rede costuma ser Profibus ou Profinet e as suas características também precisam de ser configuradas. Depois o PLC, que também está ligado à mesma rede, consegue encontrar o GSD na rede Profinet e atribuir um nome ou identificador ao mesmo. No PLC é necessário reservar uma área de leitura e outra de escrita para este ser capaz de comunicar com o equipamento identificado pelo GSD. Depois das áreas serem criadas, são definidos os inputs (informação recebida pelo PLC) e outputs (informação enviada pelo PLC).

Quando tudo estiver bem definido e conectado, as ligações no TIA Portal vão ter um aspeto parecido ao da figura 2.10.

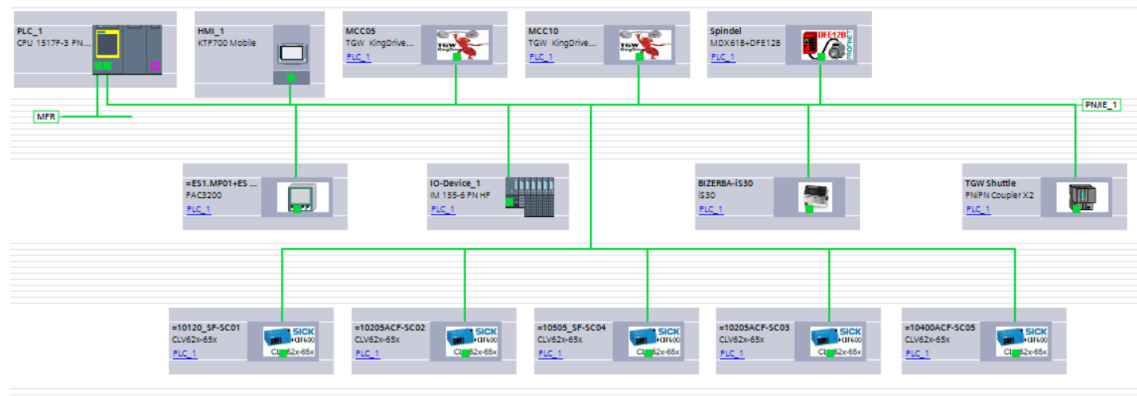


Figura 2.10: Árvore de hardware e ligações no TIA

Chegando à fase final, é feita a exportação do trabalho para um ficheiro de texto, como representado na figura 2.11, que irá ser importado para o Engineering Editor, em conjunto com os restantes ficheiros. A importação para o EE é feita ao copiar o texto referente à exportação do TIA Portal para um *clipboard*.

TIA Constants.txt - Blocco de notas

Ficheiro

Editar

Formatar

Ver

Ajuda

Keiner

Pip

65535

Automatische Aktualisierung

Pip

0

TPA 1

Pip

1

TPA 2

Pip

2

TPA 3

Pip

3

TPA 4

Pip

4

TPA 5

Pip

5

TPA 6

Pip

6

TPA 7

Pip

7

TPA 8

Pip

8

TPA 9

Pip

9

TPA 10

Pip

10

TPA 11

Pip

11

TPA 12

Pip

12

TPA 13

Pip

13

TPA 14

Pip

14

TPA 15

Pip

15

TPA 16

Pip

16

TPA 17

Pip

17

TPA 18

Pip

18

TPA 19

Pip

19

TPA 20

Pip

20

TPA 21

Pip

21

TPA 22

Pip

22

TPA 23

Pip

23

TPA 24

Pip

24

TPA 25

Pip

25

TPA 26

Pip

26

TPA 27

Pip

27

TPA 28

Pip

28

TPA 29

Pip

29

TPA 30

Pip

30

TPA 31

Pip

31

TPA OB Servo

Pip

32768

Local-MC

Hw_SubModule

51

Local-Display

Hw_SubModule

54

Local-Exec

Hw_SubModule

52

Local Hw_SubModule

49

Local-FExec

Hw_SubModule

55

Local-DP-Schnittstelle_1

Hw_Interface

60

Local-Profinet~Port_1

Hw_Interface

65

Local-Profinet~Port_2

Hw_Interface

66

Local-MFR_LP~Port_1

Hw_Interface

73

TIA Constants.txt - Blocco de notas

Ficheiro

Editar

Formatar

Ver

Ajuda

Local~PROFINET_IO_System

Hw_IoSystem

257

OB_Main_1 OB_PCYCLE

123

OB_PullPlug OB_Any 83

OB_IOWarningError OB_Any 122

OB_Diagnostic error interrupt OB_DIAG 82

OB_RACK_FLT OB_Any 86

OB_Startup OB_STARTUP 100

OB_Main OB_PCYCLE 1

Local-Common Hw_SubModule 50

Local-Device Hw_Device 32

Local~Configuration Hw_SubModule 33

Local~Profinet Hw_Interface 64

Local~MFR_LP Hw_Interface 72

Local~MFR_HP Hw_Interface 120

Local~MFR_HP~Port_1 Hw_Interface 121

=31028ACF-SC01~Proxy Hw_SubModule 258

=31028ACF-SC01~IOdevice Hw_Device 263

=31028ACF-SC01~Interface Hw_Interface 259

=31028ACF-SC01~Interface~Port_1 Hw_Interface 260

=31028ACF-SC01~Interface~Port_2 Hw_Interface 261

=31028ACF-SC01~Head Hw_SubModule 262

=31028ACF-SC01~Ctrl_Bits_in Hw_SubModule 264

=31028ACF-SC01~Ctrl_Bits_out Hw_SubModule 266

=31028ACF-SC01~_32_Byte_Input Hw_SubModule 268

=31028ACF-SC01~__8_Byte_Output Hw_SubModule 267

=31028ACF-SC01~01_Start_Remote_Config_>>>>>>> Hw_SubModule 269

=31028ACF-SC01~00_Object_Trigger_Ctrl Hw_SubModule 270

=31028ACF-SC01~11_Reading_Config(CLV62x-64x) Hw_SubModule 271

=31028ACF-SC01~23_2_5_Interleaved Hw_SubModule 272

=31028ACF-SC01~40_Data_Processing Hw_SubModule 273

=31028ACF-SC01~49_Digital_Outputs Hw_SubModule 274

=31028ACF-SC01~99_End_Remote_Config<<<<<<<< Hw_SubModule 275

=31191_SF-MCC_005~Proxy Hw_SubModule 278

=31191_SF-MCC_005~IOdevice Hw_Device 276

=31191_SF-MCC_005~PN-IO Hw_Interface 279

=31191_SF-MCC_005~PN-IO~Port_1_-RJ45 Hw_Interface 280

=31191_SF-MCC_005~PN-IO~Port_2_-RJ45 Hw_Interface 281

=31191_SF-MCC_005~Head Hw_SubModule 282

=31191_SF-MCC_005~1_System_Daten Hw_SubModule 283

=31191_SF-MCC_005~31185_SF Hw_SubModule 284

=31191_SF-MCC_005~31186_SF Hw_SubModule 285

=31191_SF-MCC_005~31187_SF Hw_SubModule 286

=31191_SF-MCC_005~31188_SF Hw_SubModule 287

Figura 2.11: Exemplo de ficheiro de texto com informação do hardware, gerado do TIA Portal

2.2.3 Outputs

Iremos agora abordar os ficheiros que são gerados depois da configuração do Engineering Editor estar completa, como funcionam e o seu papel na programação de um armazém automático.

2.2.3.1 Etiquetas de Identificação

No que diz respeito às etiquetas de identificação, não é gerado nenhum documento. O Engineering Editor é conectado diretamente à etiquetadora, através de uma ligação TCP/IP, e transmite para a máquina toda a informação necessária para que esta imprima as etiquetas destinadas aos equipamentos.

2.2.3.2 SCADA

Numa instalação de grande dimensão e tão complexa como um armazém automático, é necessário ter acesso ao estado de todos os equipamentos existentes, de uma maneira fácil e intuitiva. Para isso são utilizados sistemas SCADA (*Supervisory Control and Data Acquisition*). Nos armazéns automáticos, estes sistemas acedem ao PLC para retirar o estado e valores de variáveis específicas, para os poderem disponibilizar ao utilizador juntamente com o layout do sistema de transportadores para uma rápida recolha de informação relativa a algum equipamento.

Como foi dito em 2.2.1, um dos ficheiros gerados pelo Engineering Editor é um ficheiro Excel com a toda a informação referente à localização de transportadores, sensores e outros equipamentos. Este ficheiro vai ser depois importado, recorrendo a código VBA em Excel, para um software de supervisão e aquisição de dados. O papel deste ficheiro é "indicar" ao software SCADA a localização, no PLC, das variáveis que este precisa de ter acesso, tanto sobre a posição e estados de equipamentos como de mensagens de erro, e formar o layout da disponibilização dos transportadores.

2.2.3.3 TIA Portal

Para a correta geração do código de controlo de todo o sistema periférico, são gerados vários tipos de blocos no TIA Portal.

Function Blocks

O código de controlo dos transportadores é um dos ficheiros gerados, conforme referido em 2.2.1. Porém, não na sua totalidade, devido às situações complexas anteriormente referidas. Do Engineering Editor é gerado um ficheiro XML que posteriormente irá ser importado no TIA Portal, através da API Demo Openness.

Já no EE são definidas, automaticamente e manualmente, as funções a utilizar no programa, e fica tudo armazenado num ficheiro XML. Esse ficheiro XML irá "chamar" blocos já existentes e testados para cada situação (figura 2.12), que se encontram no TIA Portal como *function blocks*, onde é apenas necessário atribuir *inputs* e *outputs* a essas funções, processo também feito automaticamente. Resumindo, quando o ficheiro XML é importado para o TIA Portal, os *inputs* e *outputs* já estão atribuídos.

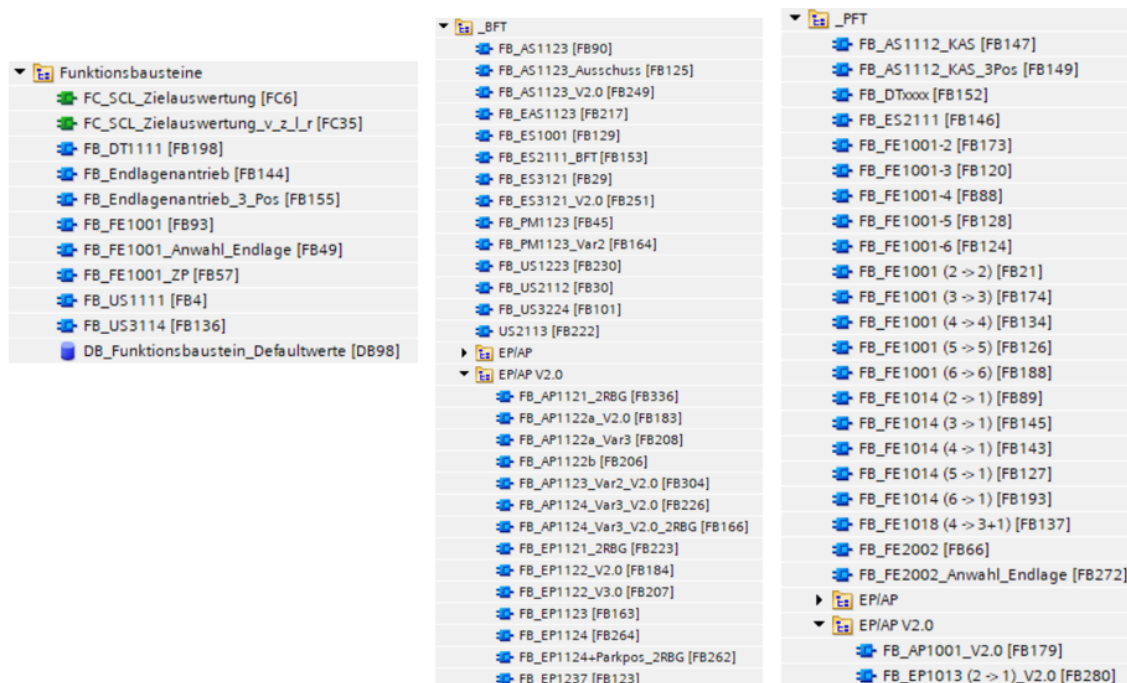


Figura 2.12: Funções já existentes de controlo de transportadores

Contudo, em todas as obras existem situações que não são abrangidas pelos blocos já existentes, e portanto, é necessário conceber programação específica, manualmente. Para esse tipo de programação também é possível utilizar os FBs já existentes. Quando esses FBs são chamados pelo EE, o programa preenche automaticamente os *inputs* e *outputs* principais do mesmo. Porém existem variáveis secundárias que o EE não preenche por não serem necessárias no funcionamento standard desse bloco.

O programador pode usar essas variáveis para customizar o bloco e elaborar a programação específica. Caso nenhum dos blocos seja apropriado para um determinado caso, a programação terá que ser feita "do zero", manualmente.

As funções existentes na figura 2.12 estão programadas para a interação entre posições. Das funções existentes, as mais usadas são:

- Funções de transferência de paletes entre posições (de 1 posição para 1, ou de 2 posições para outras 2);
- Funções de acumulação e desacumulação de paletes (de 1 posição para 2 ou mais e vice-versa);
- Funções de transferência de paletes entre posições com mesas elevatórias;
- Funções de interface de entrada e saída de transelevadores;
- Funções de gestão da posição de mesas elevatórias.

A nomenclatura da parte numérica das funções foi feita seguindo o método da figura 2.13. A parte alfabética foi feita usando iniciais de palavras alemãs (nacionalidade da empresa Aberle).

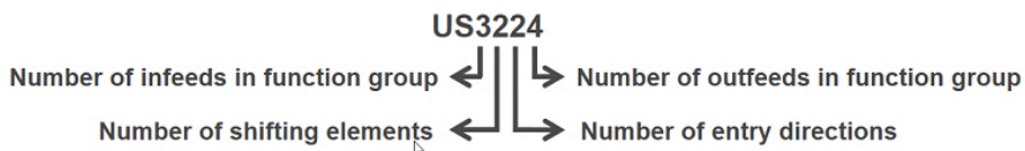


Figura 2.13: Nomenclatura blocos de funções

Usando um exemplo para explicar o que o Engineering Editor faz, consideremos que no layout em Visio temos uma configuração de uma transferência de paletes de uma posição para outra, como a da figura 2.14.

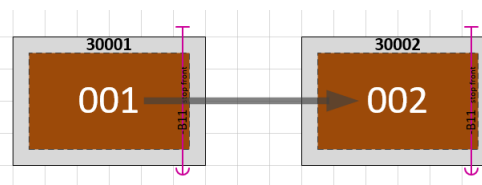


Figura 2.14: Exemplo de uma configuração de 1 posição para outra

Quando o programador fizer a exportação do layout Visio para um ficheiro Excel, e inserir esse ficheiro no Engineering Editor, o software vai automaticamente reconhecer a

transferência da figura 2.14, e irá sugerir ao programador a utilização do bloco FE1001 (figura 2.15), que no TIA Portal é a função onde se encontra programada este tipo de transferências. A direção do movimento de paletes é fornecida pelo conector dinâmico existente no Visio, que permite ligar dois objetos, sendo um a origem e outro o destino.

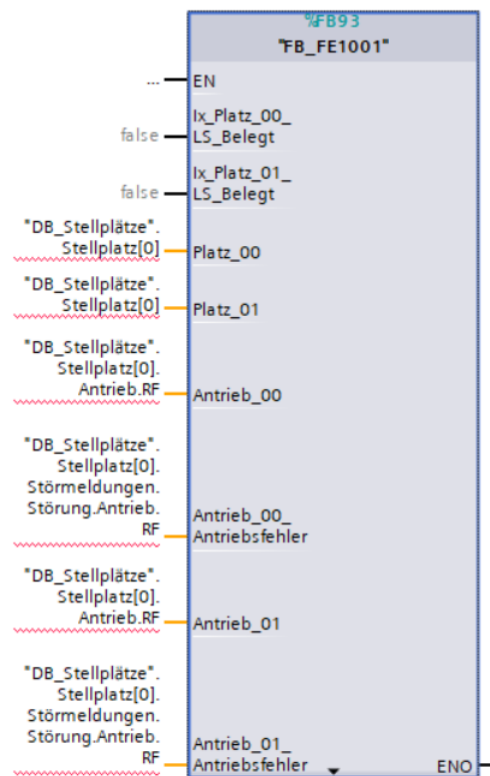


Figura 2.15: *Function block* FE1001 no TIA Portal

Nesse bloco, os inputs referentes ao número das posições, aos tipos de drives de cada uma e aos erros já vão estar preenchidos com as respectivas referências às posições 001 e 002. Essas referências vão estar localizadas numa *data block* de nome DB_Positions, onde se encontra toda a informação relativa às posições.

Data Blocks e User Data Types

É necessário inserir vários inputs nas funções abordadas em 2.2.3.3, por isso ao mesmo tempo que as funções são geradas, automaticamente, são também gerados *data blocks* e *user data types* com toda a informação relativa ao layout, como posições, tipos de drives, erros, informação que vai ser necessária atribuir às funções. Entre todas essas DB's criadas, algumas são importantes abordar.

Posições (DB_Positions)

É criado uma DB referente às posições. Nela encontra-se um array que chama uma UDT onde se encontra a configuração de todas as posições existentes no layout do Visio. Dentro de cada posição existe ainda outro array com várias configurações, como podemos ver na figura 2.16. Cada *audt_Position* pode integrar diferentes equipamentos como transportadores de rolos ou correntes, elevador, mesa rotativa, etc. e estes são tratados dentro das UDTs respectivas, associadas a uma posição. Isto permite que o controle e

tratamento de dados esteja "encapsulado" dentro de uma DB. Depois do código referente às posições ser devidamente testado, pode ser utilizado sem grande probabilidade de erro, ou a necessidade de alterações *on-site*.

DB_Position								
	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint
1	Static			☐	☐	☐	☐	☐
2	x_Init	Bool	false	☐	☒	☒	☒	☐
3	audt_Position	Array[0..*GCI_Quan...		☒	☒	☒	☒	☒
4	audt_Position[0]	*udt_Position*		☒	☒	☒	☒	☒
5	ui_Index	UInt	0	☒	☒	☒	☒	☐
6	wstr_GroupOff...	WString[10]	WSTRING#"	☒	☒	☒	☒	☐
7	usi_Group	USInt	0	☒	☒	☒	☒	☐
8	udt_State	*udt_PositionState*		☒	☒	☒	☒	☒
9	udt_DriveFlag	*udt_DriveFlag*		☒	☒	☒	☒	☒
10	udt_Target	*udt_PositionTarget*		☒	☒	☒	☒	☒
11	udt_Drive	*udt_DriveTypes*		☒	☒	☒	☒	☒
12	udt_Data	*udt_PositionData*		☒	☒	☒	☒	☒
13	udt_ErrorMessa...	*udt_ErrorMessage...		☒	☒	☒	☒	☒
14	udt_Performanc...	*udt_PerformanceA...		☒	☒	☒	☒	☒
15	udt_Configurat...	*udt_PositionConfi...		☒	☒	☒	☒	☒
16	udt_Distribution	*udt_PositionDistri...		☒	☒	☒	☒	☒
17	udt_InterfaceC...	*udt_Position_IF*		☒	☒	☒	☒	☒
18	udt_Configured	*udt_PositionConfi...		☒	☒	☒	☒	☒
19	udt_IF_HMI_O...	*udt_Position_IF_H...		☒	☒	☒	☒	☒
20	udt_Acknowle...	*udt_PositionAckn...		☒	☒	☒	☒	☒
21	audt_Position[1]	*udt_Position*		☒	☒	☒	☒	☒

Figura 2.16: DB.Position vista no TIA Portal

As várias UDT's dentro de cada posição dizem respeito a várias configurações, sendo as principais:

- **udt_State**: indica se uma posição está livre, ocupada, em erro, em modo manual ou automático ou pronta para receber ou enviar uma paleta;
- **udt_DriveFlag**: indica se uma posição está pronta para receber ou enviar uma paleta para outra posição, e indica ainda a localização da posição com que vai interagir (direita, esquerda, atrás ou à frente);
- **udt_Target**: indica como vai ser escolhido o destino da paleta que se encontra nessa posição, se vai ser o destino padrão ou um destino especificado pelo MFC;
- **udt_Drive**: indica o tipo de drive que controla o transportador relativo à posição;
- **udt_Data**: é onde se armazena a informação digital que acompanha uma paleta, quando esta chega a uma posição;
- **udt_ErrorMessage**: é onde se encontram configuradas as mensagens de erro;
- **udt_Distribution**: no caso de uma posição poder enviar paletes para várias, indica para que posição, relativa à posição de origem, a paleta é enviada;
- **udt_Acknowledge**: especificação do tipo de reconhecimento da existência de uma paleta na posição, reconhecimento físico, pelo o WinCC ou ambos.

A UDT_DriveFlag é especialmente importante porque é onde se encontram as variáveis diretamente responsáveis pela passagem de paletes entre posições. Esta UDT é composta por 20 variáveis, representado na figura 2.17.

udt_DriveFlag							
	Name	Data type	Default value	Accessible f...	Writa...	Visible in ...	Setpoint
1	x_ForwardDischarge	Bool	false	☒	☒	☒	☐
2	x_Forward	Bool	false	☒	☒	☒	☐
3	x_BackwardDischarge	Bool	false	☒	☒	☒	☐
4	x_Backward	Bool	false	☒	☒	☒	☐
5	x_LeftDischarge	Bool	false	☒	☒	☒	☐
6	x_Left	Bool	false	☒	☒	☒	☐
7	x_RightDischarge	Bool	false	☒	☒	☒	☐
8	x_Right	Bool	false	☒	☒	☒	☐
9	x_Lift	Bool	false	☒	☒	☒	☐
10	x_Lower	Bool	false	☒	☒	☒	☐
11	x_Open	Bool	false	☒	☒	☒	☐
12	x_Close	Bool	false	☒	☒	☒	☐
13	x_TurnPositive	Bool	false	☒	☒	☒	☐
14	x_TurnNegative	Bool	false	☒	☒	☒	☐
15	x_DriveFlag1	Bool	false	☒	☒	☒	☐
16	x_DriveFlag2	Bool	false	☒	☒	☒	☐
17	x_DriveFlag3	Bool	false	☒	☒	☒	☐
18	x_DriveFlag4	Bool	false	☒	☒	☒	☐
19	x_TakeOver	Bool	false	☒	☒	☒	☐
20	x_RequestDriveFlagReset	Bool	false	☒	☒	☒	☐

Figura 2.17: UDT_DriveFlag vista no TIA Portal

Significado das variáveis principais:

1. *x_Forward*, *x_Backward*, *x_Left* e *x_Right*: é ativada quando as condições necessárias para a passagem de uma paleta de uma posição para outra se verificam;
2. Várias *...Discharges*: indicam quando um movimento foi realizado, ou seja, esta variável é ativada depois de uma paleta chegar à posição de destino;
3. *x_TakeOver*: ativada, na posição de destino, quando a passagem da paleta para essa posição se inicia. Isto impeça que haja duas posições a passarem uma paleta para a mesma posição, ao mesmo tempo;
4. Variáveis *x_DriveFlag* de 1 a 4: usadas para passagens especiais.

Na figura 2.18 temos gráficos temporais destas variáveis para perceber melhor o estado de cada durante a passagem de paletes.

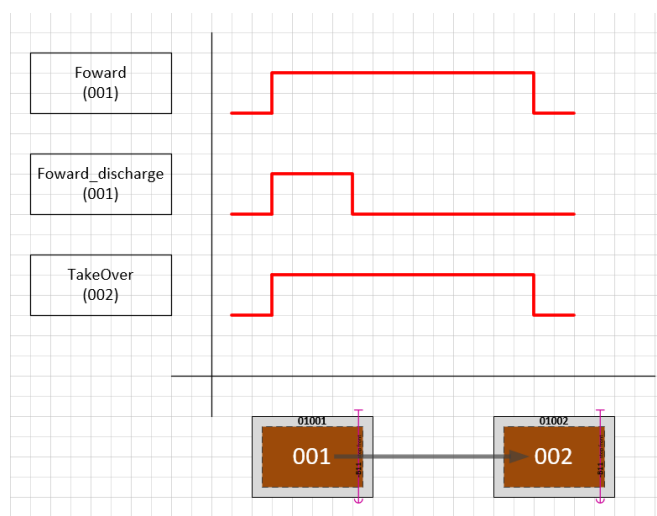
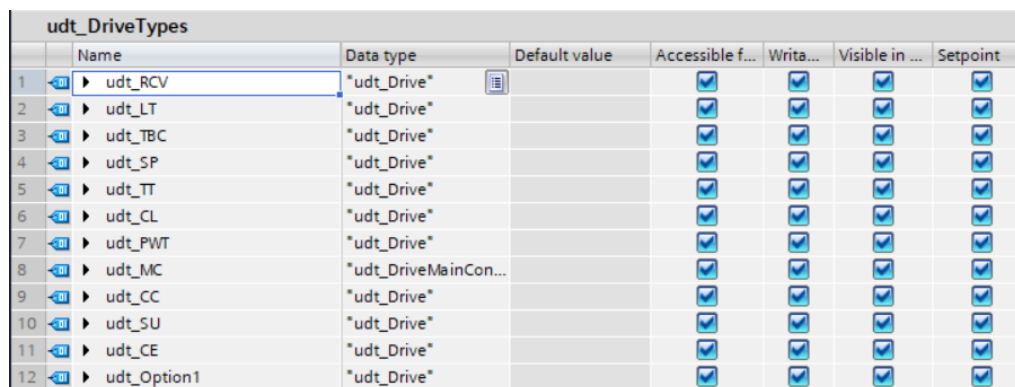


Figura 2.18: Gráficos temporais das variáveis *Forward*, *Forward_Discharge* e *TakeOver* durante a passagem de uma paleta

Drives (UDT_DriveType)

Na UDT_DriveTypes (figura 2.19) encontram-se armazenados os parâmetros dos drives de cada tipo de equipamento (transportador de rolos, de correntes, mesa rotativa, elevador, etc.). Na configuração das funções é sempre pedido qual a configuração de drive a usar, dependendo do tipo de equipamento.



	Name	Data type	Default value	Accessible f...	Writa...	Visible in ...	Setpoint
1	udt_RCV	*udt_Drive*		☒	☒	☒	☒
2	udt_LT	*udt_Drive*		☒	☒	☒	☒
3	udt_TBC	*udt_Drive*		☒	☒	☒	☒
4	udt_SP	*udt_Drive*		☒	☒	☒	☒
5	udt_TT	*udt_Drive*		☒	☒	☒	☒
6	udt_CL	*udt_Drive*		☒	☒	☒	☒
7	udt_PWT	*udt_Drive*		☒	☒	☒	☒
8	udt_MC	*udt_DriveMainCon...		☒	☒	☒	☒
9	udt_CC	*udt_Drive*		☒	☒	☒	☒
10	udt_SU	*udt_Drive*		☒	☒	☒	☒
11	udt_CE	*udt_Drive*		☒	☒	☒	☒
12	udt_Option1	*udt_Drive*		☒	☒	☒	☒

Figura 2.19: UDT_DriveTypes vista no TIA Portal

Uma análise mais profunda mostra que dentro de cada variável correspondente aos tipos de equipamentos temos uma configuração mais detalhada, recorrendo a uma outra UDT, de nome UDT_Drive (figura 2.20). De um modo geral, esta UDT tem a função de fazer a ligação entre funções e os drives. Para isso, são chamadas outras UDTs dentro desta, cada uma com a sua função:

- udt_DriveControl: especificação do modo de controlo do drive, manual ou automático, e dentro de cada um é possível escolher direções e velocidades;
- udt_DriveState: indica se o drive está a operar ou não, e especifica ainda o modo de operação, de controlo. Caso não esteja a operar, também indica a razão (por não está a ser preciso ou por emergência);
- udt_IF_Manual: quando o drive está em modo manual, possui variáveis que indicam a direção em que está a funcionar (positiva ou negativa) e se está em alta velocidade;
- udt_Limits: indica quando as posições limite de um equipamento foram atingidas, e quando o equipamento está na posição "default".

udt_Drive							
	Name	Data type	Default value	Accessible f...	Writa...	Visible in ...	Setpoint
1	udt_Control	*udt_DriveContr...		☒	☒	☒	☒
2	s_Automatic	Struct		☒	☒	☒	☐
3	s_Manual	Struct		☒	☒	☒	☐
4	udt_State	*udt_DriveState*		☒	☒	☒	☒
5	x_Used	Bool	false	☒	☒	☒	☐
6	x_Running	Bool	false	☒	☒	☒	☐
7	x_Disturbed	Bool	false	☒	☒	☒	☐
8	x_Manual	Bool	false	☒	☒	☒	☐
9	x_Setup	Bool	false	☒	☒	☒	☐
10	x_EmergencyStop	Bool	false	☒	☒	☒	☐
11	x_Auto	Bool	false	☒	☒	☒	☐
12	x_ManualModeReleased	Bool	false	☒	☒	☒	☐
13	x_PneumaticallyControlled	Bool	false	☒	☒	☒	☐
14	udt_ManualMode	*udt_IF_Manual*		☒	☒	☒	☒
15	x_Positive	Bool	false	☒	☒	☒	☐
16	x_Negative	Bool	false	☒	☒	☒	☐
17	x_FastSpeed	Bool	false	☒	☒	☒	☐
18	udt_OperatingData	*udt_DriveOperatin...		☒	☒	☒	☒
19	udi_OperatingSeconds	UDInt	0	☒	☒	☒	☐
20	udi_DriveFlag	UDInt	0	☒	☒	☒	☐
21	udt_Limits	*udt_DriveLimits*		☒	☒	☒	☒
22	x_DefaultPositionReached	Bool	false	☒	☐	☒	☐
23	x_Top	Bool	false	☒	☐	☒	☐
24	x_Middle	Bool	false	☒	☐	☒	☐
25	x_Bottom	Bool	false	☒	☐	☒	☐
26	x_Open	Bool	false	☒	☐	☒	☐
27	x_Closed	Bool	false	☒	☐	☒	☐

Figura 2.20: UDT_Drive vista no TIA Portal

TIA Portal Openness

O TIA Portal Openness é uma interface de programação de aplicações para o TIA Portal. Permite aos utilizadores criarem aplicações próprias de manipulação de projetos no TIA Portal, usando uma ferramenta de programação externa à Siemens. Esta interface permite realizar várias tarefas ligadas a projetos de automação, como programar blocos, HMIs, adicionar hardware, entre outras bem como a troca de informação entre o TIA Portal e outra ferramenta, através de ficheiros XML.

A Siemens disponibiliza duas aplicações de demonstração do TIA Openness, a Start Openness e Demo Openness (figura 2.21). Depois de estar tudo configurado no Engineering Editor, o ficheiro gerado é importado para o TIA Portal utilizando o TIA Openness, mais propriamente a API Demo Openness. A Siemens desenvolveu esta API com o propósito de mostrar aos utilizadores as funcionalidades presentes no TIA Openness.

No processo de programação que estamos a abordar, o Demo Openness entra na fase final do mesmo, quando, depois de o ficheiro XML ter sido gerado pelo Engineering Editor, é importado para o TIA Portal. Esta importação é feita através da funcionalidade *Import* (figura 2.21). O ficheiro gerado do EE é um ficheiro XML (*Extensible Markup Language*), formato que contém elementos e atributos que são usados para definir um objeto.

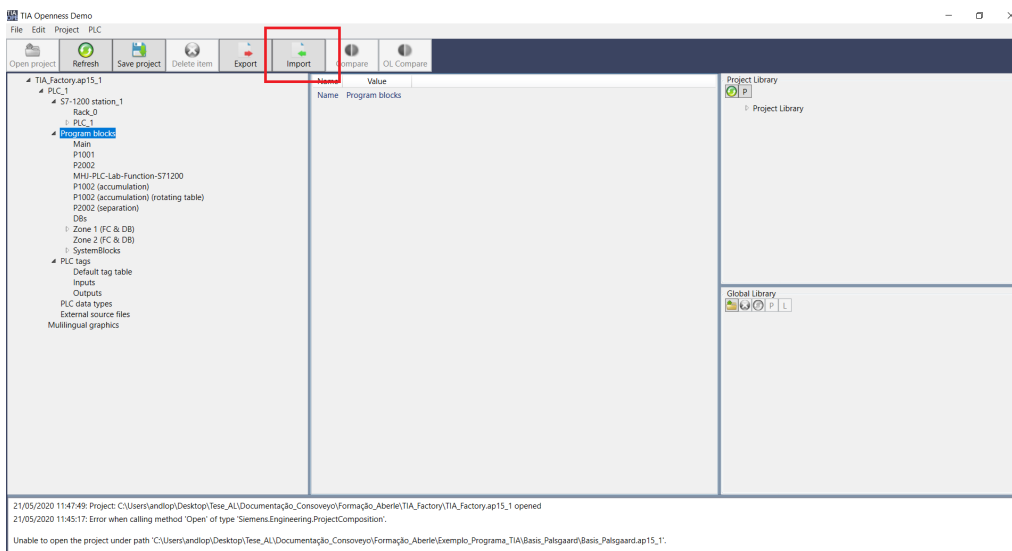


Figura 2.21: Aplicação OpennessDemo

2.2.3.4 Tabelas de Monitorização

Como foi referido em 2.2.1, o EE também gera um ficheiro Excel correspondente às tabelas de monitorização no TIA Portal (figura 2.22). Nas tabelas de monitorização encontra-se a informação sobre o estado das variáveis correspondentes aos sensores e outros dispositivos existentes no sistema. São usadas na fase de comissionamento para forçar as variáveis a tomar certos valores enquanto são monitorizadas, com o propósito de testar as soluções desenvolvidas. Depois de geradas são copiadas para o TIA Portal.

2	//-----Stellplatz 1-----Handgruppe 1 -----Funktionsgruppe 11001-----					
3	//Sensoren					
4		=11001_RF-B17		Bool		false
5		=11001_RF-B13		Bool		false
6		=11001_RF-B16		Bool		false
7	//Takte					
8		DB_Stellplätze.Stellplatz[1].Takte.vorwärts		Bool		false
9		DB_Stellplätze.Stellplatz[1].Takte.vorwärts_vorziehen		Bool		false
10		DB_Stellplätze.Stellplatz[1].Takte.EINFÖRDERN_aktiv		Bool		false
11	//Daten P1MP10011001					
12		DB_Stellplatzdaten.Datenplatz[1].Telegrammart				false
13	//Typabhängig Konturenkontrolle)					
14		DB_Stellplatzdaten.Datenplatz[1].Kontur.BYTE_01				false
15		DB_Stellplatzdaten.Datenplatz[1].Kontur.BYTE_02				false
16		DB_Stellplatzdaten.Datenplatz[1].Kontur.BYTE_03				false
17		DB_Stellplatzdaten.Datenplatz[1].Kontur.BYTE_04				false
18		DB_Stellplatzdaten.Datenplatz[1].Kontur.BYTE_05				false
19		DB_Stellplatzdaten.Datenplatz[1].Kontur.BYTE_06				false
20		DB_Stellplatzdaten.Datenplatz[1].Kontur.BYTE_07				false
21		DB_Stellplatzdaten.Datenplatz[1].Kontur.BYTE_08				false
22		DB_Stellplatzdaten.Datenplatz[1].Kontur.BYTE_09				false
23		DB_Stellplatzdaten.Datenplatz[1].Kontur.BYTE_10				false
24		DB_Stellplatzdaten.Datenplatz[1].Kontur.BYTE_11				false
25		DB_Stellplatzdaten.Datenplatz[1].Kontur.BYTE_12				false
26		DB_Stellplatzdaten.Datenplatz[1].Hoehe				false
27		DB_Stellplatzdaten.Datenplatz[1].Gewicht.Byte_01_02_03_04				false

Figura 2.22: Exemplo do um ficheiro Excel gerado correspondente às tabelas de monitorização

Com este capítulo fica concluído que a metodologia, bem como a ferramenta criada pelo grupo Körber aperfeiçoa a programação de um sistema periférico. Esta melhoria prende-se com o facto de um engenheiro não ter que programar manualmente situações rotineiras, uma vez que o código é gerado automaticamente pelo Engineering Editor, podendo-se focar nas situações diferentes do normal. Este método possibilita ainda o comissionamento e manutenção mais fáceis, uma vez que se encontra assente num standart de programação bem definido.

3

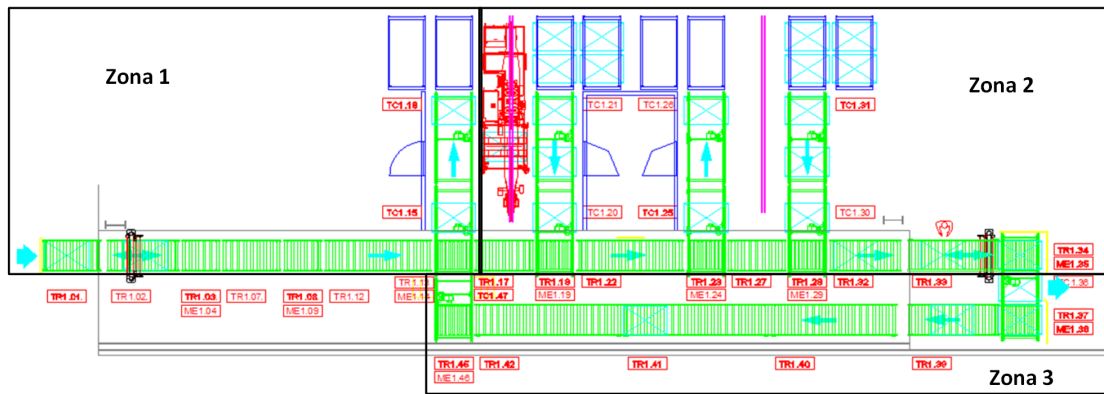
Metodologia praticada na Consoveyo S.A.

Neste capítulo iremos fazer uma breve abordagem à metodologia utilizada na Consoveyo S.A., nomeadamente as ferramentas e equipamentos utilizados, princípios de funcionamento, o processo de desenvolvimento dos sistemas automatizados e como é feita a organização das soluções desenvolvidas.

3.1 Descrição geral do sistema

Na figura 3.1 temos um exemplo simples de um layout geral desenvolvido pela Consoveyo S.A. constituídos por vários conjuntos de transportadores e transelevadores. A obra representada é mais simples, comparando com outras concebidas pela empresa, porém, é suficiente para explicar o funcionamento das soluções desenvolvidas. Portanto, na figura 3.1 temos o layout dividido em três zonas funcionais:

- Zona 1: zona que recebe as paletes que entram no circuito, sendo depois manuseadas pelo sistema de transporte e interface com os transelevadores;
- Zona 2: zona constituída pelos transportadores que manuseiam as paletes destinadas ao armazenamento onde se realiza a "alimentação" aos transelevadores;
- Zona 3: zona de circulação de paletes que provenientes do circuito ou do armazém e têm como destino a saída do circuito;



Zona 1

Esta zona de transporte, ilustrada na figura 3.2, abrange os transportadores entre o TR1.01 e o TC1.16 inclusive. O posto de controlo existente nos transportadores TR1.01 e TR1.02 tem como função receber as paletes do empilhador, no TR1.01, fazer a verificação dimensional e proceder à sua pesagem, no TR1.02. Estando estas de acordo com as normas pré-estabelecidas avançam para os transportadores que interagem com o transelevador para serem armazenadas.

A paleta é colocada no TR1.01. Depois do operador dar ordem para a movimentar o PLC comunica com o WMS (*Warehouse Management System*) e este dá a ordem ao PLC para a paleta avançar. A paleta passa então pelo TR1.02 para ser verificada e caso esteja de acordo com as dimensões avança, caso contrário recua para o TR1.01. No caso da paleta ser rejeitada o operador terá que a verificar e arranjar se for possível ou então retirá-la. Em qualquer um dos casos a decisão de rejeitar ou aceitar a paleta é feita pelo WMS. O PLC verifica as dimensões e o peso, comunica com o WMS, transmitindo essa informação, e o WMS informa o PLC do destino da paleta, que pode ser o TR1.01 se for rejeitada, ou o TC1.26 se a paleta for aceite e o destino seja a ala 2.

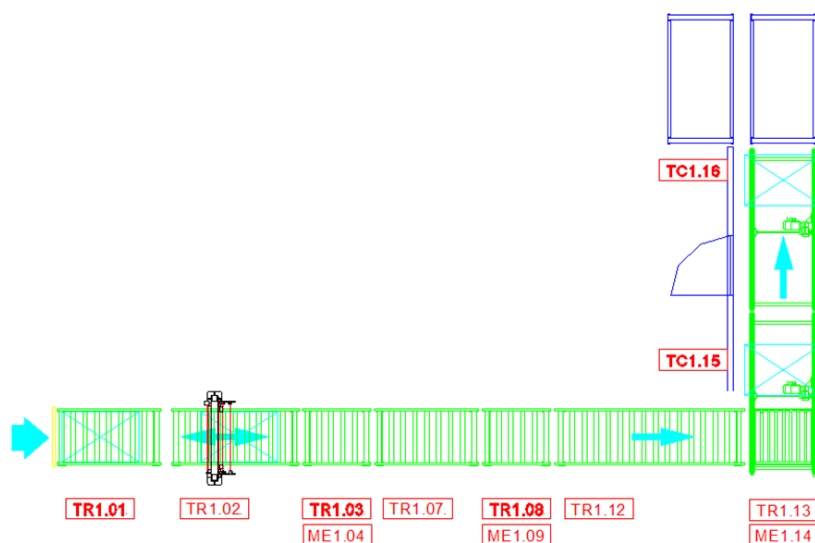


Figura 3.2: Layout - Zona 1

Zona 2

Esta zona de transporte, ilustrada na figura 3.3, abrange os transportadores entre o TR1.17 e o ME1.35 inclusive. Tem como função receber as paletes dos transelevadores das alas 1 e 2 e encaminhá-las para a saída, podendo também encaminhá-las para o transportador de entrada para o armazém na ala 1. Nesta zona existe um segundo posto de controlo que verifica tanto as dimensões como o peso das paletes.

Para cada paleta que chegue ao transportador de entrada para o armazém na ala 1, TC1.26, o WMS é informado pelo PLC da existência desta paleta, que por sua vez envia um comando ao transelevador para ir buscar a paleta e colocá-la no armazém. Quando o transelevador coloca uma paleta num transportador de saída do armazém, p.e. o TC1.31 da ala 1, o PLC informa o WMS da existência de uma paleta nesse transportador e aguarda uma comunicação de retorno do WMS, com o destino dessa paleta. Depois do WMS receber a indicação de que o transelevador colocou a paleta nesse transportador, este preenche a informação relativa ao destino dessa paleta para o PLC poder processar. As paletes que saem do armazém podem ter como destino o segundo posto de controlo de controlo no transportador TR1.33 ou o transportador de saída, o TC1.36. Junto do transportador TR1.33 existe um posto de *picking* onde o operador poderá proceder a descarga parcial da carga.

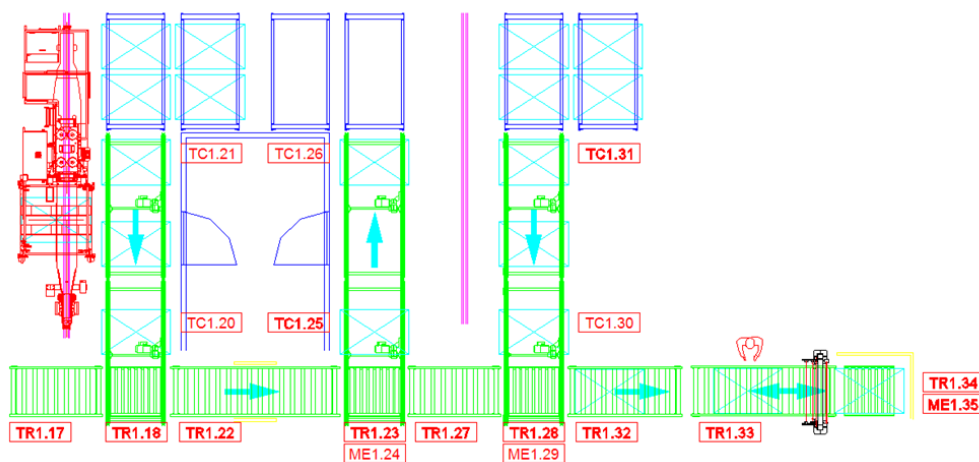


Figura 3.3: Layout - Zona 2

Zona 3

Esta zona de transporte, ilustrada na figura 3.4, abrange os transportadores entre o TC1.26 e o TC1.47 inclusive. Tem como função encaminhar as paletes para o transportador de saída, o TC1.36, fazê-las recircular no caso do destino ser um dos transportadores, de interface com os transelevadores, para entrada no armazém.

As paletes depois de passarem pelo segundo posto de controlo é-lhes atribuído um destino pelo WMS, destino esse que pode ser o transportador de saída. Neste caso as paletes com este destino vão parar no TC1.36 e serão posteriormente retiradas pelo operador utilizando um empilhador. Se as paletes tiverem como destino a entrada do armazém não param no TC1.36, continuam pelo TR1.37 e voltam a entrar na linha de desvio para a entrada no armazém depois de passarem o TC1.47.

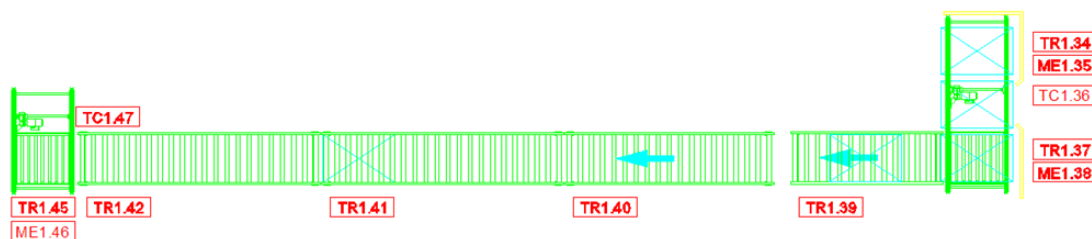


Figura 3.4: Layout - Zona 3

3.2 Princípio de funcionamento dos transportadores

Os transportadores são os equipamentos principais de transporte de paletes, e outros objetos, entre diferentes pontos de um sistema, nomeadamente armazéns, entradas e saídas do sistema. Nesta secção iremos fazer uma breve abordagem aos transportadores mais usados na empresa.

3.2.1 Composição dos transportadores

Na Consoveyo S.A. são usadas diversas variantes de transportadores dependendo do movimento que se pretende realizar, o que permite desenvolver soluções mais funcionais num armazém automático. As variantes usadas são:

- Transportador de rolos (TR);
- Transportador de correntes (TC);
- Mesa elevatória (ME);
- Mesa rotativa (MR);
- Elevador (EL);
- Batente escamoteável (BE);
- Transportador de gravidade (TG);
- Transportador virtual (TV);

Destas variantes, as mais utilizadas são os transportadores de rolos (TR) e de correntes (TC) e as mesas elevatórias (ME). Os dois tipos de transportadores referidos são os que possibilitam um movimento mais simples e direto entre dois pontos. Quanto às mesas elevatórias, são também muito usadas para a mudança de direção e elevação de cargas. Em quase todas as obras, as cargas são postas em circulação numa maior altura, por questões de segurança e de facilidade de passagem entre diferentes tipos de equipamento.

Transportadores de Correntes

Este tipo de transportador serve para movimentar cargas entre dois pontos. A carga está no topo de, pelo menos, duas vigas paralelas, as quais possuem correntes de rolamentos. Existem dois tipos de sistemas de tração de correntes: sistema de retorno automático e não automático. Na figura 3.5 encontra-se a parte interior deste tipo de transportadores, desenvolvidos na Consoveyo S.A..

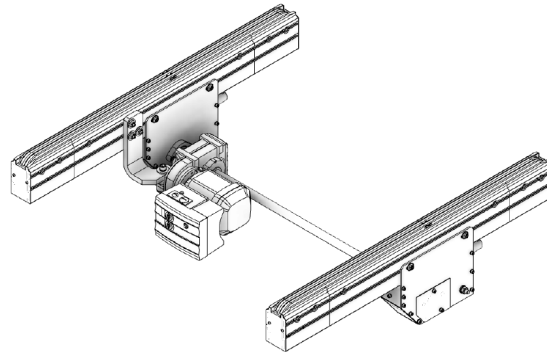


Figura 3.5: Interior de um transportador de correntes, desenvolvido pela Consoveyo S.A.

Transportadores de rolos

Tal como o transportador de correntes, o transportador de rolos, figura 3.6, tem como principal função movimentar carga entre dois pontos. A carga é colocada sobre rolos dispostos paralelamente uns com os outros. Um dos rolos é acionado por um motor, e transmite a força e movimento para os restantes rolos através do uso de um mecanismo de correntes de rolos.

A escolha entre transportadores de correntes ou de rolos prende-se em vários aspetos. Por uma questão de facilidade de centragem, preferem-se os tapetes de rolos pois estes têm golas interiores e exteriores que permitem centrar as paletes de uma maneira mais fácil. Sempre que se deseja mudar a direção de uma palete, sem se recorrer a uma mesa rotativa (o movimento é mais demorado), é necessário usar os dois tipos de transportadores, um para fazer com que a palete entre na mesa elevatória e outro para fazer com que a palete saia. Isto é necessário devido à natureza da mesa elevatória onde, por exemplo, se uma palete entrar através de rolos, a palete é depois levantada e movimentada na perpendicular recorrendo a correntes.

Noutras situações, a escolha entre transportadores de correntes ou rolos é feita consoante a natureza dos equipamentos com que estes transportadores irão interagir. No caso de alimentar/receber cargas de um elevador, é, preferencialmente, escolhido o transportador de correntes devido a uma maior estabilidade da carga.

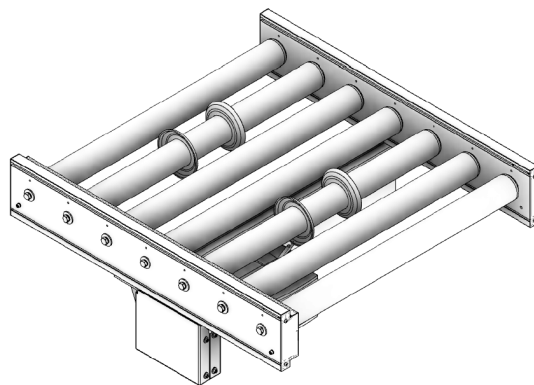


Figura 3.6: Transportador de rolos desenvolvido pela Consoveyo S.A.

Mesa Elevatória

Este equipamento é usado para fazer a elevação de cargas. Quando um transportador de rolos é aplicado no topo da mesa elevatória, é possível fazer a transferência de cargas (normalmente do transportador de rolos para o de correntes e vice-versa). A mesa elevatória pode ser usada em várias situações e posições de trabalho, usando variações da mesma (diferentes drivers, posições e potências). Este equipamento pode ainda ser usado em situações especiais e com diferentes tipos de montagens, como acoplada a uma mesa rotativa ou com suportes fixos para levantar cargas.

Na figura 3.7 temos o interior de uma mesa elevatória, desenvolvida pela Consoveyo S.A..

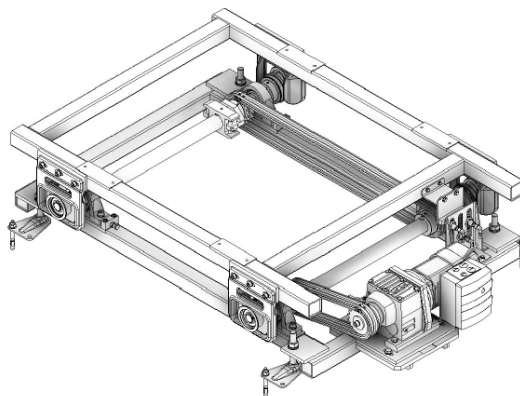


Figura 3.7: Interior de uma mesa elevatória, desenvolvida pela Consoveyo S.A.

3.2.2 Modos de funcionamento

Os transportadores, tal como a maioria dos equipamentos existentes num armazém automático, podem funcionar em dois modos: automático e manual. O modo normal de funcionamento é o automático, e todo o sistema foi projetado para funcionar corretamente neste modo, porém, em certas alturas devido a erros técnicos ou humanos, ou para testes de equipamento é necessário operar em modo manual, e estes sistemas devem contemplar essa opção.

3.2.2.1 Modo Automático

O modo normal de movimento de paletes num sistema é o automático, onde o PLC se encarrega de gerir o sistema, encaminhando as paletes para os respetivos destinos. O funcionamento neste modo segue o diagrama apresentado na figura 3.8.

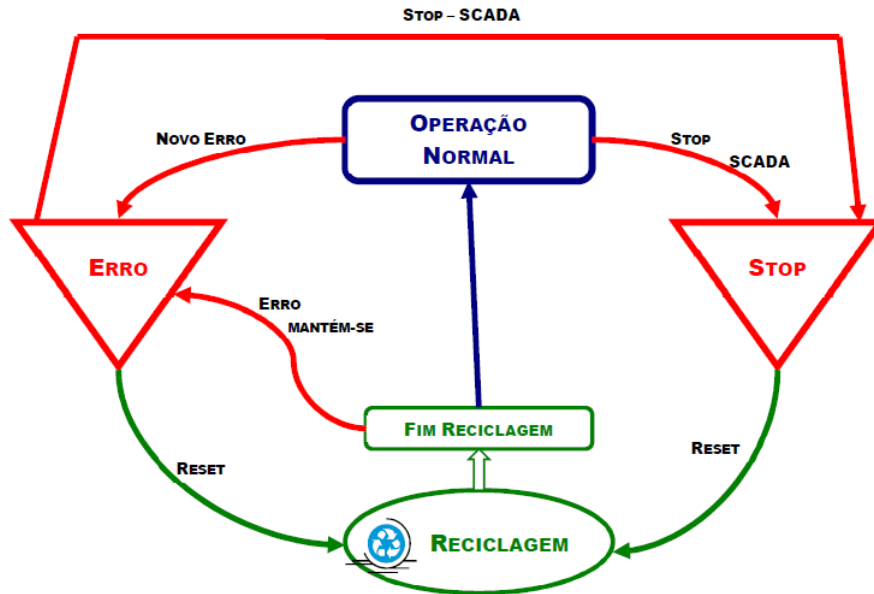


Figura 3.8: Diagrama de estados possíveis em modo automático

Quando o sistema entra em erro, só a área afetada fica em erro, ficando as restantes em operação normal. Ainda que em erro, a área permite que os restantes transportadores não afetados processem a movimentação normal. Os transportadores em erro suspendem qualquer movimento até que o operador restabeleça as condições normais de operação.

No caso de um erro por emergência, todos os transportadores da área abrangidos por essa emergência ficarão em estado de erro. O estado de erro elimina-se pelo processo de reciclagem.

É possível colocar uma área em stop, para paragem de sistema, ou manutenção de informação lógica de paletes nos transportadores. A reciclagem é um processo que pretende apagar informação do erro existente e verificar se o transportador ou a área se mantém em erro.

3.2.2.2 Modo Manual

O modo manual é ativado quando o utilizador o seleciona através de um comando existente no painel de controlo da zona pretendida. A partir do momento em que este modo é selecionado, todos os movimentos automáticos em execução são suspensos. Não existem diagnósticos de erro, permanecendo somente a vigilância do estado dos botões de emergência. Por questões de segurança, a saída deste modo de operação coloca a zona em questão em erro, sendo necessário ativar o processo de reciclagem para voltar ao modo automático.

Depois do modo manual ser ativado o controlo dos tapetes é feito através de uma

consola de operação, apresentada na figura 3.9, que pode ser ligada em qualquer ponto do sistema para o utilizador ter acesso ao equipamento existente nesse ponto.

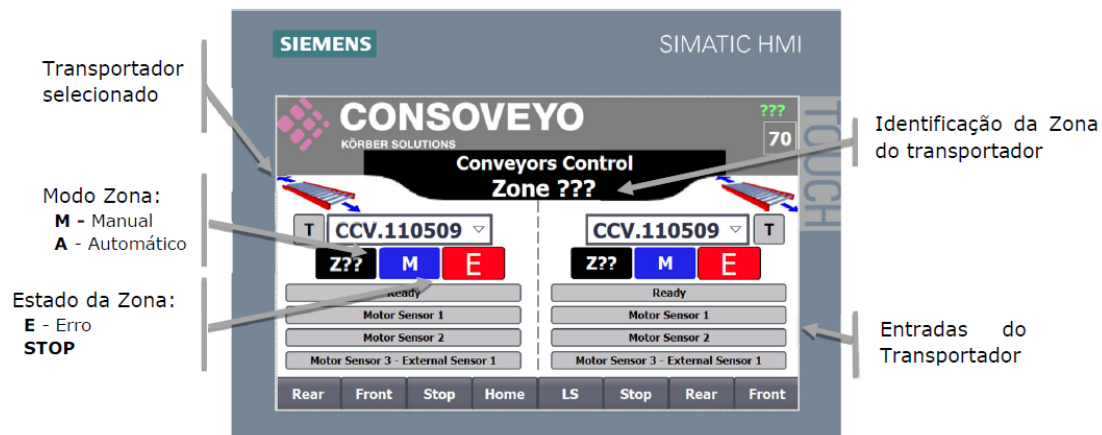


Figura 3.9: Ecrã de controlo dos transportadores da consola de operação

3.3 Ferramentas e Metodologias

Para o desenvolvimento de um armazém automático, desde o primeiro contacto com o cliente até todo o projeto estar bem definido, são usados vários software, cada um com uma função específica num enorme processo de projeção e programação. Nesta secção iremos fazer uma pequena descrição de cada um desses software e explicar de um breve modo como eles se inserem no desenvolvimento de um projeto.

3.3.1 AutoCAD

O programa AutoCAD é um software CAD (*Computer-Aided Design*) utilizado em diversas indústrias na criação de desenhos de duas e três dimensões. É possível editar geometria 2D e modelos 3D com superfícies e objetos sólidos e de malha, fazer anotações com texto, linhas de indicação e tabelas e é também possível personalizar o programa com aplicações e APIs suplementares. Possui ainda diversas bibliotecas com desenhos já feitos, para facilitar, ao utilizador, a criação de projetos [23].

Na Consoveyo S.A. este software é usado para fazer o design do primeiro layout de um armazém automático, sendo constantemente alterado e aperfeiçoado com o propósito de atingir a solução mais eficaz possível para o cliente, sendo os requisitos do mesmo o aspeto mais importante neste desenvolvimento. As soluções construtivas disponíveis na empresa e o orçamento fornecido pelo cliente também são fatores relevantes.

O layout é concebido utilizando uma extensa biblioteca, que é complementada e melhorada ao longo do tempo, com os desenhos de todos os transportadores, transelevadores, motores, sensores, etc., tudo que exista numa instalação. Posteriormente, esse desenho vai servir como base para a criação de todos os outros projetos nos restantes software.

3.3.2 Excel

O Excel é um software patentado pela Microsoft, e é um editor de folhas de cálculo com uma interface intuitiva com capacidades de ferramentas de cálculo e de construção de tabelas. Dispõem de milhares de funções que tornaram este programa num dos software mais utilizados em todo o mundo, principalmente no mundo industrial [24].

Na Consoveyo S.A. o Excel é utilizado para gerar o "esqueleto" da programação dos transportadores. Depois do layout em AutoCAD estar completado, é exportado para o Excel uma tabela com todos os transportadores existentes e toda a informação associada a cada um. Nessa tabela encontra-se especificado, relativamente a cada transportador, o tipo, ID, *power line*, endereço lógico, rede e endereço AS-i, o index e as zonas adjacentes.

Posteriormente, recorrendo a macros do Excel (sequência de comandos e funções armazenados num programa Visual Basic que pode ser usado sempre que o utilizador precisar de realizar uma tarefa repetitiva) é gerado o esqueleto do programa (figura 3.10). Por último, este código é inserido no TIA Portal ou SIMATIC STEP 7 numa UDT, onde o programador pode ir buscar a informação relativamente a cada transportador.

```
BEGIN
T[1].ID :=101;
T[1].Zone :=1;
T[1].Power_Line :=1;
T[1].TR :=TRUE;
T[1].TC :=FALSE;
T[1].ME :=FALSE;
T[1].MR :=FALSE;
T[1].BE :=FALSE;
T[1].EL :=FALSE;
T[1].TG :=FALSE;
T[1].T_Virtual :=FALSE;
T[1].Direction :=TRUE;
T[1].Data_Inc :=TRUE;
T[1].B7_b2 :=FALSE;
T[1].B7_b3 :=FALSE;
T[1].B7_b4 :=FALSE;
T[1].B7_b5 :=FALSE;
T[1].BE_Special :=FALSE;
T[1].ME_Special :=FALSE;
```

Figura 3.10: Código esqueleto gerado pelas macros no Excel

3.3.3 Siemens TIA Portal e SIMATIC STEP 7

O Siemens TIA Portal e o SIMATIC Step 7 são as plataformas utilizadas na Consoveyo S.A. para programação de PLCs utilizados nas soluções desenvolvidas pelos mesmos. Cada vez mais é usado o Siemens TIA Portal pois é um programa mais avançado, com ferramentas e funcionalidades mais abrangentes e desenvolvidas.

O SIMATIC STEP 7 é um pacote de software usado para configurar e programar PLCs SIMATIC, nas linguagens abrangidas pela norma IEC 61131-3. Dentro deste pacote temos as versões STEP 7 Micro/DOS e STEP 7 Micro/Win para aplicações mais simples do PLC SIMATIC S7-200. Como foi dito anteriormente, a utilização deste software está a ser descontinuada, em prol do TIA Portal, e atualmente, para o desenvolvimento de novos projetos já é utilizado o segundo [25].

O software TIA Portal (*Totally Integrated Automation*) é patenteado pela Siemens, desenvolvido para conseguir dar uma melhor resposta à crescente exigência dos projetos de automação. É uma ferramenta que otimiza os processos, rotinas de máquina e operacionais e harmoniza várias ferramentas relacionadas com a engenharia de automação, com o objetivo de ter acesso a todas elas a partir de um só sítio [26].

Seguindo o processo referido em 3.3.2, o código esqueleto é depois inserido no TIA Portal, normalmente numa DB ou numa UDT, figura 3.11. Uma DB é um bloco onde se guarda informação que vai ser utilizada várias vezes ao longo de um programa, e onde se pode guardar informação de vários tipos (int, real, time, bool, word, byte, etc.). Uma UDT é o mesmo que uma DB com uma pequena diferença: é possível usar uma UDT como base de uma DB para, por exemplo, poder alterar a estrutura lógica de um determinado transportador sem alterar a dos restantes.

UDT_CNV_T_CONFIG								
	Name	Data type	Default value	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
1	ID	Dint	0	☒	☒	☒	☐	Conveyor Identification Number
2	Zone	Int	1	☒	☒	☒	☐	Zone Number
3	Power_Line	Int	0	☒	☒	☒	☐	Power Liner Number
4	TR	Bool	false	☒	☒	☒	☐	Type: Roller Conveyor
5	TC	Bool	false	☒	☒	☒	☐	Type: Chain Conveyor
6	ME	Bool	false	☒	☒	☒	☐	Type: Lift Table
7	MR	Bool	false	☒	☒	☒	☐	Type: Rotative Table
8	BE	Bool	false	☒	☒	☒	☐	Type: Retractable Stopper
9	EL	Bool	false	☒	☒	☒	☐	Type: Elevator
10	TG	Bool	false	☒	☒	☒	☐	Type: Gravitational Conveyor
11	T_Virtual	Bool	false	☒	☒	☒	☐	Type: Virtual Conveyor
12	Direction	Bool	false	☒	☒	☒	☐	
13	Data_Inc	Bool	false	☒	☒	☒	☐	TRUE: Enable Data Incoherence; FALSE: Disable Data
14	B7_b2	Bool	false	☒	☒	☒	☐	
15	B7_b3	Bool	false	☒	☒	☒	☐	
16	B7_b4	Bool	false	☒	☒	☒	☐	
17	TR_Roller_Drv	Bool	false	☒	☒	☒	☐	
18	BE_Special	Bool	false	☒	☒	☒	☐	
19	ME_Special	Bool	false	☒	☒	☒	☐	Lift Table with more than one sensor for UP or DOW...

Figura 3.11: Exemplo de uma UDT no TIA Portal, com a informação referente à configuração de um transportador

Posteriormente é concebido, manualmente, o programa de controlo dos transportadores, normalmente em LD. Este programa contempla todos os erros e redundâncias possíveis durante o funcionamento do sistema, bem como a interface dos transportadores com outros equipamentos.

3.3.4 Eplan

Como referido na subsecção 2.2.2.2, este software é utilizado para o auxílio a projetos de automação. Na Consoveyo S.A., este software é utilizado na elaboração da arquitetura elétrica de um sistema periférico, e outros tipos de equipamentos. Todo o projeto elétrico, agrupamento de materiais, listagem de equipamentos usados para que possa ser feita a compra dos mesmos, listas para os pontos de ligação e verificação de qualidade do esquema elétrico é feito usando este programa.

3.3.5 *Warehouse Management System (WMS) e Material Flow Control (MFC)*

Para um controlo eficiente do fluxo de material num armazém, é necessário uma "ponte" entre o software de WMS e os diferentes controladores existentes no armazém automático. O software MFC funciona como essa ponte, recebendo informação do WMS, sobre cada produto transportado, e reencaminhando-a para os diferentes PLCs, para que o sistema de controlo físico do armazém tenha o suporte necessário lógico [27].

No WMS encontra-se toda a informação acerca dos produtos manuseados no sistema periférico. Esta informação é essencial para o correto funcionamento de todo o armazém automático ao garantir que as operações de gestão de inventário, *picking* e direcionamento de stock sejam corretamente executadas [28].

Estas ferramentas possuem ainda algumas funcionalidades de supervisão e controlam continuamente os diferentes processos de que estão a decorrer ao longo do armazém. O MFC certifica-se que o sistema de controlo do armazém age de maneira rápida e correta consoante as informações recebidas [27].

Na Consoveyo S.A. o MFC funciona como intermediário entre o PLC que controla os equipamentos que transportam as paletes e o WMS. Quando uma paleta chega à entrada de um sistema envia um pedido ao WMS (através do MFC) para que este lhe transmita o destino da paleta. Depois de o WMS transmitir ao PLC esta informação, o PLC encarrega-se de transportar essa paleta desde o ponto onde ela se encontra até ao seu destino. Para além disso, o WMS processa todos os dados referentes ao transporte e stock de paletes e disponibiliza-os ao utilizador para que este possa supervisionar o seu estado ao longo de todo o sistema, através de um SCADA.

3.4 Protocolos de comunicação utilizados

Para todos os software e equipamentos se manterem em sintonia e toda a informação seja atualizada, nos diferentes pontos de um armazém automático durante o funcionamento do mesmo, é necessário a troca de informação entre eles. Para isso são utilizados protocolos de comunicação, que permitem a correta passagem de informação. Nesta secção iremos abordar os protocolos de comunicação utilizados na empresa.

3.4.1 PROFINET

A comunicação Profinet, resumidamente, é a rede Profibus sobre Ethernet, e encontra-se normalmente ao nível do campo. Profibus é uma rede de dispositivos (ou controladores) cujo propósito é facilitar a construção de sistemas descentralizados, interligando controladores e seus periféricos numa única rede. É baseada no princípio mestre/escravo onde um ou mais mestres troca bits, bytes, words, etc. com os escravos na forma de variáveis I/O [2].

Os equipamentos que normalmente são ligados através desta rede são:

- Controladores de rede Profinet – associados a módulos de entradas/saídas digitais transmitindo a informação agrupada de sensores e atuadores diretamente ao PLC;

- Dispositivos de comunicação em infravermelhos – criando ligação óptica entre o PLC e os equipamentos disponíveis nos elementos móveis (traselevadores e carros de transferência);
- Encoders rotativos, laser ou de leitura – para o posicionamento de eixos de movimentação de máquinas;
- Leitores de código de barra e variadores de velocidade;
- Master AS-i (para interface com a rede AS-i).

Em modo multi-mestre a rede funciona como Token Bus. Token Bus é um protocolo de MAC (gere a partilha do meio de comunicação por base dos intervenientes) onde o meio é controlado por apenas um token/testemunho. Só a estação que detém o testemunho pode transmitir, e o tempo que a detém é limitado. Numa rede Profinet, para saber distinguir cada aparelho concentra-se mais no nome do mesmo e no seu endereço IP [2].

Para configurar a rede e definir que informação é trocada entre mestres e escravos, os mestres são configurados a partir de ficheiros GSD (*General Station Description*) ou EDS (*Electronic Data Sheet*) disponibilizados pelos fabricantes de escravos. Na Consoveyo S.A. o protocolo de comunicação Profinet é usado em toda a rede industrial.

3.4.2 Rede AS-i (*Actuator Sensor - Interface*)

Esta rede é utilizada principalmente para controlo de processo ao nível de sensores e atuadores que contêm um pequeno volume de informação e trabalham a nível de “bits”. Esta rede trabalha apenas com um sistema mestre (Master AS-i) que lê ciclicamente os dados dos escravos que se encontram na sua rede com um tempo máximo entre leituras de 5 ms [2].

A interface com o mestre AS-i proporciona uma ligação entre o nível de controlo e o nível de campo do sistema de controlo dos transportadores. Ele gere automaticamente a transmissão de dados na rede AS-i colocando os sinais de sensores e motores disponíveis para sistemas de comunicação de níveis mais elevados, tal como Profinet.

O endereço do escravo AS-i pode ser definido utilizando uma consola dedicada para o efeito ou em alternativa utilizando o mestre AS-i da respetiva rede. O endereço 0 está reservado para endereçamento automático ou para endereçamento a partir do mestre AS-i (desde que seja único na rede AS-i). Devido ao facto dos motores possuírem capacidade de ligação até 2 sensores (sensores ópticos e indutivos) a maioria dos sensores de campo encontram-se conectados aos motores, utilizando a rede AS-i como canal de comunicação com o PLC.

Na Consoveyo S.A. o protocolo de comunicação AS-i é usado para controlar os motores, para permitir a rotação num dos sentidos ou pará-lo, e recebem os sensores que se ligam no módulo do motor (costumam ser 4 bits).

Neste capítulo foi feito um resumo de como é desenvolvido, atualmente na Consoveyo S.A., o projeto de um armazém automático. Existem vários aspetos que podem ser melhorados, como a elaboração do código PLC, pois não se encontram bem definidos. A metodologia a ser implementada pelo grupo Körber pretende realizar estas melhorias e tornar todo o processo mais eficiente e lucrativo para a empresa.

Abordagem prática - Geração de código PLC

Para a parte prática desta dissertação foi definida a criação de uma aplicação que fosse de encontro com o standard do grupo Körber para a possível utilização na Consoveyo S.A.. Esta escolha foi feita com base no facto da Consoveyo S.A. não ter um standard interno bem definido que use o software Visio que, como vimos na em 2.2.2.1, tem várias funcionalidades muito úteis na criação de um layout de transportadores. O facto da empresa não usar qualquer tipo de simulador para verificar o código concebido também pesou nesta decisão.

Neste capítulo é feita uma descrição detalhada de geração automática de código PLC, realizada pela aplicação desenvolvida, para o controlo de um sistema periférico. Em primeiro lugar são abordadas as ferramentas consideradas para a elaboração da aplicação, nomeadamente as suas potencialidades e possibilidades de interação com outras ferramentas.

De seguida é definido o objetivo da aplicação e todas as ferramentas e métodos constituintes da mesma. É explorado o papel de cada programa durante o processo e explicado o funcionamento da interface com o utilizador, em Visio. A modificação de ficheiros XML também é abordada de um modo mais profundo, uma vez que é a base para a geração automática de código PLC.

Importante referir a dissertação de mestrado realizada pelo ex-aluno do Mestrado de Automação de Engenharia Mecânica, Henrique Domingos de nome *Desenvolvimento de uma aplicação de geração automática de código para o PLC de controlo de um shuttle car*. A presente aplicação tem aspetos semelhantes, uma vez que também gera automaticamente código no TIA Portal, porém nenhuma metodologia ou ferramenta foi utilizada por ter conhecimento da dissertação mencionada.

4.1 Ferramentas e metodologias utilizadas

Nesta secção são abordadas as ferramentas e metodologias utilizadas no desenvolvimento da parte da aplicação que gera automaticamente código PLC. A primeira ferramenta explorada foi o Sharp7 e a sua biblioteca de classes. Posteriormente foi explorado

e estudado o TIA Portal Openness devido às suas funcionalidades de geração automática de código e interface com o TIA Portal.

4.1.1 Sharp 7

Numa primeira fase de exploração de ferramentas que pudessem interagir com o TIA Portal a partir de uma interface externa, foi encontrado um projeto de nome *Sharp7* com métodos, em C#, que permitem interagir com o TIA Portal. Porém, as funções existentes neste projeto não são aplicáveis em toda a gama de PLCs. Na figura 4.1 temos as funções existentes e em que gama de PLCs podem ser aplicadas.

	CPU						
	300	400	WinAC	1200	1500	LOGO	S7200
DB Read/Write	O	O	O	O(1)	O(1)	O(2)	O(2)
EB Read/Write	O	O	O	O	O		
AB Read/Write	O	O	O	O	O		
MK Read/Write	O	O	O	O	O		
TM Read/Write	O	O	O	-	-	-	-
CT Read/Write	O	O	O	-	-	-	-
Get/Set PLC Date and Time	O	O	O	-	-	-	-
Get PLC Status	O	O	O	-	-	-	-
Read SZL	O	O	O	-	-	-	-
Get AG Block Info	O	O	O	-	-	-	-
DB Get	O	O	O	-	-	-	-
Control Run/Stop	O	O	O	-	-	-	-
Security functions	O	O	O	-	-	-	-

(1) See S71200/1500 Notes

(2) The entire memory is mapped in the V Area accessible as DB 1 from the outside.

Figura 4.1: Compatibilidade de funções com PLCs

O projeto referido vem acompanhado com documentação de nome *Sharp 7 Reference manual*. No processo de estudo desta ferramenta foram testadas as suas funcionalidades ao desenvolver um programa que alterasse valores de variáveis existentes num *data block*, representado na figura 4.2.

Data_Block				
	Name	Data type	Offset	Start value
1	Static			
2	Transportador	Array[0..5] o...	...	
3	Transportador[0]	Struct	...	
4	ID	Int	...	0
5	Zona	Int	...	0
6	Power_line	Int	...	0
7	TR	Bool	...	false
8	TC	Bool	...	false
9	ME	Bool	...	false
10	MR	Bool	...	false
11	EL	Bool	...	false
12	BE	Bool	...	false
13	TG	Bool	...	false
14	TV	Bool	...	false
15	Direction	Bool	...	false

Figura 4.2: DB a alterar no TIA Portal

Para tornar possível a comunicação entre o software de programação em C# e o simulador de PLC do TIA Portal, foi necessário criar um servidor, pois a biblioteca de Sharp 7 comunica através de TCP/IP. Este servidor foi criado com a ajuda do software *NetToPLCsim* (figura 4.3).

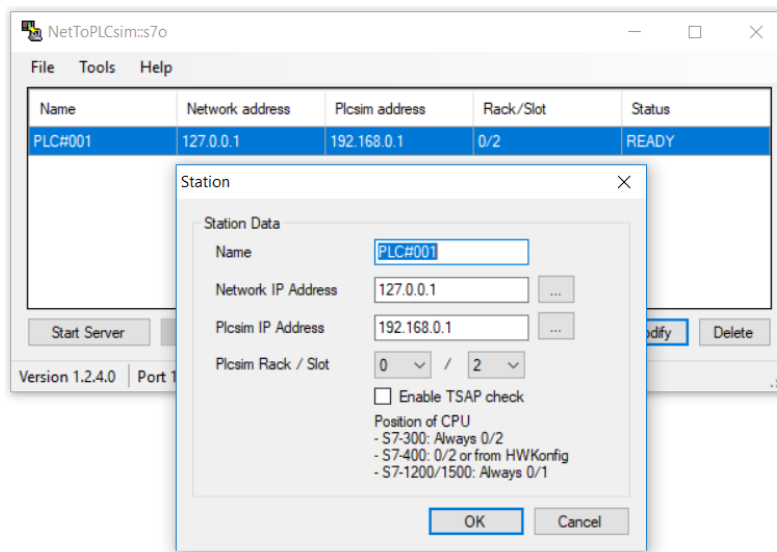


Figura 4.3: Criação do servidor no software *NetToPLCsim*

Posteriormente usaram-se algumas das funções presentes na figura 4.1. Para além dessas funções, foi utilizado o método *ConnecTo()* que permite fazer a ligação entre o programa e o PLC em questão. Depois de estabelecida a comunicação, foi possível modificar as variáveis do *data block* através de um *form* criado na aplicação em C#, como podemos ver na figura 4.4.

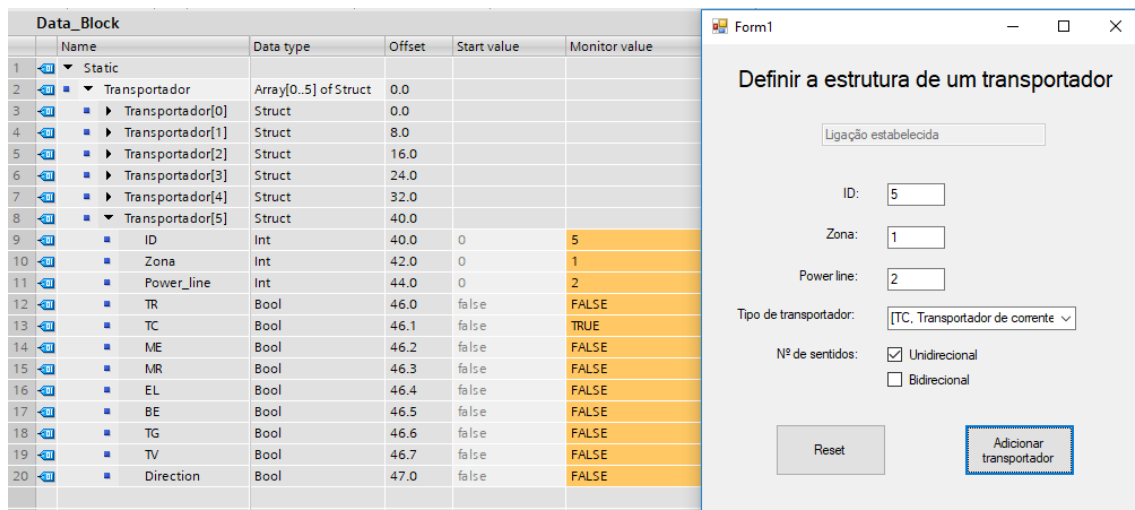


Figura 4.4: Interface de alteração de variáveis no TIA Portal

Porém, verificou-se que deste modo as variáveis eram apenas escritas no PLC, não alterando o *start value* de cada uma. Para isso é necessário o utilizador selecionar a opção *Copy snapshots to start values* e, ao reiniciar o PLC, os valores previamente carregados seriam perdidos, mantendo-se os valores iniciais.

Tendo então conhecimento das funcionalidades do Sharp 7, não foi escolhida esta ferramenta na abordagem da parte prática, uma vez que só permite a alteração e parametrização de variáveis e não a criação das mesmas. Esta ferramenta teria uma boa aplicação na monitorização e alteração em tempo real de variáveis já existentes num programa PLC.

4.1.2 TIA Portal Openness

A descrição do TIA Portal Openness encontra-se em 2.2.3.3 e como foi referido, a Siemens disponibiliza duas APIs com a instalação do TIA Portal Openness: Start Openness (figura 4.5) e Demo Openness (figura 4.6). Ambas as APIs permitem alterar propriedades de projetos do TIA Portal através de uma interface, bem como exportar e importar ficheiros. Portanto, desde a primeira interação com este software foi possível perceber que as suas funcionalidades seriam essenciais no desenvolvimento de uma aplicação para a geração de código automático.

Depois de analisada a API Start Openness foi possível perceber as suas utilidades e funcionalidades, como iniciar o software TIA Portal, cortar a ligação com o mesmo, procurar por um determinado projeto e gravar ou fechar.

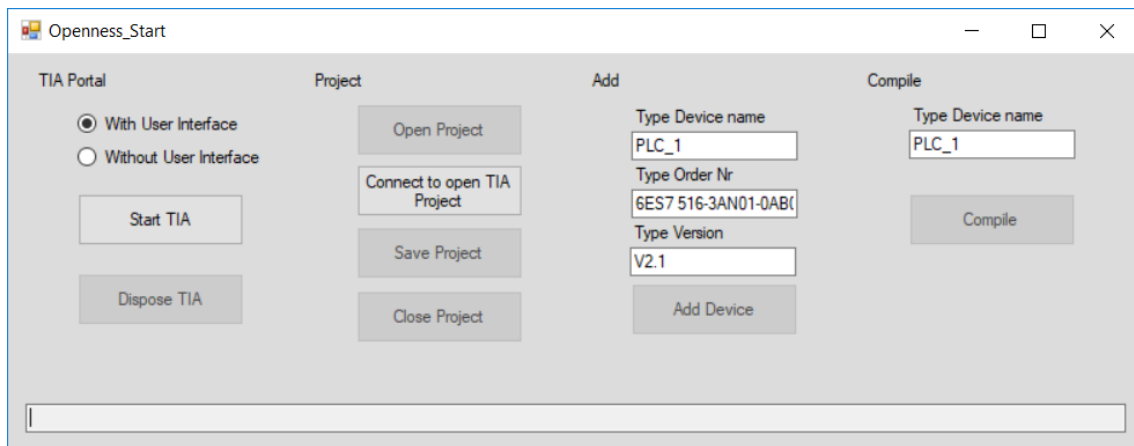


Figura 4.5: Interface da aplicação *StartOpenness*

A segunda aplicação Demo Openness já possui um leque mais vasto de funções, e já dá uma ideia do real potencial do TIA Portal Openness. A própria interface desta aplicação é mais complexa, comparado com a da aplicação Start Openness, como podemos ver na figura 4.6. Dentro desta aplicação existem dois projetos: *TiaOpennessHelper* e *TiaOpennessDemo*, as quais possuem vários métodos para interagir com o TIA Portal, que podem ser usados em programas criados por utilizadores. Lendo a documentação associada a esta aplicação (*TIA Portal Openness: Introduction and Demo Application* e *Automating projects with scripts*) foi possível identificar os métodos existentes nestes projetos e as suas finalidades:

- *TiaPortal*: inicia o TIA Portal;
- *Projects.Open(new FileInfo(filename))*: abre um projeto TIA Portal já existente, ao especificar onde este se encontra;
- *Delete()*: apagar elementos da árvore de um equipamento;

- *Compile()*: compilar o projeto do TIA Portal;
- *Export(path, exportOption)*: exportar elementos do projeto, através de ficheiros XML, escolhendo o local onde vão ser armazenados e o tipo de exportação;
- *Import(path, importOption)*: importar ficheiros para o projeto, através de ficheiros XML, especificando onde se encontram e o tipo de importação;
- *GoOnline()*: estabelece uma conexão online com um autómato ou simulador que esteja conectado;
- *GoOffline()*: corta a ligação que o projeto tenha com um autómato ou simulador;
- *Project.Save()*: grava o projeto aberto no TIA Portal;
- *Project.Close()*: fecha o projeto aberto no TIA Portal.

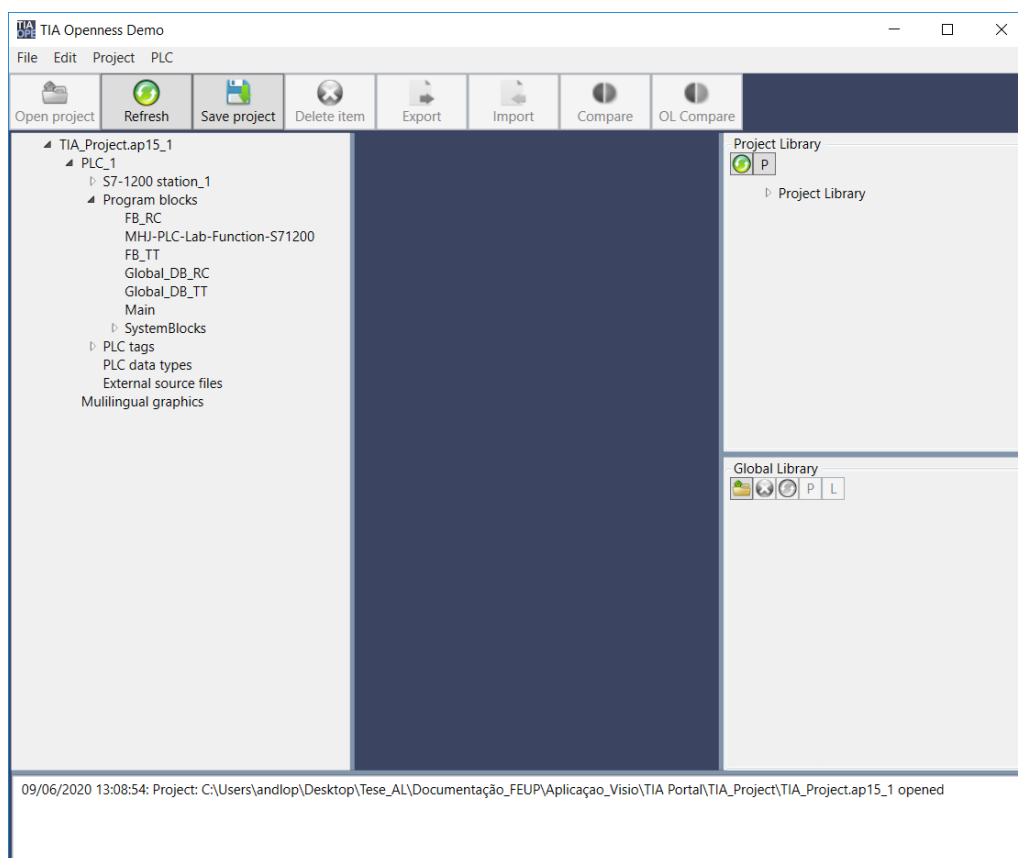


Figura 4.6: Interface da aplicação *OpennessDemo*

Depois de exploradas e avaliadas todas estas funções, o TIA Portal Openness foi escolhido para criar a ponte entre a ferramenta de programação externa à Siemens (Visual Studio) e o TIA Portal.

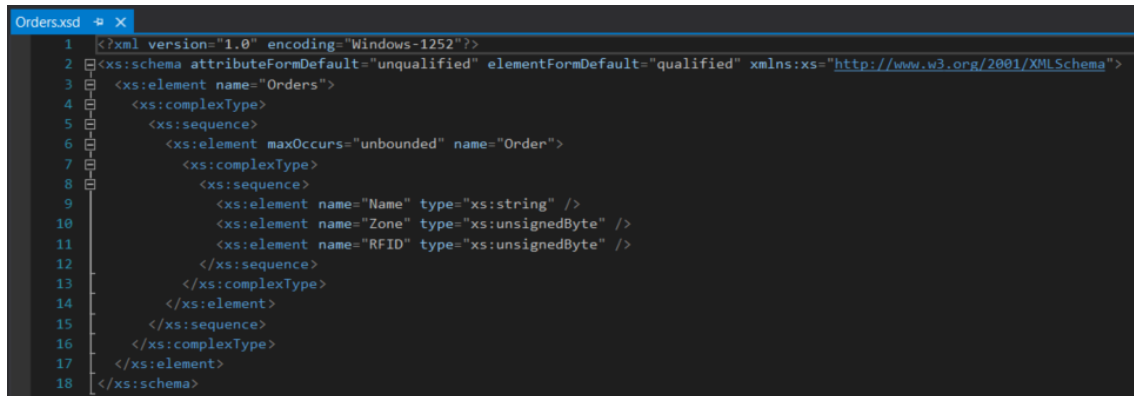
4.1.3 Ficheiros XML

Os ficheiros XML têm como propósito o armazenamento e transporte de informação de uma maneira clara e universal, uma vez que são vários os softwares e ferramentas que

permitem a leitura e codificação deste tipo de ficheiros. O TIA Portal Openness permite a criação de vários tipos de blocos de um programa PLC através da importação de ficheiros XML [29].

Os ficheiros XML importados têm que seguir uma estrutura pré-definida para que sejam corretamente descodificados. A estrutura que um determinado ficheiro XML tem que seguir é denominado por *schema*. Na figura 4.7 temos um pequeno exemplo de um *schema* de um ficheiro XML usado na aplicação desenvolvida.

Na geração de ficheiros XML para o controlo de equipamentos no TIA Portal, existe um *schema* para cada tipo de bloco.



```
1 <?xml version="1.0" encoding="Windows-1252"?>
2 <xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
3   <xs:element name="Orders">
4     <xs:complexType>
5       <xs:sequence>
6         <xs:element maxOccurs="unbounded" name="Order">
7           <xs:complexType>
8             <xs:sequence>
9               <xs:element name="Name" type="xs:string" />
10              <xs:element name="Zone" type="xs:unsignedByte" />
11              <xs:element name="RFID" type="xs:unsignedByte" />
12            </xs:sequence>
13          </xs:complexType>
14        </xs:element>
15      </xs:sequence>
16    </xs:complexType>
17  </xs:element>
18 </xs:schema>
```

Figura 4.7: *Schema* do ficheiro XML *Orders*

4.1.3.1 Estrutura de um ficheiro XML

Como foi dito anteriormente, os ficheiros XML são elementares e claros, isso porque são construídos seguindo uma estrutura em forma de árvore. Existe o primeiro elemento (também chamado de elemento raiz ou *root element*) e este contém várias ramificações que são outros elementos denominados de elementos filhos ou *child elements*. Todos os elementos que têm elementos filhos são denominados de elementos pais ou *parent elements*. Para além de se poder incluir elementos dentro de outros, cada elemento também pode conter texto ou atributos. Toda a lógica de um documento XML é assente nestas ramificações e segue as seguintes regras de sintaxe:

- Todos os elementos têm uma *tag* inicial e uma *tag* final, caso tenham elementos filhos ou texto (figura 4.8);
- Geralmente, na primeira linha de todos os ficheiros XML existe um prólogo onde se encontra a versão do ficheiro e/ou a codificação usada no mesmo;
- Os nomes dos elementos não podem conter espaços e são sensíveis a letras maiúsculas ou minúsculas.

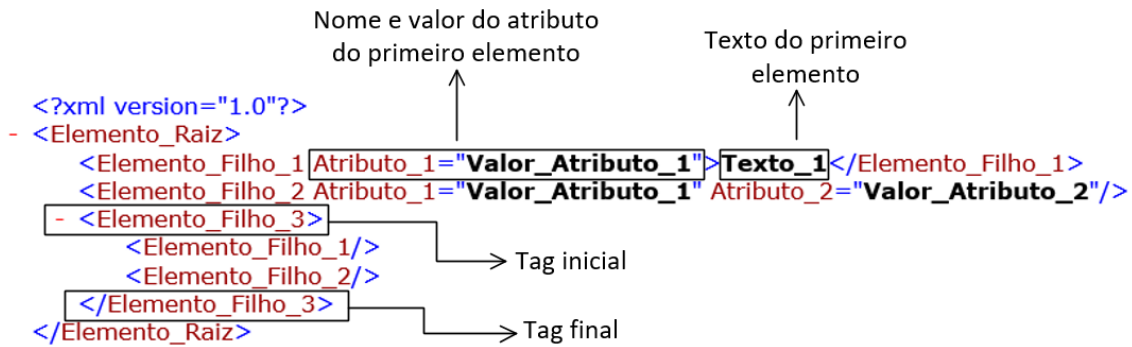


Figura 4.8: Exemplo de um ficheiro XML

4.1.4 TIA Portal

Foi escolhido o software de automação TIA Portal para programar e albergar o código PLC desta aplicação. A escolha foi simples uma vez que, neste momento, é o software com mais funcionalidades neste campo e o usado na Consoveyo S.A. na programação de PLCs e comunicações com outros software. Para além disso consegue comunicar com o TIA Portal Openness que permite a edição e manipulação de projetos no TIA Portal através de outras aplicações, funcionalidade essencial para a elaboração da aplicação abordada na parte prática desta dissertação.

A programação em TIA Portal pode ser escrita nas linguagens referidas em 2.1.1.2 e funciona à volta de 4 tipos de blocos:

- *Data block* (DB): blocos que guardam dados referentes aos programas;
- *Organization block* (OB): existem vários tipos de *organization blocks*, porém, no programa desenvolvido é apenas usado o *program cycle*. Este tipo de OB é sempre executado e é o bloco principal do programa. É onde as instruções de controlo do programa e outros blocos são colocados;
- *Function block* (FB): blocos de código que armazenam os seus valores de maneira permanente em DBs instanciados para que permaneçam disponíveis depois deste bloco ter sido executado;
- *Function* (FC): blocos de código ou subrotinas sem memória dedicada (aspeto que o difere de um *function block*).

Para além destes blocos existe ainda uma zona dedicada às *tags* do PLC, ou seja, as variáveis correspondentes a todo o hardware necessário controlar numa instalação, que costuma ter sensores, botões, sinalizadores, drives, leitores de códigos, balanças, entre outros.

Toda esta informação encontra-se associada a um determinado PLC. Para isso é necessário selecionar qual o PLC a usar, do catálogo providenciado no TIA Portal, no início de cada projeto. Para cada hardware selecionado temos vários campos com informação a adicionar como os blocos de programação, tabelas de *tags*, módulos e outros.

4.2 Objetivo da aplicação

A aplicação desenvolvida tem como objetivo gerar automaticamente código PLC, no TIA Portal, para controlo de um layout de transportadores para a movimentação de paletes, de modo a flexibilizar este tipo de programas ao diminuir o tempo de programação manual e ao oferecer uma interface gráfica para uma melhor compreensão do layout formado. Para além de gerar código, a aplicação também cria, automaticamente, um ambiente virtual de simulação 3D com todos os equipamentos e aparelhos necessários para testar o código gerado. O cenário virtual gerado é uma "réplica" do desenho realizado em Visio.

A aplicação gera o código PLC para transportadores de rolos, mesas de transferência e mesas rotativas a partir de um desenho em Visio, onde também se encontra especificado os pontos de entrada e saída do circuito. Deste modo é possível criar uma infinidade de layouts, ao adicionar diferentes equipamentos, criando vários caminhos por onde as paletes podem seguir.

Muitas vezes as empresas de desenvolvimento de armazéns automáticos não usam software de simulação para testar as soluções encontradas, e respetivo código, devido à elevada quantidade de tempo que leva a recriar um armazém automático num software destes, bem como a complexidade dos mesmos. Ao gerar o cenário de simulação ao mesmo tempo que é gerado o código PLC, este obstáculo deixaria de existir, tendo as empresas um ponto de controlo extra antes de implementar a solução no armazém.

Na figura 4.9 encontra-se um exemplo de um sistema periférico criado no Visio.

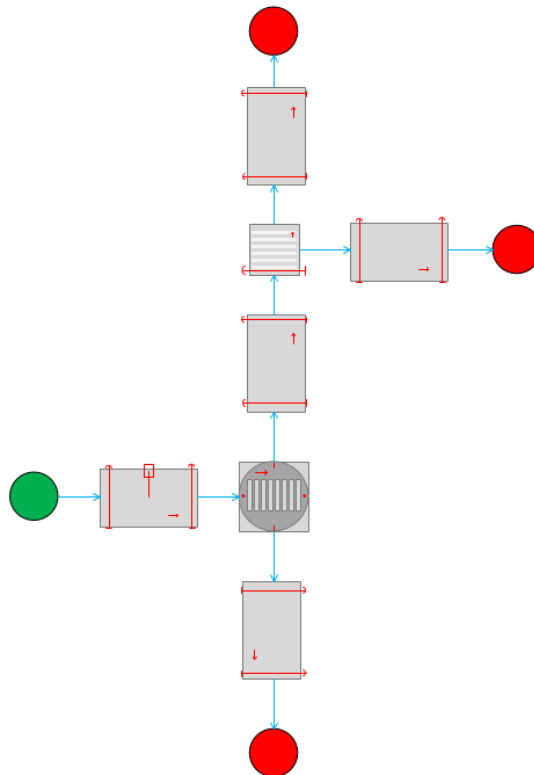


Figura 4.9: Layout criado em Visio

4.3 Estrutura da aplicação

Na figura 4.10 encontra-se um diagrama que representa o funcionamento da aplicação desenvolvida, desde a criação do layout em Visio até ao código gerado em TIA Portal e o cenário virtual de simulação recriado automaticamente no Factory IO.



Figura 4.10: Diagrama representativo do funcionamento da aplicação

A primeira fase deste processo é a construção do layout em Visio e a gravação do mesmo no formato pré-definido *.vsdx*. Esta extensão indica que se trata de um desenho em Visio e são ficheiros OPC/XML onde é possível extrair do mesmo ficheiros XML correspondentes ao conteúdo do desenho, como as formas existentes, as páginas e como se relacionam. Na figura 4.11 temos o ficheiro obtido e usado para retirar informação do desenho, quando é

feita a extração do ficheiro *.vsdx*, onde conseguimos ver a estrutura do mesmo.

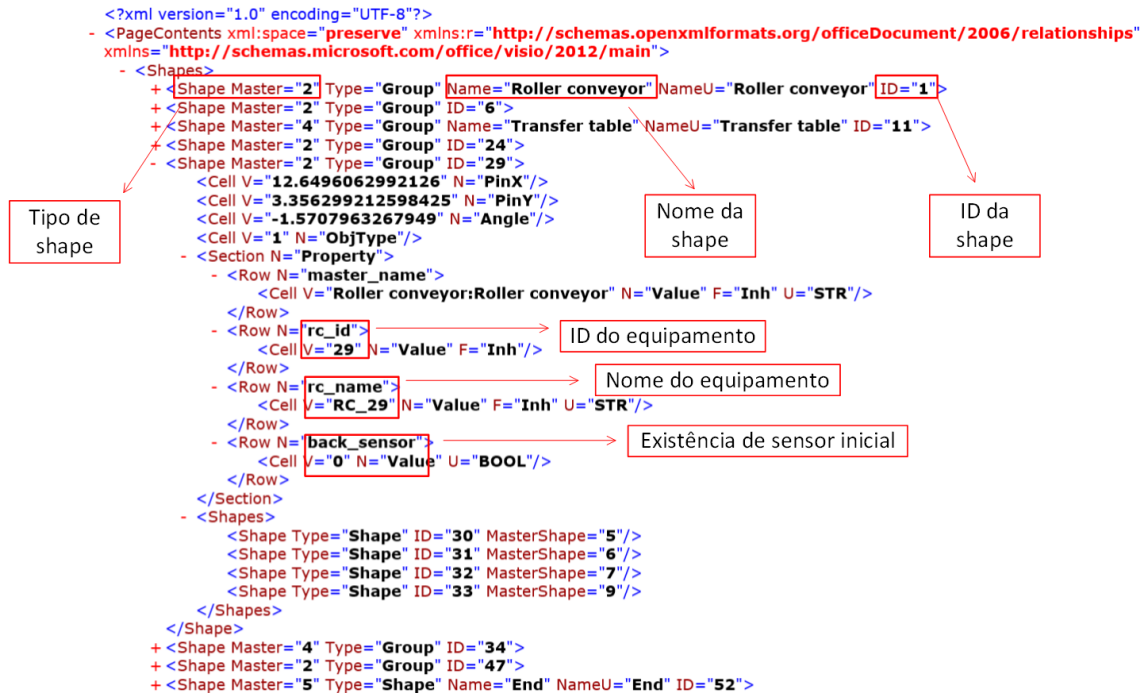


Figura 4.11: Ficheiro XML correspondente ao desenho criado em Visio, extraído do ficheiro de formato *.vsdx*

Concluído e gravado o layout em Visio passa-se para a aplicação .NET desenvolvida no Visual Studio. A aplicação vai extrair e ler o ficheiro XML correspondente ao desenho em Visio (*page1.xml*), e vai devolver informação referente ao mesmo, como a quantidade de transportadores, mesas de transferência e mesas rotativas existentes, a existência de sensores, os equipamentos com que cada um interage e as rotas desde a entrada do circuito até às diferentes saídas. Toda essa informação é vital e é utilizada para a geração de ficheiros XML para serem importados para o TIA Portal e para a construção do circuito no simulador 3D.

De seguida está na altura de gerar os documentos XML que têm como destino o TIA Portal. Estes ficheiros não são propriamente criados. A aplicação acede a ficheiros XML modelo para cada tipo de bloco (*global* e *instance* DBs, FBs, OBs e *tag tables*), previamente adquiridos a partir da API Demo Openness. A aplicação altera estes ficheiros "limpos", ou seja, sem informação nenhuma acerca de programação PLC, ao adicionar-lhes informação correspondente ao código que vai ser gerado para o controlo dos equipamentos, seguindo uma estrutura pré-definida (*schemas*) para o ficheiro XML de cada tipo de bloco.

A geração do ficheiro XML para construção do ambiente de simulação segue um princípio semelhante aos ficheiros que têm como destino o TIA Portal. A aplicação acede a um ficheiro modelo de um cenário em Factory IO, onde já existe uma coluna, um quadro elétrico e três botões. Posteriormente, consoante o desenho do Visio, esse ficheiro é alterado, e é adicionada a informação referente a cada objeto que é necessário colocar no ambiente virtual. É também feita a associação das *tags* existentes no Factory IO com as variáveis correspondentes no TIA Portal.

Por último, a aplicação guarda os ficheiros XML alterados para depois os importar no projeto modelo do TIA Portal. O código usado para realizar esta importação é o

mesmo que na API referida anteriormente (Demo Openness). O projeto modelo apenas contém três FBs e um OB. As FBs contêm o código PLC de controlo de cada equipamento individual, uma o código dos transportadores de rolos, outra das mesas de transferência e a última das mesas rotativas.

O documento XML correspondente ao ambiente virtual, já com toda a informação necessária, é guardado como ficheiro XML e como *Factory IO Scene*, para que possa ser aberto com o respetivo programa. Para isso o ficheiro XML é guardado uma segunda vez, mas com a extensão *.factoryio*.

4.4 Construção do layout em Visio

Do ponto de vista do utilizador, o primeiro passo no uso da aplicação é a criação do sistema periférico pretendido no Visio. Para que esta construção do layout em Visio fosse correta e seguisse sempre as mesmas regras e métodos bem definidos, foi criado um *stencil* com blocos pré-definidos, cada um com uma determinada função.

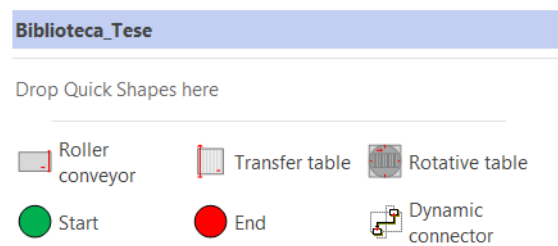


Figura 4.12: *Stencil* criado para a construção do layout em Visio

Para criar o *stencil* da figura 4.12 foram nomeados e desenhados os blocos, e programados outros aspetos dos mesmos, como a existência de leitor RFID ou de determinados sensores, para serem visíveis quando uma determinada variável estivesse ativada, aspetos ilustrados na figura 4.13. Essas variáveis foram também inseridas manualmente na *shape data* de cada bloco. A *shape data* de um bloco é uma secção que contém todas as variáveis associadas ao bloco, que podem ser adicionadas por um utilizador ou já estarem implementadas. Nessa secção é também possível alterar o valor de uma determinada variável.

No *stencil* criado os dados existentes na *shape data* variam de bloco para bloco. A única variável comum a todos é o ID do bloco no desenho.

Define Shape Data

Label:

Name:

Type: Language:

Format: Calendar:

Value:

Prompt:

Sort key:

☒ Ask on drop ☐ Hidden

Properties:

Label	Name	Type
ID	rc_id	Number
Name	rc_name	String
Back sensor	back_sensor	Boolean
RFID Reader	rfid	Boolean

Figura 4.13: Atribuição das variáveis na *shape data* de um bloco

Como podemos ver na figura 4.12, existem 6 blocos:

- *Start*: indica o princípio do circuito, ou seja, o local onde as paletes entram no sistema de transportadores;
- *End*: indica o fim do circuito, ou seja o local ou locais onde as paletes saem do sistema de transportadores;
- *Roller conveyor*: representa um transportador de rolos com a ilustração do sentido que a paleta se movimenta;
- *Transfer table*: representa uma mesa de transferência onde as paletes podem mudar de direção (para a esquerda ou a direita) ou seguir em frente;
- *Rotative table*: representa uma mesa rotativa onde, tal como nas mesas de transferência, as paletes podem mudar de direção. Diferem das mesas rotativas pois permitem um alívio do fluxo de paletes;
- *Dynamic connector*: são setas usadas para ligar os blocos uns aos outros, para indicar ao programa qual o caminho que as paletes podem tomar.

Start

O bloco *Start*, como dito anteriormente, representa o ponto de entrada das paletes no circuito de transportadores, e não tem nenhum parâmetro alterável no Visio. No Factory IO, é representado por um *Emitter*, que é um objeto que gera paletes e caixas, em intervalos de tempo definidos pelo utilizador. A aplicação desenvolvida só permite a existência de um ponto de entrada.

Na figura 4.14 encontram-se as duas representações, no Visio e no Factory IO.

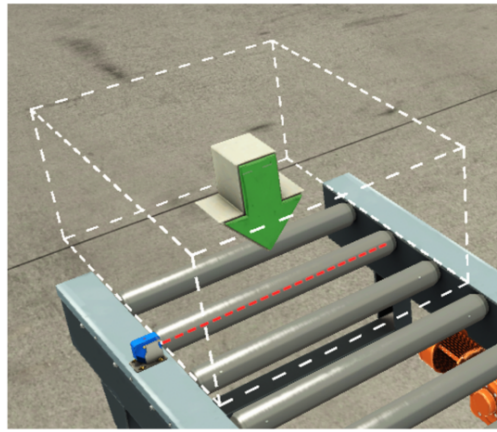


Figura 4.14: Bloco *Start* no Visio (esquerda) e *Emitter* no Factory IO (direita)

End

No bloco *End* o utilizador tem que inserir qual o número da zona de saída, informação que é definida quando é inserido o bloco, ou pode ser posteriormente alterada na *shape data* do mesmo (figura 4.15). Uma vez que a aplicação está preparada para gerar código PLC que envia as paletes para um determinado destino consoante o seu código, é necessário especificar cada destino. De notar que é obrigatório haver pelo menos um destino, e que exista um destino com o número 0, que corresponde à zona das paletes rejeitadas. A aplicação desenvolvida permite a existência do número de saídas que o utilizador desejar.

Shape Data - End		□ ×	
ID	43		
Zone	0		

Figura 4.15: Informação a inserir no bloco *End*

No Factory IO, o bloco *End* corresponde a um *Remover* que é um objeto que retira as paletes do cenário de simulação conforme estas chegam ao destino final, como se encontra na figura 4.16.

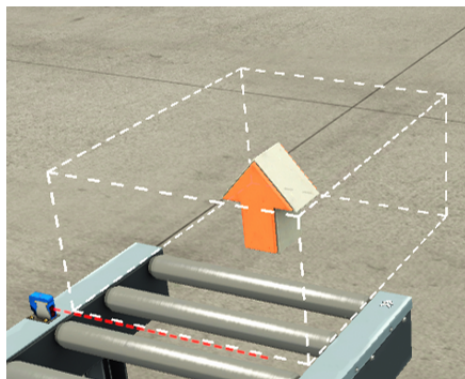
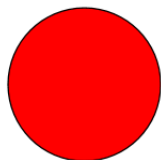


Figura 4.16: Bloco *End* no Visio (esquerda) e *Remover* no Factory IO (direita)

Roller Conveyor

Na *shape data* do bloco *Roller conveyor* (figura 4.17) conseguimos consultar o seu ID, nome (o nome deste bloco é "RC_(ID do bloco)"), consultar/alterar a existência de um sensor na zona inicial do equipamento e de um leitor RFID. Notar que o ID do bloco no Visio é o mesmo ID usado para identificar o equipamento posteriormente. Cada transportador deste tipo possui um drive para ativação do mesmo.

O transportador movimenta paletes apenas numa direção através da ativação do *drive*.

Shape Data - Roller conveyor	
ID	2
Name	RC_2
Back sensor	TRUE
RFID Reader	TRUE

Figura 4.17: *Shape Data* de um *Roller conveyor*

O aspeto deste bloco, tanto no Visio como o objeto correspondente no Factory IO, encontram-se na figura 4.18, onde temos um transportador de rolos com as opções de existência de sensor na parte inicial e de leitor RFID ativadas. O sensor na parte final existe sempre para o correto controlo deste tipo de equipamento.

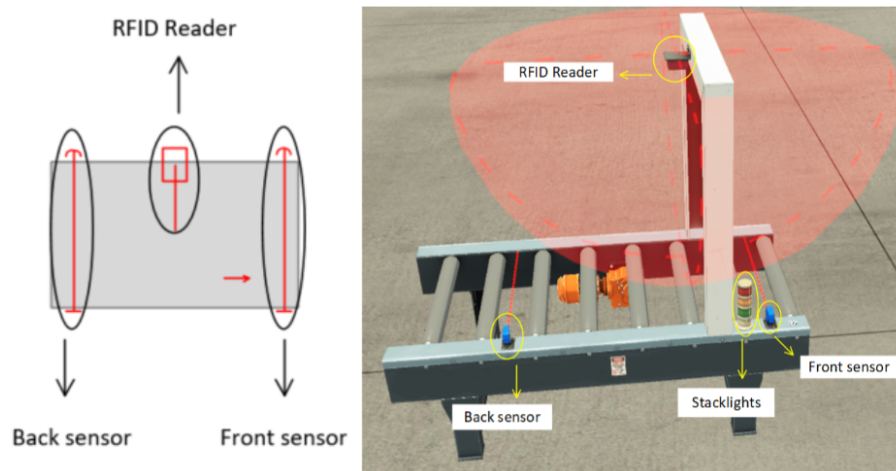


Figura 4.18: Bloco *Roller conveyor* no Visio (esquerda) e no Factory IO (direita)

Transfer Table

Na *shape data* do bloco *Transfer table* (figura 4.19) conseguimos ter acesso à mesma informação que no bloco *Roller conveyor*, menos a escolha da existência de um sensor, pois neste caso existe sempre um sensor no início da mesa e não é possível adicionar mais nenhum.

As mesas de transferência possuem três drives: um que ativa os rolos que movimentam as paletes em frente, e os outros dois ativam as correntes que movimentam as paletes para a esquerda ou direita.

Shape Data - Transfer table		□ ×
ID	6	
Name	TT_6	

Figura 4.19: *Shape Data* de uma *Transfer table*

O aspeto deste bloco, tanto no Visio como o objeto correspondente no Factory IO, encontram-se na figura 4.20, onde temos uma mesa de transferência apenas com um sensor na entrada, que existe sempre para conseguir o correto controlo de cada mesa de transferência.

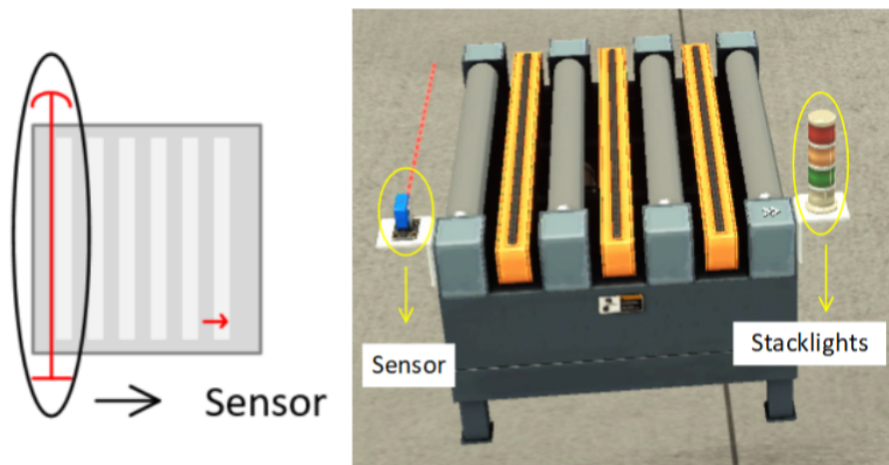


Figura 4.20: Bloco *Transfer table* no Visio (esquerda) e no Factory IO (direita)

Rotative table

Na *shape data* do bloco *Rotative table* conseguimos ter acesso à mesma informação que no bloco *Transfer table* (figura 4.21) uma vez que todos os sensores associados a este equipamento já se encontram contemplados no bloco. As mesas rotativas possuem dois sensores de deteção de paletes, um na zona inicial (*Back sensor*) e outro na zona final (*Front sensor*). Possuem ainda outros dois sensores que detetam a posição da mesa. Um ativa-se quando a mesa se encontra na posição vertical, de entrada de paletes (*Sensor limit 0*) e o outro quando a mesa se encontra na posição horizontal, depois de rodar 90° (*Sensor limit 90*).

As mesas rotativas possuem ainda três drives: movimentar a paleta em frente, movimentar a paleta para trás e rodar a mesa. A mesa roda para a posição horizontal quando o drive correspondente a esse movimento é ativado. Caso contrário a mesa encontra-se na posição vertical.

Shape Data - Rotative table

ID	69
Name	RT_69

Figura 4.21: *Shape Data* de uma *Rotative table*

O aspeto deste bloco, tanto no Visio como o objeto correspondente no Factory IO, encontram-se na figura 4.22. Os sensores que detetam as diferentes posições da mesa não se encontram visíveis no cenário de simulação.

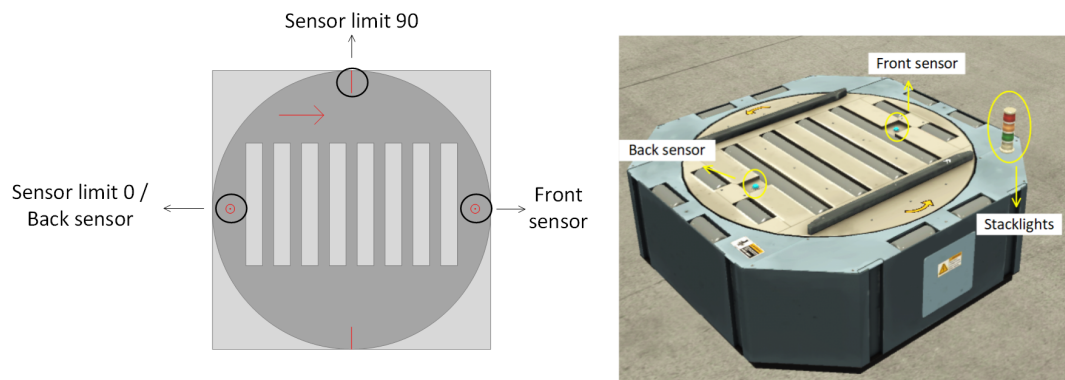


Figura 4.22: Bloco *Rotative table* no Visio (esquerda) e no Factory IO (direita)

Dynamic connectors

O ficheiro XML retirado do Visio tem informação relativa a todos os blocos usados num determinado desenho e a respetiva *Shape data* desses blocos. No caso do bloco *Dynamic connector*, no ficheiro XML da figura 4.23 encontra-se informação detalhada das ligações que fazem, como o bloco de origem e de destino, e os pontos onde se ligam a cada bloco (*connection points* existente no perímetro de alguns blocos).

Com esta informação é possível descobrir quais os blocos que se encontram ligados e, no caso das mesas de transferência e mesas rotativas, é possível descobrir em que direção se encontram os transportadores de rolos que recebem paletes das mesmas.



Figura 4.23: Informação associada a um *Dynamic connector*, representada no ficheiro XML

4.5 Interface de controlo de ordens

A aplicação desenvolvida, numa perspetiva real, será utilizada por um engenheiro encarregue de programar o controlo dos equipamentos existentes num circuito de transportadores. Nessa programação, terão que existir variáveis que digam respeito ao conteúdo e destino de cada paleta, informação normalmente fornecida pelo WMS.

Portanto, para aproximar a aplicação desenvolvida a um contexto empresarial real, foi desenvolvida em paralelo, uma pequena aplicação .NET no Visual Studio para inserir dados relativos a encomendas. Esta aplicação tem como objetivo substituir o WMS, no âmbito desta dissertação, na testagem do código por parte do engenheiro, para assegurar que as paletes percorrem o trajeto correto. No mundo real, o equivalente a esta pequena aplicação seria um interface do cliente para este submeter os dados de cada encomenda.

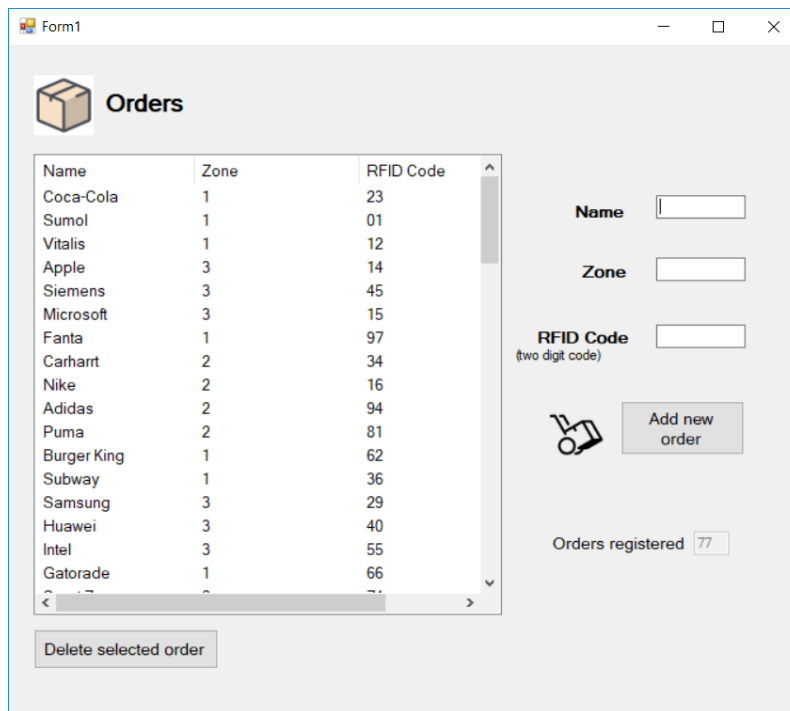


Figura 4.24: Interface da aplicação de controlo de ordens

Como podemos ver na figura 4.24, no interface da aplicação temos uma *listview* com todas as encomendas existentes na base de dados. Para submeter uma nova encomenda/ordem na base de dados é necessário preencher três campos: o nome da encomenda, a zona de destino no circuito e o código RFID (constituído por dois dígitos). Existe ainda a opção de apagar uma encomenda da base de dados.

Depois de todas as encomendas serem inseridas, é gerado um simples ficheiro XML com a informação referente a cada encomenda (figura 4.25). Este documento evita a perda de dados previamente submetidos, na abertura desta interface. Por isso, e já abordadas as características dos documentos XML, foi escolhido este tipo de ficheiro para armazenar os dados. Sempre que uma encomenda é adicionada ou apagada na interface de ordens, o mesmo acontece no ficheiro.

```
<?xml version="1.0"?>
- <Orders>
  - <Order>
    <Name>Coca-Cola</Name>
    <Zone>1</Zone>
    <RFID>23</RFID>
  </Order>
  - <Order>
    <Name>Sumol</Name>
    <Zone>1</Zone>
    <RFID>01</RFID>
  </Order>
  - <Order>
    <Name>Vitalis</Name>
    <Zone>1</Zone>
    <RFID>12</RFID>
  </Order>
```

Figura 4.25: Aspeto das ordens no ficheiro XML de armazenamento

Toda esta informação será depois importada para o TIA Portal, e ficará armazenada numa DB de nome Global_DB_Orders (figura 4.26). O PLC tem que aceder a estes dados para fazer uma correspondência entre o código RFID lido e os códigos existentes na DB. Se encontrar uma correspondência, o programa dará a ordem de destino para a zona correspondente, caso contrário, a paleta irá para a zona 0 (zona das encomendas rejeitadas).

Importante referir que o software de simulação Factory IO lê os código RFID com 4 dígitos mas, para o funcionamento desta aplicação, foram apenas considerados os últimos dois dígitos na atribuição e leitura de códigos. Esta decisão foi tomada apenas para que não fosse necessário inserir informação acerca de 9999 encomendas para o correto funcionamento da simulação.

Global_DB_Orders			
	Name	Data type	Start value
1	Static		
2	Order	Array[0..76] of Struct	
3	Order[0]	Struct	
4	Name	String	'Coca-Cola'
5	Zone	Int	1
6	RFID Code	Int	23
7	Order[1]	Struct	
8	Name	String	'Sumol'
9	Zone	Int	1
10	RFID Code	Int	01
11	Order[2]	Struct	
12	Name	String	'Vitalis'
13	Zone	Int	1
14	RFID Code	Int	12

Figura 4.26: Global DB com a informação referente às ordens

4.6 Geração automática de código para PLC

Nesta secção é explicado como é gerado o código PLC em função dos equipamentos e disposição dos mesmos no desenho em Visio. No TIA Portal são chamados os *function blocks* correspondentes a cada equipamento, e preenchidos os respetivos *inputs* e *outputs* do mesmo.

Na figura 4.27 temos o *function block* gerado, já com os respetivos *inputs* e *outputs* preenchidos, para o controlo do transportador indicado.

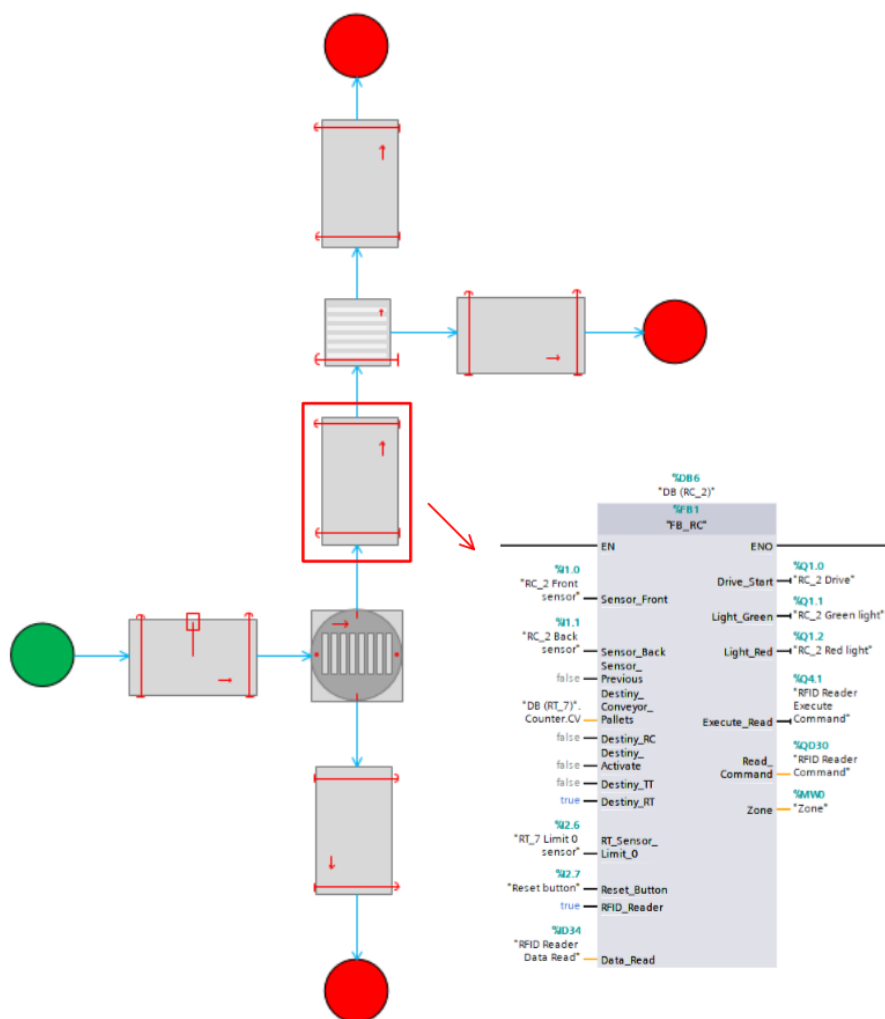


Figura 4.27: Exemplo de um bloco gerado para um determinado equipamento no layout criado

4.6.1 Interface da aplicação

Como foi referido em 4.3, foi desenvolvida uma aplicação .NET no Visual Studio, utilizando a linguagem de programação C#. Resumidamente, o propósito desta aplicação é extrair a informação necessária do desenho em Visio e, com isso, gerar documentos XML a serem importados no TIA Portal para controlo do ambiente de simulação correspondente, também gerado automaticamente no Factory IO.

Nesta secção é explicado o que faz cada botão existente na interface desta aplicação. Como podemos ver na figura 4.28 a interface é bastante simples, tendo o utilizador apenas que seleccionar o desenho Visio com o layout concluído e seleccionar os botões existentes. O botão *Select Visio file* abre uma caixa de diálogo para o utilizador procurar e seleccionar o desenho a ser importado.

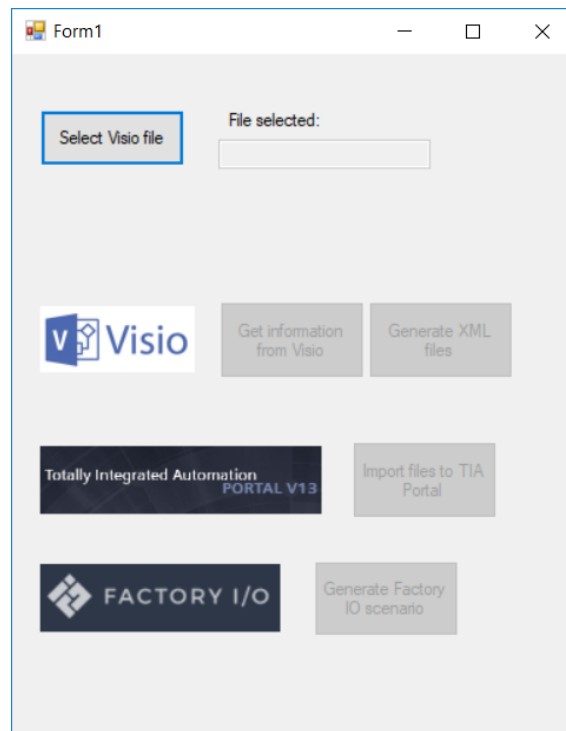


Figura 4.28: Interface .NET da aplicação desenvolvida no Visual Studio

Os restantes botões só se encontram disponíveis quando a informação necessária para realizar a sua tarefa também já se encontra disponível. A função de cada um é:

- *Get information from Visio*: retira e armazena a informação do circuito criado, como a quantidade de cada equipamento e os dados referentes a cada um, na aplicação .NET;
- *Generate XML files*: gera os documentos XML a serem importados no TIA Portal;
- *Import files to TIA Portal*: importa os documentos XML gerados para o TIA Portal, usando métodos provenientes da API *OpennessDemo*;
- *Generate Factory IO scenario*: gera o ficheiro XML correspondente ao cenário virtual em Factory IO e grava-o com a extensão *.factoryio*.

4.6.2 Geração de ficheiros XML

Com todos os dados relativos aos equipamentos adquiridos, o programa já pode gerar os ficheiros XML. O programa não cria os ficheiros XML de raiz. Em vez disso, acede a ficheiros XML já existentes, correspondentes a cada tipo de bloco, e insere os dados necessários, como referido em 4.3.

Estes documentos iniciais foram obtidos ao exportar blocos vazios do TIA Portal, usando as funcionalidades da API Demo Openness. Os blocos correspondentes às DBs instanciadas do controlo de cada equipamento não se encontram vazias aquando a exportação, contêm a informação sobre a programação do controlo de cada equipamento.

Apesar dos ficheiros XML serem diferentes consoante o tipo de bloco a que correspondem, todos eles são constituídos por dois elementos principais, como podemos ver na figura 4.29.

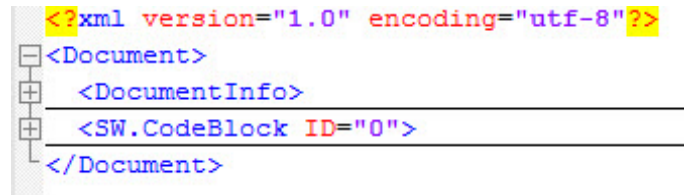


Figura 4.29: Elementos principais dos ficheiros XML provenientes do TIA Portal

No elemento *DocumentInfo* encontra-se informação sobre o documento, como a data e hora em que foi gerado, as configurações de exportação e os programas envolvidos e respetivas versões.

No elemento *SW.CodeBlock*, o seu nome varia consoante o tipo de bloco, como se encontra representado na figura 4.30. Ainda referente a este elemento, pode ter um ou dois elementos filhos. No caso dos ficheiros correspondentes a *Tag Tables* apenas existe um elemento filho de nome *AttributeList*. Nos ficheiros correspondentes aos restantes blocos (OB, *Global DB* e *Instance DB*) temos dois elementos filhos: *AttributeList* e *ObjectList*.

O elemento *AttributeList* contém a informação geral e atributos de um elemento, bem como dados correspondentes à interface com outros membros ou elementos, caso exista. Atributos que são definidos pelo sistema ou que têm um valor standard atribuído não aparecem neste ficheiro.

O elemento *ObjectList* contém informação acerca da configuração do bloco como *networks*, *tags*, linguagens utilizadas e itens. Contém ainda comentários associados ao bloco e áreas de compilação (*SW.CompileUnit*). Toda esta informação está especificada por IDs em numeração hexadecimal, que podem tomar valores entre 0 e 2147483647.



Figura 4.30: Exemplo dos elementos principais e o *CodeBlock* de cada tipo de ficheiro. 1) *Global DB*; 2) *Tag Table*; 3) *Instance DB*; 4) *OB*

4.6.3 Manipulação de ficheiros XML

Existem várias maneiras de ler e alterar um ficheiro XML a partir de uma aplicação em C#. No início da construção da aplicação foram consideradas várias alternativas de edição: *XmlTextReader* e *XmlTextWriter*, *XmlDocument* e *XmlSerialize*. Depois de uma análise das capacidades e análise de cada um, a primeira alternativa (*XmlTextReader* e *XmlTextWriter*) foi descartada pois, apesar de possibilitar uma rápida leitura e escrita de ficheiros XML, não tinha funcionalidades que permitissem a edição de ficheiros já existentes.

Apesar da alternativa *XmlSerialize* permitir uma melhor estruturação, e de forma menos trabalhosa, da informação a adicionar nos documentos XML através da serialização e deserialização, a aprendizagem da alternativa *XmlDocument* foi mais intuitiva e clara numa altura que a edição de documentos XML era uma técnica com a qual não estava familiarizado.

Foi então escolhida a alternativa *XmlDocument* pois permitia a leitura, edição, escrita e gravação de uma maneira simples. Permitia também a fácil navegação pelos elementos com o uso de ciclos e o encontro de elementos específicos através do *XPath*. Um *XPath* é a sintaxe usada para percorrer um documento XML e encontrar listas de elementos, elementos, textos e atributos, e encontra-se representado na tranche de código 4.1.

```
1 XmlNode node = xml_doc.SelectSingleNode("/Document/SW.Blocks.  
GlobalDB/AttributeList/Interface/def:Sections/def:Section/" +  
2 "def:Member/@Datatype", manager);
```

Tranche de Código 4.1: Exemplo de um *XPath* usado na aplicação desenvolvida para chegar a um elemento

Na tranche de código 4.2 temos o código, usando o *XmlDocument*, de criação do ficheiro XML que se encontra na figura 4.31.

```
1 XmlDocument ficheiro_xml = new XmlDocument(); // Criar ficheiro XML  
2  
3 XmlElement elemento_raiz = ficheiro_xml.CreateElement("Elemento_Raiz  
"); // Criar elemento raiz (Elemento_Raiz)  
4 ficheiro_xml.AppendChild(elemento_raiz); // Atribuir elemento_eaiz  
ao ficheiro XML  
5  
6  
7 XmlElement elemento_filho_1 = ficheiro_xml.CreateElement("  
Elemento_Filho_1"); // Criar elemento_filho_1  
8 elemento_raiz.AppendChild(elemento_filho_1); // Definir  
elemento_filho_1 como elemento filho do elemento_raiz  
9  
10 XmlText texto_1 = ficheiro_xml.CreateTextNode("Texto_1"); // Criar  
texto_1  
11 elemento_filho_1.AppendChild(texto_1); // atribuir texto_1 ao  
elemento_filho_1  
12  
13 elemento_filho_1.SetAttribute("Atributo_1", "Valor_Atributo_1"); //  
Atribuir atributo_1 ao elemento_filho_1  
14  
15 XmlElement elemento_filho_2 = ficheiro_xml.CreateElement("  
Elemento_Filho_2"); // Criar elemento_filho_2  
16 elemento_raiz.AppendChild(elemento_filho_2); // Definir  
elemento_filho_2 como elemento filho do elemento_raiz  
17
```

```

18     elemento_filho_2.SetAttribute("Atributo_1", "Valor_Atributo_1"); //
        Atribuir atributo_1 ao elemento_filho_2
19     elemento_filho_2.SetAttribute("Atributo_2", "Valor_Atributo_2"); //
        Atribuir atributo_2 ao elemento_filho_2
20
21     XmlElement elemento_filho_3 = ficheiro_xml.CreateElement("
        Elemento_Filho_3"); // Criar elemento_filho_3
22     elemento_raiz.AppendChild(elemento_filho_3); // Definir
        elemento_filho_2 como elemento filho do elemento_raiz
23
24     XmlElement elemento_filho_3_1 = ficheiro_xml.CreateElement("
        Elemento_Filho_1"); // Criar elemento_filho_filho_1
25     elemento_filho_3.AppendChild(elemento_filho_3_1); // Definir
        elemento_filho_3_1 como elemento filho do elemento_filho_3
26
27     XmlElement elemento_filho_3_2 = ficheiro_xml.CreateElement("
        Elemento_Filho_2"); // Criar elemento_filho_filho_2
28     elemento_filho_3.AppendChild(elemento_filho_3_2); // Definir
        elemento_filho_3_2 como elemento filho do elemento_filho_3
29
30     ficheiro_xml.Save(@"C:\Users\andlop\Desktop\auxiliar.xml"); //
        Gravar ficheiro XML

```

Tranche de Código 4.2: Código C# usado na criação de um ficheiro XML

```

<?xml version="1.0"?>
- <Elemento_Raiz>
    <Elemento_Filho_1 Atributo_1="Valor_Atributo_1">Texto_1</Elemento_Filho_1>
    <Elemento_Filho_2 Atributo_1="Valor_Atributo_1" Atributo_2="Valor_Atributo_2"/>
- <Elemento_Filho_3>
    <Elemento_Filho_1/>
    <Elemento_Filho_2/>
    </Elemento_Filho_3>
</Elemento_Raiz>

```

Figura 4.31: Ficheiro XML criado com o código da figura 4.2

Não foi possível ter acesso aos *schemas* dos ficheiros XML correspondentes a cada tipo de bloco, fornecidos pela Siemens. Logo, para que a alteração destes ficheiros fosse correta e não originasse erros aquando a importação no TIA Portal, o método seguido para estudar cada estrutura foi o seguinte:

1. Exportação de um documento XML correspondente a cada tipo de bloco com este vazio, recorrendo à API Demo Openness;
2. Utilização da ferramenta do Visual Studio (figura 4.32), que permite a geração de um *schema* através da submissão de um ficheiro XML, para a análise dos *schemas* de cada bloco. Importante referir que esta ferramenta gera os *schemas* com alguns erros, pelo que serviu apenas como ponto de partida para a análise da estrutura de cada ficheiro;
3. Leitura da documentação fornecida pela Siemens em [30] com a explicação das funções de alguns elementos e atributos existentes nos ficheiros XML;
4. Edição manual de um ficheiro XML e importação do mesmo no TIA Portal com o objetivo de analisar os resultados das alterações efetuadas;

5. Exportação do documento XML corresponde a cada bloco com alterações semelhantes às efetuadas manualmente, mas desta vez efetuadas diretamente no TIA Portal;
6. Verificação final através da comparação dos documentos XML alterados manualmente e exportados de um bloco alterado diretamente no TIA Portal.

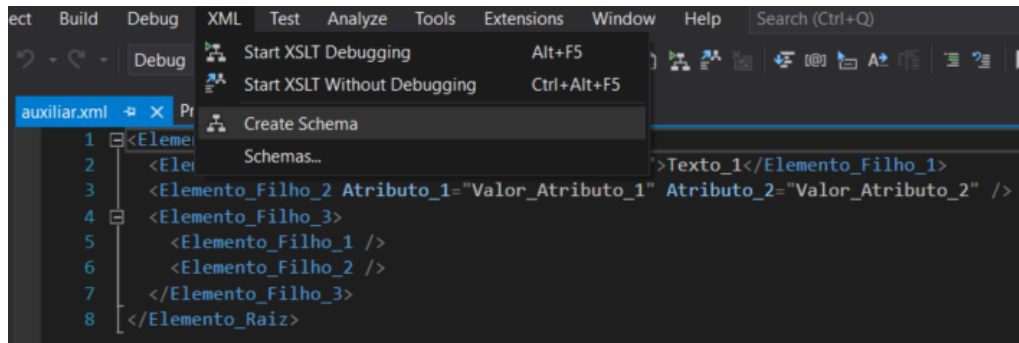


Figura 4.32: Ferramenta do Visual Studio que permite a geração de *schemas*

Entre 4.6.3.1 e 4.6.3.4 é explicado detalhadamente o processo por trás da edição de cada tipo de documento XML. São também abordadas algumas técnicas de programação usadas para conseguir executar certas operações e aceder a determinados dados, tanto na aplicação em C# como no TIA Portal.

4.6.3.1 Ficheiro XML de *Tag Tables*

O documento XML correspondente a uma *tag table* sem informação, retirado do TIA Portal através da API Demo Openness encontra-se na figura 4.33. No funcionamento da aplicação, são gerados dois ficheiros do tipo *Tag Table*: um para os *Inputs* e outro para os *Outputs*. Estes diferem apenas no nome e na informação que vão conter.

```
<?xml version="1.0" encoding="UTF-8"?>
- <Document>
  <Engineering version="V15.1"/>
  + <DocumentInfo>
    - <SW.Tags.PlcTagTable ID="0">
      - <AttributeList>
        <Name>Inputs</Name>
      </AttributeList>
    </SW.Tags.PlcTagTable>
  </Document>
```

Figura 4.33: Exemplo de um ficheiro XML de uma *Tag Table* sem informação

No ficheiro correspondente aos *inputs* vão existir variáveis correspondentes a:

- Sensores iniciais e finais de transportadores de rolos;
- Sensores de entrada de mesas de transferências;
- Sensores de entrada, saída e de deteção das posições das mesas rotativas;
- Código RFID lido de cada encomenda;

- Botões de *start*, *stop* e *reset*.

No ficheiro correspondente aos *outputs* vão existir variáveis correspondentes a:

- Drives de transportadores de rolos;
- Drives (em frente, para a esquerda e direita) de mesas de transferências;
- Drives (em frente, para trás e de rotação) de mesas rotativas;
- Sinalizadores luminosos de todos os equipamentos;
- Ordem para o leitor RFID executar uma leitura, e que tipo de leitura;
- Controlo do *emitter*;
- Zona correspondente ao código RFID lido (esta variável é armazenada na memória do simulador PLC).

No documento XML uma variável numa *tag table* é definida pelos parâmetros existentes na figura 4.34. Como foi referido em 4.6.2, no elemento *ObjectList* temos dados de configuração do bloco, onde existem dois IDs, pelo que, os IDs de *SW.Tags.PlcTag* incrementam em intervalos de 3.

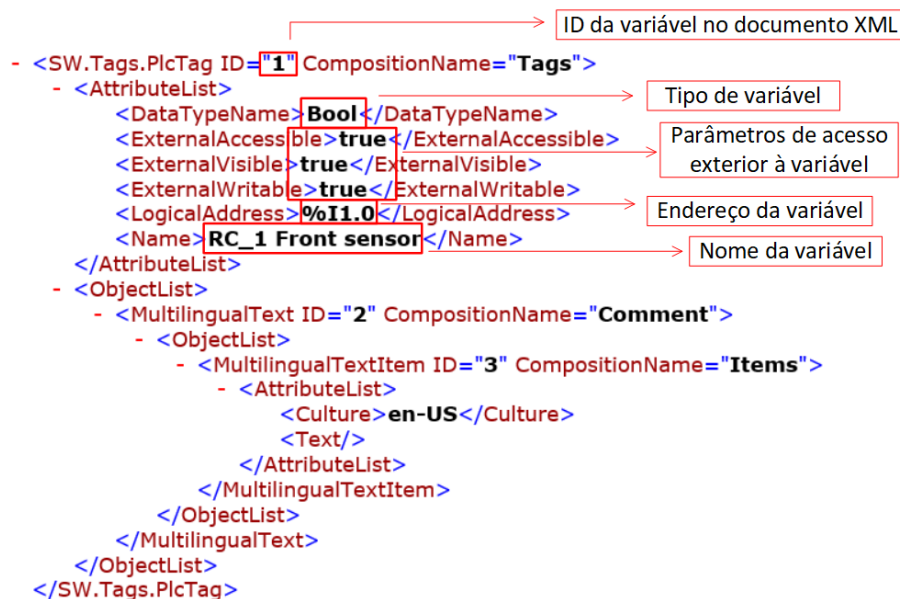


Figura 4.34: Parâmetros para definição de uma variável numa *tag table*

Na aplicação em C#, consoante os equipamentos existentes, é gerada uma lista com todos os *inputs* e *outputs*. Para além das variáveis que não estão associadas a nenhum equipamento, como a zona, controlo do *emitter*, botões e variáveis relacionadas com o leitor RFID, cada equipamento tem, individualmente, as seguintes *tags* associadas:

- Transportador de rolos:
 - *Input*: sensor final;

- *Input*: sensor inicial (opcional);
- *Output*: drive;
- *Output*: luz verde sinalizadora;
- *Output*: luz vermelha sinalizadora;
- Mesa de transferência:
 - *Input*: sensor de entrada;
 - *Output*: drive para a direção em frente;
 - *Output*: drive para a direção da direita (opcional);
 - *Output*: drive para a direção da esquerda (opcional);
 - *Output*: luz verde sinalizadora;
 - *Output*: luz vermelha sinalizadora.
- Mesa rotativa:
 - *Input*: sensor de entrada;
 - *Input*: sensor de saída;
 - *Input*: sensor de detecção da posição da mesa a 0 graus;
 - *Input*: sensor de detecção da posição da mesa a 90 graus;
 - *Output*: drive para movimentação da paleta em frente;
 - *Output*: drive para movimentação da paleta para trás;
 - *Output*: drive de rotação da mesa;
 - *Output*: luz verde sinalizadora;
 - *Output*: luz vermelha sinalizadora.

A alteração do documento inicial é feita criando uma nova *SW.Tags.PlcTag* para cada *input* ou *output* existente na lista. Posteriormente são adicionados os respectivos dados (tipo, nome, parâmetros de acesso e endereço).

4.6.3.2 Ficheiro XML de *Instance* DBs

O documento XML correspondente às *Instance* DBs é um caso especial, uma vez que, no seu estado inicial, não se encontra vazio. Este ficheiro é conseguido ao exportar uma *Instance* DB de um *function block* que contém a programação para controlo de um dos equipamentos. Nele já se encontram todas as variáveis utilizadas no respetivo FB.

A aplicação vai apenas copiar este documento, alterando o seu nome e número da *instance* DB. Por exemplo, o documento XML correspondente a uma *instance* DB de um transportador de rolos tem que ser copiado o mesmo número de vezes que a quantidade deste equipamento, e o mesmo se verifica para os restantes. Ou seja, cada equipamento tem que ter a sua própria *instance* DB associada.

Na figura 4.35 temos uma perspetiva geral do ficheiro XML correspondente a uma *instance* DB e na figura 4.36 temos as variáveis de entrada, saída e auxiliares dessa *instance* DB, idênticas às existentes no respetivo FB.

```

<?xml version="1.0" encoding="UTF-8"?>
- <Document>
  <Engineering version="V15.1"/>
  + <DocumentInfo>
  - <SW.Blocks.InstanceDB ID="0">
    - <AttributeList>
      <AutoNumber>true</AutoNumber>
      <CodeModifiedDate ReadOnly="true">2020-08-17T16:56:05.8439017Z</CodeModifiedDate>
      <CompileDate ReadOnly="true">2020-08-17T16:56:31.2219761Z</CompileDate>
      <CreationDate ReadOnly="true">2020-08-17T16:56:05.7939298Z</CreationDate>
      <DBAccessibleFromOPCUA ReadOnly="true">true</DBAccessibleFromOPCUA>
      <DownloadWithoutReinit ReadOnly="true">false</DownloadWithoutReinit>
      <HeaderAuthor/>
      <HeaderFamily/>
      <HeaderName/>
      <HeaderVersion>0.1</HeaderVersion>
      <InstanceOfName>FB_RC</InstanceOfName>
      <InstanceOfNumber ReadOnly="true">1</InstanceOfNumber>
      <InstanceOfType>FB</InstanceOfType>
    + <Interface>
      <InterfaceModifiedDate ReadOnly="true">2020-07-27T14:46:26.7561551Z</InterfaceModifiedDate>
      <IsConsistent ReadOnly="true">true</IsConsistent>
      <IsKnowHowProtected ReadOnly="true">false</IsKnowHowProtected>
      <IsRetainMemResEnabled ReadOnly="true">false</IsRetainMemResEnabled>
      <MemoryLayout>Optimized</MemoryLayout>
      <MemoryReserve ReadOnly="true">100</MemoryReserve>
      <ModifiedDate ReadOnly="true">2020-08-17T16:56:05.8439017Z</ModifiedDate>
      <Name>InstanceDB_RC</Name>
      <Number>1</Number>
      <ParameterModified ReadOnly="true">2020-07-27T14:46:26.7561551Z</ParameterModified>
      <ProgrammingLanguage>DB</ProgrammingLanguage>
      <StructureModified ReadOnly="true">2020-07-27T14:46:26.7561551Z</StructureModified>
      <UDABlockProperties ReadOnly="true"/>
      <UDAEnableTagReadback ReadOnly="true">false</UDAEnableTagReadback>
    </AttributeList>
    + <ObjectList>
    </SW.Blocks.InstanceDB>
  </Document>

```

Nome da Instance DB

Número da Instance DB

Function block correspondente

Figura 4.35: Ficheiro XML correspondente a uma Instance DB

```

- <Section Name="Input">
  + <Member Name="Sensor_Front" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="Sensor_Back" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="Sensor_Previous" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="Destiny_Conveyor_Pallets" Accessibility="Public" Remanence="NonRetain" Datatype="Int">
  + <Member Name="Destiny_RC" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="Destiny_Activate" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="Destiny_TT" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="Destiny_RT" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="RT_Sensor_Limit_0" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="Reset_Button" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="RFID_Reader" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="Data_Read" Accessibility="Public" Remanence="NonRetain" Datatype="DInt">
</Section>
- <Section Name="Output">
  + <Member Name="Drive_Start" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="Light_Green" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="Light_Red" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="Execute_Read" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="Read_Command" Accessibility="Public" Remanence="NonRetain" Datatype="DInt">
  + <Member Name="Zone" Accessibility="Public" Remanence="NonRetain" Datatype="Int">
</Section>
<Section Name="InOut"/>
- <Section Name="Static">
  + <Member Name="f_trig_1" Accessibility="Public" Datatype="F_TRIG" Version="1.0">
  + <Member Name="r_trig_1" Accessibility="Public" Datatype="R_TRIG" Version="1.0">
  + <Member Name="Counter" Accessibility="Public" Remanence="Retain" Datatype="CTUD_INT" Version="1.0">
  + <Member Name="SR_1" Accessibility="Public" Remanence="Retain" Datatype="Bool">
  + <Member Name="Mem_Light" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="Mem_Hold" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
  + <Member Name="Digit_1" Accessibility="Public" Remanence="NonRetain" Datatype="Int">
  + <Member Name="Digit_2" Accessibility="Public" Remanence="NonRetain" Datatype="Int">
  + <Member Name="RFID_Code" Accessibility="Public" Remanence="NonRetain" Datatype="Int">
  + <Member Name="Brand" Accessibility="Public" Remanence="NonRetain" Datatype="String">
  + <Member Name="SR" Accessibility="Public" Remanence="NonRetain" Datatype="Bool">
</Section>

```

Figura 4.36: Variáveis existentes na Instance DB para controlo de um transportador de rolos

4.6.3.3 Ficheiro XML de *Global DBs*

No caso dos ficheiros XML correspondentes a *global DBs*, o documento no estado inicial já possui alguma informação. Todas as *global DBs* geradas são constituídas por *arrays of struct* e outras variáveis (em alguns casos). Um *array of struct* é um *array* onde cada elemento contém mais do que um tipo de variável. Porém, todos os elementos são constituídos pelas mesmas variáveis.

Global_DB_RC				
	Name	Data type	Start value	
1	Static			
2	RC	Array[0..9] of Struct		
3	RC[0]	Struct		
4	ID	Int	1	
5	Name	String	'RC_1'	
6	Back_sensor	Bool	True	
7	Origin	Int	0	
8	Destiny	Int	6	
9	RFID Reader	Bool	True	
10	RC[1]	Struct		
11	ID	Int	6	
12	Name	String	'RC_6'	
13	Back_sensor	Bool	False	
14	Origin	Int	1	
15	Destiny	Int	11	
16	RFID Reader	Bool	False	

Figura 4.37: Exemplo da Global_DB_RC gerada, vista no TIA Portal

A aplicação gera quatro ficheiros XML correspondentes a *Global DBs*:

- **Global_DB_RC**: contém informação sobre todos os transportadores de rolos (figura 4.37);
- **Global_DB_TT**: contém informação sobre todas as mesas de transferência;
- **Global_DB_RT**: contém informação sobre todas as mesas rotativas;
- **Global_DB_Orders**: contém o nome, código RFID e zona de destino de todas as encomendas inseridas na aplicação de controlo de ordens;
- **Global_DB_Zones**: contém a rota para cada destino, mais precisamente qual a direção que cada paleta deve tomar nas mesas de transferência e mesas rotativas por onde passa para chegar a um determinado destino.

Seguindo uma lógica semelhante à geração de *instance DBs*, a informação existente no documento inicial de uma *global DB* são a designação e tipo de cada variável, sem o valor das mesmas, como se encontra representado na figura 4.38. No caso de um *array of struct* é necessário definir a designação dos elementos (todos os elementos vão ter o mesmo nome mas índices diferentes) e as variáveis existentes dentro deles. Inicialmente, a *length* do *array* é zero, sendo a aplicação que a altera, consoante o número de transportadores de rolos, mesas de transferência, mesas rotativas, ordens ou zonas, dependendo da *global DB*.

```

<?xml version="1.0" encoding="UTF-8"?>
- <Document>
  <Engineering version="V15.1"/>
  + <DocumentInfo>
  - <SW.Blocks.GlobalDB ID="0">
    - <AttributeList>
      <AutoNumber>true</AutoNumber>
      <CodeModifiedDate ReadOnly="true">2020-06-30T15:29:36.3353863Z</CodeModifiedDate>
      <CompileDate ReadOnly="true">2020-06-30T15:29:43.4902526Z</CompileDate>
      <CreationDate ReadOnly="true">2020-06-18T11:53:00.1552779Z</CreationDate>
      <DBAccessibleFromOPCUA ReadOnly="true">true</DBAccessibleFromOPCUA>
      <DownloadWithoutReinit ReadOnly="true">false</DownloadWithoutReinit>
      <HeaderAuthor/>
      <HeaderFamily/>
      <HeaderName/>
      <HeaderVersion>0.1</HeaderVersion>
    - <Interface>
      - <Sections xmlns="http://www.siemens.com/automation/Openness/SW/Interface/v3">
        - <Section Name="Static">
          - <Member Name="RC" Accessibility="Public" Remanence="NonRetain" Datatype="Array[0..0] of Struct">
            + <AttributeList>
              + <Member Name="ID" Datatype="Int">
              + <Member Name="Name" Datatype="String">
              + <Member Name="Back_sensor" Datatype="Bool">
              + <Member Name="Origin" Datatype="Int">
              + <Member Name="Destiny" Datatype="Int">
              + <Member Name="RFID Reader" Datatype="Bool">
            </Member>
          </Section>
        </Sections>
      </Interface>

```

Designação do elemento

Tipo de variável

Variáveis existentes dentro de cada elemento do array (nome e tipo)

Figura 4.38: Parte do ficheiro XML correspondente à Global_DB_RC

Na figura 4.39 temos a informação das variáveis correspondentes aos IDs de cada transportador no ficheiro XML da Global_DB_RC com informação referente a um layout criado no Visio. O índice encontra-se definido no atributo *Path* do elemento *Subelement* e o ID correspondente encontra-se no texto do elemento filho *StartValue*. A definição das restantes variáveis segue a mesma lógica.

```

<Section Name="Static">
- <Member Name="RC" Accessibility="Public" Remanence="NonRetain" Datatype="Array[0..9] of Struct">
  + <AttributeList>
  - <Member Name="ID" Datatype="Int">
    + <AttributeList>
    - <Comment>
      <MultiLanguageText Lang="en-US">Conveyor ID</MultiLanguageText>
    </Comment>
    - <Subelement Path="0">
      <StartValue>1</StartValue>
    </Subelement>
    - <Subelement Path="1">
      <StartValue>6</StartValue>
    </Subelement>
    - <Subelement Path="2">
      <StartValue>24</StartValue>
    </Subelement>
    - <Subelement Path="3">
      <StartValue>29</StartValue>
    </Subelement>

```

Figura 4.39: Zona do documento XML onde se encontram definidos os IDs de cada elemento da Global_DB_RC

Dos quatro ficheiros gerados, o que corresponde à Global_DB_Zones tem uma particularidade. Esta *Global DB* tem um *array of struct* dentro de outro, como se encontra representado na figura 4.40.

Global_DB_Zones			
	Name	Data type	Start value
1	Static		
2	Number of Zones	Int	3
3	Zone	Array[0..2] of Struct	
4	Zone[0]	Struct	
5	Zone Number	Int	0
6	Number of Equipments	Int	1
7	Direction Points	Array[0..1] of Struct	
8	Direction Points[0]	Struct	
9	Equipment ID	Int	7
10	Direction	Char	'R'
11	Direction Points[1]	Struct	
12	Equipment ID	Int	0
13	Direction	Char	'L'
14	Zone[1]	Struct	
15	Zone Number	Int	1
16	Number of Equipments	Int	2
17	Direction Points	Array[0..1] of Struct	
18	Direction Points[0]	Struct	
19	Equipment ID	Int	7
20	Direction	Char	'L'
21	Direction Points[1]	Struct	
22	Equipment ID	Int	37
23	Direction	Char	'R'

Figura 4.40: Global_DB_Zones vista no TIA Portal

A colocação dos valores das variáveis *Equipment_ID* e *Direction* é diferente (figura 4.41). É seguida a lógica de uma matriz, onde a área destinada ao valor é definida por dois números: o primeiro corresponde ao índice do *array Zone* e o segundo corresponde ao índice do *array Direction Points*.

```

- <Section Name="Static">
+ <Member Name="Number of Zones" Accessibility="Public" Remanence="NonRetain"
  Datatype="Int">
- <Member Name="Zone" Accessibility="Public" Remanence="NonRetain"
  Datatype="Array[0..2] of Struct">
+ <AttributeList>
+ <Member Name="Zone Number" Datatype="Int">
+ <Member Name="Number of Equipments" Datatype="Int">
- <Member Name="Direction Points" Datatype="Array[0..1] of Struct">
+ <AttributeList>
- <Member Name="Equipment ID" Datatype="Int">
+ <AttributeList>
- <Subelement Path="0,0">
  <StartValue>7</StartValue>
</Subelement>
- <Subelement Path="1,0">
  <StartValue>7</StartValue>
</Subelement>
- <Subelement Path="1,1">
  <StartValue>37</StartValue>
</Subelement>
- <Subelement Path="2,0">
  <StartValue>7</StartValue>
</Subelement>
- <Subelement Path="2,1">
  <StartValue>37</StartValue>
</Subelement>

```

Figura 4.41: Definição dos *Equipment_ID* no documento XML da Global_DB_Zones

4.6.3.4 Ficheiro XML de OB

O ficheiro XML correspondente ao OB é o ficheiro mais extenso e complexo. Nele temos que detalhar qual o *function block* a ser chamado, em que *newtork* se encontra e os seus *inputs* e *outputs*. O ficheiro inicial "limpo" não tem nenhuma informação acerca

do layout nele. Apenas informação geral como a linguagem de programação usada, nome, número e tipo de bloco, e outras configurações.

No documento XML os dois primeiros IDs de *ObjectList* dizem respeito a configurações do bloco, nomeadamente aos comentários existentes e linguagem usada (figura 4.42). Na primeira *network* do OB encontra-se programada a lógica para controlo do *emitter* com os botões. Esta lógica foi programada em *ladder* usando um *set reset* (figura 4.43) e a codificação correspondente no ficheiro XML encontra-se na figura 4.44.

```
<?xml version="1.0" encoding="UTF-8"?>
- <Document>
  <Engineering version="V15.1"/>
  + <DocumentInfo>
  - <SW.Blocks.OB ID="0">
    + <AttributeList>
    - <ObjectList>
      - <MultilingualText ID="1" CompositionName="Comment">
        - <ObjectList>
          - <MultilingualTextItem ID="2" CompositionName="Items">
            - <AttributeList>
              <Culture>en-US</Culture>
              <Text/>
            </AttributeList>
          </MultilingualTextItem>
        </ObjectList>
      </MultilingualText>
```

Figura 4.42: Dois primeiros IDs no ficheiro XML do OB

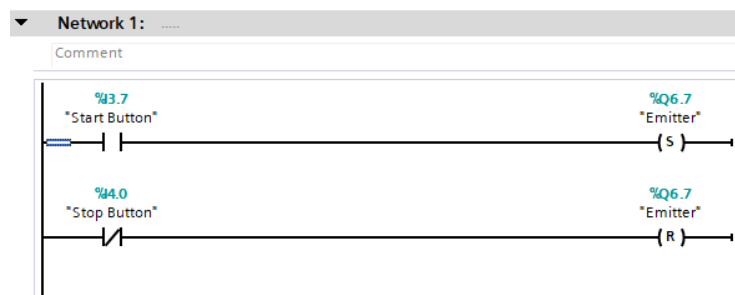


Figura 4.43: Network com a lógica de controlo do *emitter* no TIA Portal

```
- <Parts>
- <Access Scope="GlobalVariable" UID="21">
  - <Symbol>
    <Component Name="Start Button"/>
    <Address Informative="true" Area="Input" Type="Bool" BitOffset="40"/>
  </Symbol>
</Access>
- <Access Scope="GlobalVariable" UID="22">
  - <Symbol>
    <Component Name="Emitter"/>
    <Address Informative="true" Area="Input" Type="Bool" BitOffset="64"/>
  </Symbol>
</Access>
- <Access Scope="GlobalVariable" UID="23">
  - <Symbol>
    <Component Name="Stop Button"/>
    <Address Informative="true" Area="Input" Type="Bool" BitOffset="40"/>
  </Symbol>
</Access>
- <Access Scope="GlobalVariable" UID="24">
  - <Symbol>
    <Component Name="Emitter"/>
    <Address Informative="true" Area="Input" Type="Bool" BitOffset="64"/>
  </Symbol>
</Access>
<Part Name="Contact" UID="25"/>
<Part Name="Coil" UID="26"/>
<Part Name="Contact" UID="27">
  <Negated Name="operand"/>
</Part>
<Part Name="RCoil" UID="28"/>
</Parts>
```

Tags utilizadas

Tipos de contactos associados às tags

Como as tags se conectam

```
- <Wires>
- <Wire UID="29">
  <Powerrail/>
  <NameCon Name="in" UID="25"/>
  <NameCon Name="in" UID="27"/>
</Wire>
- <Wire UID="30">
  <IdentCon UID="21"/>
  <NameCon Name="operand" UID="25"/>
</Wire>
- <Wire UID="31">
  <NameCon Name="out" UID="25"/>
  <NameCon Name="in" UID="26"/>
</Wire>
- <Wire UID="32">
  <IdentCon UID="22"/>
  <NameCon Name="operand" UID="26"/>
</Wire>
- <Wire UID="33">
  <IdentCon UID="23"/>
  <NameCon Name="operand" UID="27"/>
</Wire>
- <Wire UID="34">
  <NameCon Name="out" UID="27"/>
  <NameCon Name="in" UID="28"/>
</Wire>
- <Wire UID="35">
  <IdentCon UID="24"/>
  <NameCon Name="operand" UID="28"/>
</Wire>
</Wires>
```

Figura 4.44: Codificação no ficheiro XML do controlo do *emitter*

Para o controlo dos equipamentos é necessário atribuir os *function blocks* correspondentes a cada um. Para isso, como referido anteriormente, é necessário associar uma *instance DB* a cada *function block* que se chama. Para além disso é necessário especificar quais são as *tags* a serem usadas e onde se encontram ligadas. Toda essa informação tem que ser colocada no ficheiro XML. Na figura 4.45 encontra-se um exemplo da colocação de um simples *function block* numa OB.

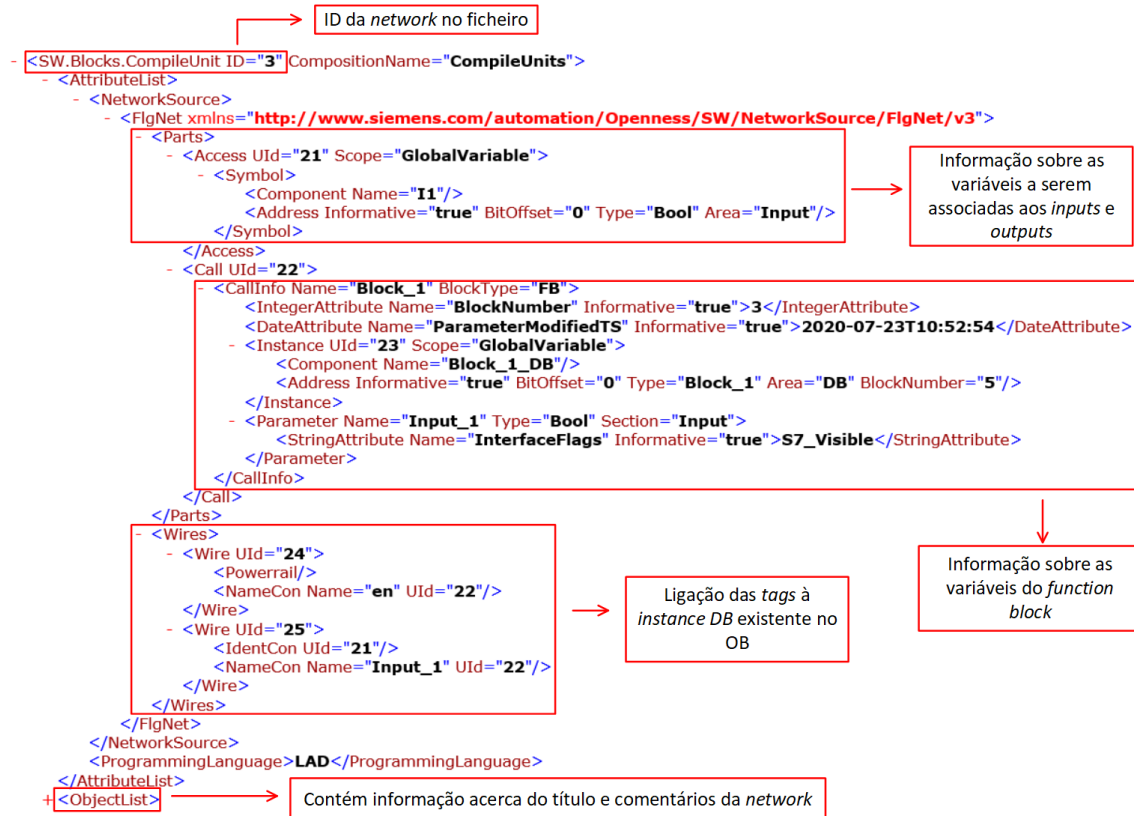


Figura 4.45: Estrutura do ficheiro XML de um OB

Tendo em conta a estrutura do ficheiro XML de um OB, a sua edição na aplicação C# é feita por fases. Sabendo que a existência de cada equipamento vai corresponder ao chamamento do respetivo *function block*, foi definida a colocação de um por *network*. As diferentes fases são então:

1. Definição de uma nova *network*;
2. Definição das variáveis a serem usadas;
3. Associação do *function block* a uma determinada *instance DB*;
4. Definição das ligações entre variáveis locais e globais;
5. Colocação da informação do elemento *ObjectList*;

Definição de uma nova *network*

A criação de uma nova *network* passa pela adição de um novo elemento, *SW.Blocks.CompileUnit*. Este elemento encontra-se na localização *Document/SW.Blocks.OB/ ObjectList*. O ID deste elemento, tal como nos restantes ficheiros XML, encontra-se em

numeração hexadecimal. Para realizar esta operação foi criado e usado o método *EnterNetwork*, cujo funcionamento encontra-se explicado na figura 4.47.

Na figura 4.46 encontra-se representado um ficheiro XML correspondente a um OB. Nesse *organization block* existem várias *networks* criadas.

```
<?xml version="1.0" encoding="UTF-8"?>
- <Document>
  <Engineering version="V15.1"/>
  + <DocumentInfo>
    - <SW.Blocks.OB ID="0">
      + <AttributeList>
        - <ObjectList>
          + <MultilingualText ID="1" CompositionName="Comment">
            - <SW.Blocks.CompileUnit ID="3" CompositionName="CompileUnits">
              - <AttributeList>
                - <NetworkSource>
                  - <FlgNet xmlns="http://www.siemens.com/automation/Openness/SW/NetworkSource/FlgNet/v3">
                    + <Parts>
                      + <Wires>
                    </FlgNet>
                  </NetworkSource>
                  <ProgrammingLanguage>LAD</ProgrammingLanguage>
                </AttributeList>
              + <ObjectList>
                </SW.Blocks.CompileUnit>
              + <SW.Blocks.CompileUnit ID="8" CompositionName="CompileUnits">
              + <SW.Blocks.CompileUnit ID="D" CompositionName="CompileUnits">
              + <SW.Blocks.CompileUnit ID="12" CompositionName="CompileUnits">
              + <SW.Blocks.CompileUnit ID="17" CompositionName="CompileUnits">
              + <SW.Blocks.CompileUnit ID="1C" CompositionName="CompileUnits">
              + <SW.Blocks.CompileUnit ID="21" CompositionName="CompileUnits">
              + <SW.Blocks.CompileUnit ID="26" CompositionName="CompileUnits">
            </SW.Blocks.CompileUnit>
          </ObjectList>
        </AttributeList>
      </SW.Blocks.OB>
    </DocumentInfo>
  </Document>

```

Networks definidas no OB

Figura 4.46: Elementos no ficheiro XML correspondentes a várias *networks*

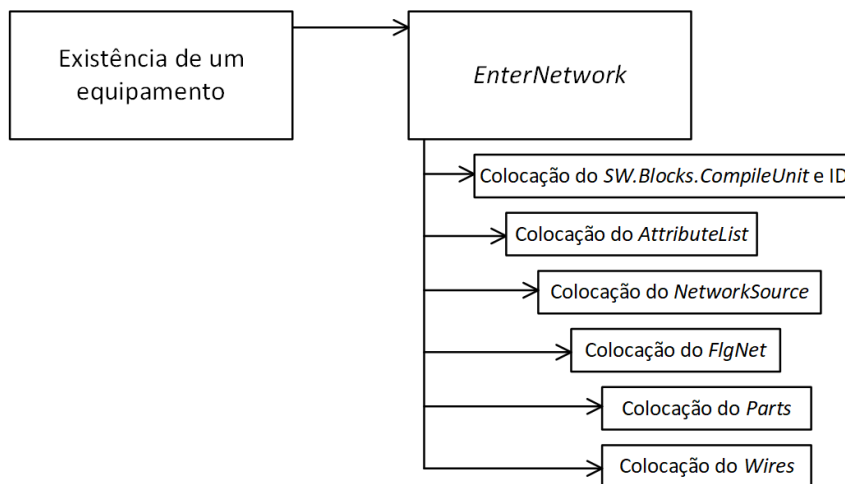


Figura 4.47: Diagrama de funcionamento do método *EnterNetwork*

Definição das variáveis a serem usadas

Na colocação das variáveis a serem usadas como *inputs* e *outputs* dos *function blocks* foram definidas cinco tipos:

- Booleanos provenientes de *tag tables*;
- Números inteiros provenientes de *tag tables*;
- Números inteiros provenientes de *global DBs*;
- Números inteiros provenientes de *instance DBs*;
- Booleanos simples (apenas um *true* ou *false*).

Para isso foram criados e usados métodos na aplicação C# uma vez que os campos necessários a colocar no ficheiro XML dependiam da origem da variável. Todas estas variáveis são inseridas através do elemento *Access*, na localização *Document/SW.Blocks.OB /ObjectList/SW.Blocks.CompileUnit/AttributeList/NetworkSource/FlgNet/Parts*, acompanhado de dois atributos:

- *Scope*: significa se a variável tem origem noutra bloco ou *tag table* (*GlobalVariable*), ou se é simplesmente colocada no *function block* (*LiteralConstant*);
- *Uid*: ID da variável no *function block* em questão. Este ID pode tomar valores entre 21 e 2147483647, e é representado em numeração decimal.

Na figura 4.48 encontram-se representados todos os elementos, associados a uma determinada variável, necessários a definir para a correta geração do ficheiro XML.

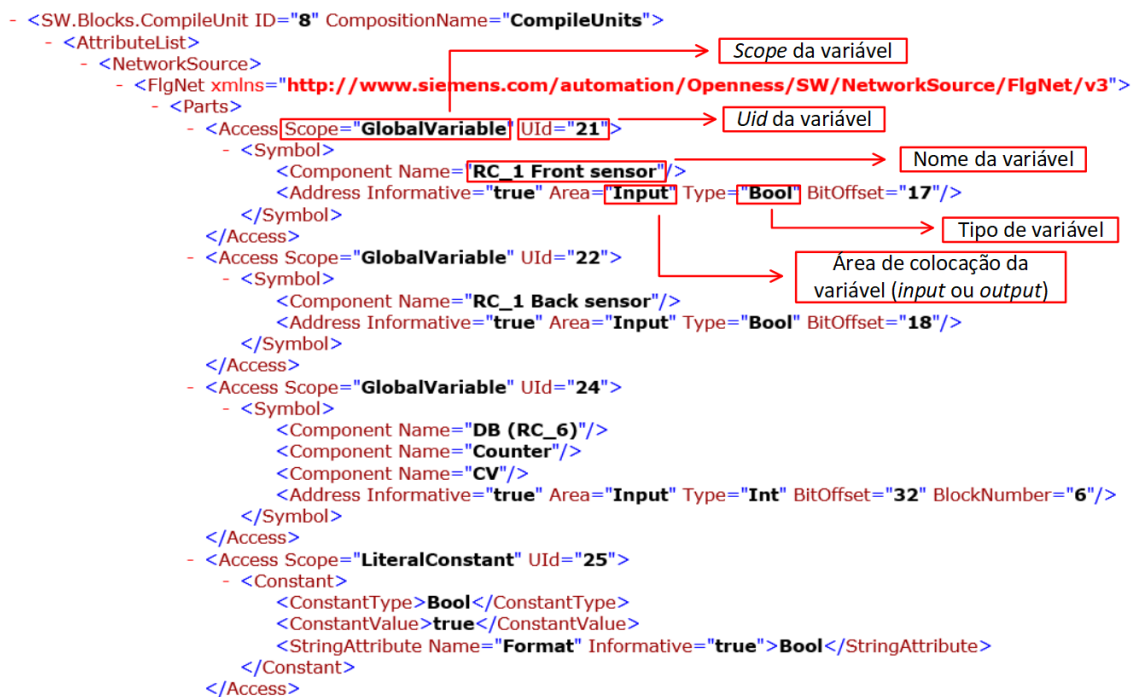


Figura 4.48: Elementos no ficheiro XML correspondentes à definição das variáveis

Colocação de variável no <i>function block</i>	Tipo de variável	Origem	Método usado
	Número inteiro	Tag table	EnterInt_IO
		Instance DB	EnterInt_InstanceDB
		Global DB	EnterInt_GlobalDB
	Booleano	Tag table	EnterBool_IO
		Nenhuma (simples true ou false)	EnterSimpleBool

Figura 4.49: Lógica para escolha dos métodos de colocação de variáveis

Depois de selecionado o método a usar, de acordo com o diagrama da figura 4.49, é apenas necessário colocar o nome da variável que irá ser colocada. O método percorre

a lista de *inputs* ou *outputs*, dependendo o tipo de variável global a inserir, até encontrar a variável correspondente e coloca toda a informação correspondente e necessária no documento XML.

No diagrama da figura 4.50 encontra-se representada a lógica seguida pela aplicação .NET na colocação de variáveis no documento XML.



Figura 4.50: Processo de colocação de informação de uma variável no *function block*

Associação do *function block* a uma determinada *instance DB*

É necessário cada equipamento ter a sua própria *instance DB* associada, apesar destas, no caso do mesmo tipo de equipamentos, serem provenientes dos mesmos *function blocks*. No documento XML tem que estar definido o nome, número, e qual o *function block* correspondente à *instance DB* em questão. Posteriormente é feita uma listagem de todas as variáveis existentes nela. Todos estes dados encontra-se representados, no ficheiro XML, na figura 4.51.

Para isso é criado o elemento *Call* na mesma localização que o elemento *Access* referido anteriormente. Na aplicação C# foi criado e usado o método *EnterFBInformation* (figura 4.52) onde se encontram listadas todas as variáveis do FB através da criação do elemento *Parameter* para cada uma.

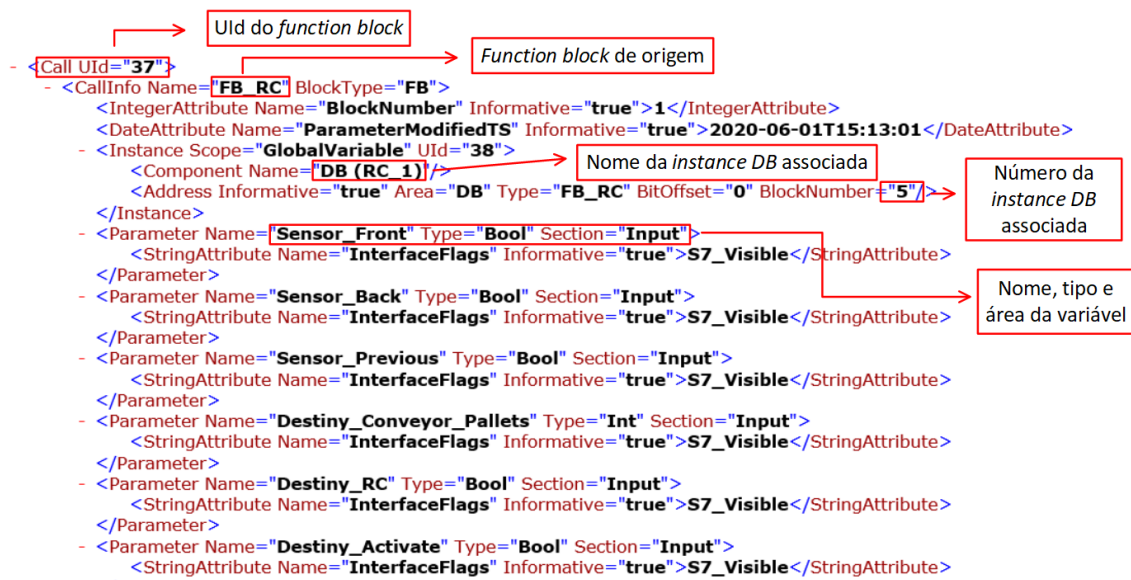


Figura 4.51: Elementos no ficheiro XML correspondentes à informação do *function block* e *instance DB*

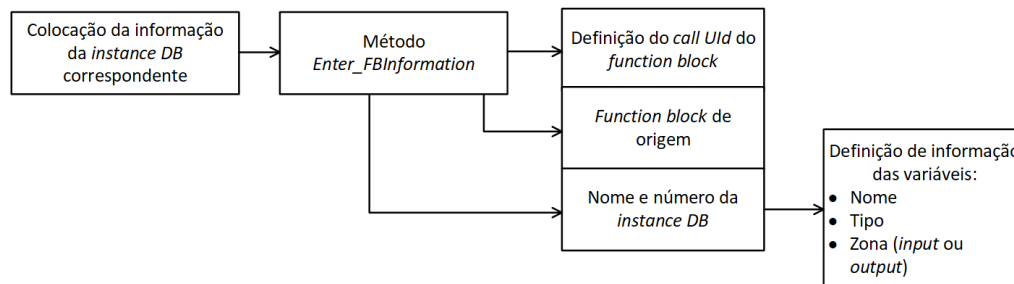


Figura 4.52: Lógica da colocação de informação acerca do *function block* e *instance DB*

Definição das ligações entre variáveis locais e globais

As conexões entre as variáveis a serem atribuídas aos comandos de cada FB utilizado encontram-se especificadas no elemento *Wires*. Este elemento tem como propósito especificar todas as ligações existentes numa *network*, quer contenha um FB ou não. Na figura 4.53 encontra-se a estrutura dos elementos filhos do elemento *Wires*.

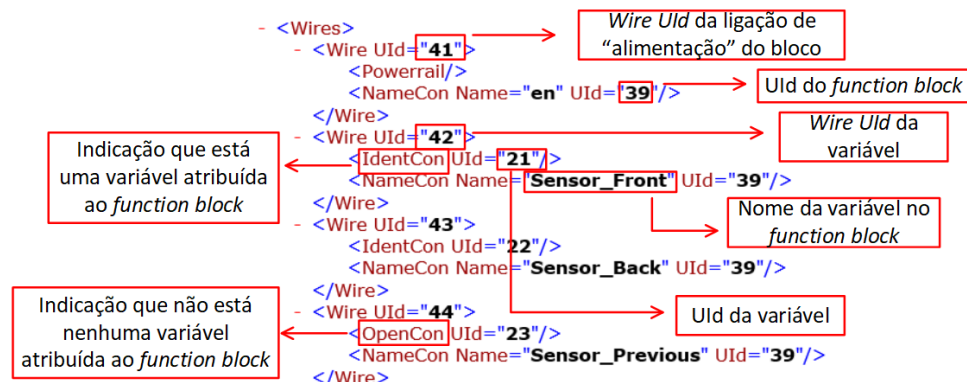


Figura 4.53: Estrutura do elemento *Wires*

É necessário existir um elemento *Wire*, elemento filho de *Wires*, para cada variável local existente. A ordem pela qual se encontram no documento XML tem que ser a mesma pela qual se encontram definidas no *function block*. O *Wire Uid* é o ID de uma ligação, e não pode ter o mesmo valor que o *Uid* de uma variável.

Na figura 4.54 encontra-se representada a lógica seguida pela aplicação .NET na colocação de informação no elemento *Wires*.

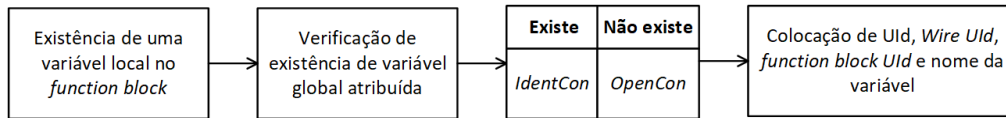


Figura 4.54: Lógica da colocação de informação do elemento *Wires*

Colocação da informação do elemento *ObjectList*

Como foi referido em 4.6.3.4, todos os elementos correspondentes a uma *network* têm o elemento *ObjectList* como elemento filho. Este elemento contém sempre a mesma informação (figura 4.55), variando apenas, de *network* para *network*, os IDs referentes à informação do título e dos comentários. Na aplicação C# esta informação é inserida usando método criado *EnterConfigurationElements*.

```

- <ObjectList>
- <MultilingualText ID="4" CompositionName="Comment">
- <ObjectList>
- <MultilingualTextItem ID="5" CompositionName="Items">
- <AttributeList>
<Culture>en-US</Culture>
<Text/>
</AttributeList>
</MultilingualTextItem>
</ObjectList>
</MultilingualText>
- <MultilingualText ID="6" CompositionName="Title">
- <ObjectList>
- <MultilingualTextItem ID="7" CompositionName="Items">
- <AttributeList>
<Culture>en-US</Culture>
<Text/>
</AttributeList>
</MultilingualTextItem>
</ObjectList>
</MultilingualText>
</ObjectList>
  
```

Figura 4.55: Informação contida no elemento *ObjectList* de uma *network*

4.6.4 Importação dos ficheiros XML para o TIA Portal

Depois de retirada a informação do desenho do Visio e gerados os ficheiros XML, os mesmos estão prontos a serem importados para o TIA Portal, mais especificamente para o projeto modelo anteriormente referido. Esta operação é executada pelo botão *Import files to TIA Portal* da interface da aplicação. O código utilizado para realizar esta importação foi retirado da classe *TiaOpennessHelper*, que acompanha a API Demo Openness e são utilizados certos métodos:

1. *StartTIA()*: Inicia o TIA Portal;
2. *OpenProject(string ProjectPath)*: Abre o projeto modelo, de nome *TIA_Project*, depois de inserido a localização do mesmo;

3. *ImportTagTable(string file)*: Importa as *tag tables* correspondentes aos *inputs* e *outputs*, depois de aceder à zona *PLC Tags* do respetivo hardware no projeto;
4. *ImportBlocks(string file)*: Importa as *global DBs*, *instance DBs* e *OB*, depois de aceder à zona *Program blocks* do respetivo hardware.

Estes métodos são utilizados na aplicação desenvolvida, sendo necessário especificar a localização, no computador, de todos os documentos XML.

4.6.5 *Function Blocks* desenvolvidos no TIA Portal

Para garantir o correto funcionamento dos diferentes equipamentos que podem existir no layout foi criado um *function block* para cada. Nestes FBs encontra-se o código para a funcionamento do equipamento, individualmente. Foi escolhida esta abordagem porque permite a reutilização de código, já testado e verificado, para o controlo de um determinado componente sempre que seja necessário.

Os FBs foram criados diretamente no TIA Portal, e a aplicação em C# apenas os "chama". Em cada bloco é necessário definir, pelo menos, dois tipos de variáveis locais: os *inputs* e os *outputs*. Para além destas, ainda temos as variáveis locais estáticas, temporais e constantes. Enquanto que as primeiras têm/podem ter outras variáveis globais atribuídas, as segundas só existem dentro do respetivo *function block* e normalmente funcionam como variáveis auxiliares.

Na figura 4.56 temos um exemplo das variáveis locais de um FB. Na figura 4.57 temos um exemplo de código usando essas mesmas variáveis.

Function Block			
	Name	Data type	Default value
1	▼ Input		
2	Input 1	Bool	false
3	Input 2	Bool	false
4	Input 3	Int	0
5	Input 4	Int	0
6	▼ Output		
7	Output 1	Bool	false
8	Output 2	Bool	false

Figura 4.56: Exemplo de variáveis locais de um *function block*

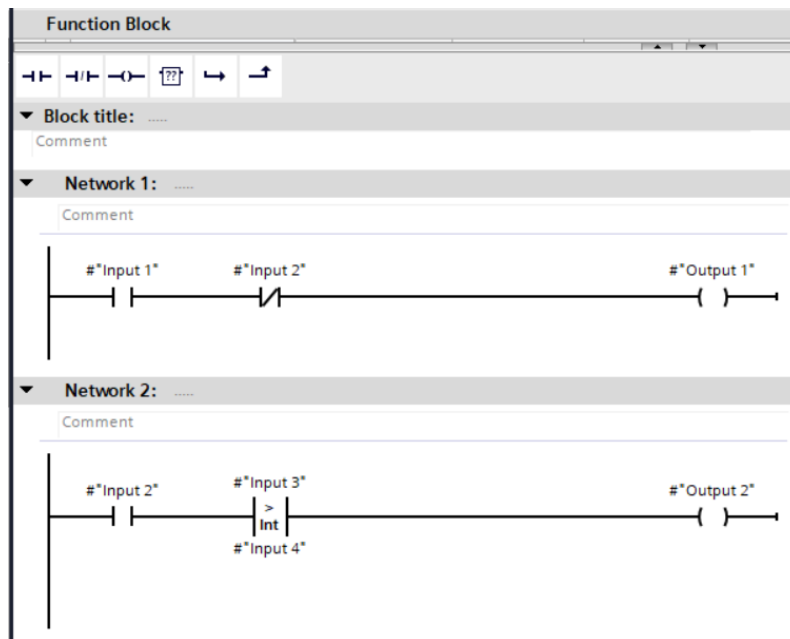


Figura 4.57: Exemplo da programação de um *function block*

FB_RC

Function block criado para o controlo dos transportadores de rolos. Possui um contador que incrementa quando é detetada uma paleta no sensor de entrada (*Sensor_Back*) ou no sensor de saída do transportador de rolos anterior (*Sensor_Previous*), e reduz quando é detetada uma transição negativa do sensor de saída (*Sensor_Front*) do transportador em questão. O equipamento é ativado quando existe pelo menos uma paleta no mesmo e nenhuma condição de *stand by* da paleta se verifica (na programação PLC é representada pela variável *Mem_Hold*).

Este tipo de equipamento pode ter ou não um leitor RFID associado. No caso de ter, a variável *RFID_Reader* toma o valor *true* e o programa executa a programação que permite executar leituras e retirar os últimos dois dígitos de cada código lido. Posteriormente, é feita a correspondência entre estes dois dígitos e a informação existente na *Global_DB.Orders*. A zona de destino da encomenda é guardada na memória do PLC, para que os restantes equipamento a possam aceder (código em SCL, tranche de código 4.3).

Sempre que um determinado FB deste tipo de equipamento é instanciado a aplicação C# preenche os *inputs* e *outputs* com as variáveis globais correspondentes (figura 4.58).

No *grafcet* existente na figura 4.59 encontra-se representada a lógica sequencial de operações seguida para o correto funcionamento deste tipo de equipamentos.

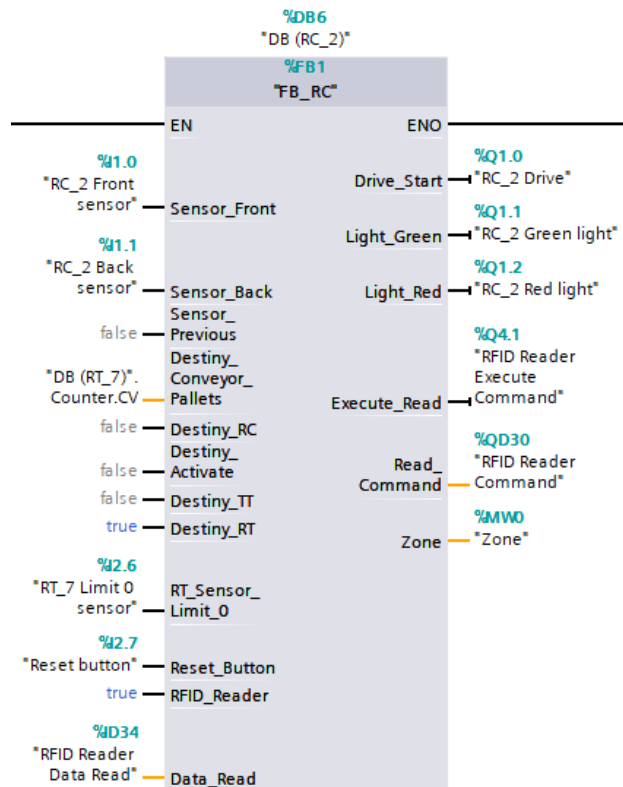


Figura 4.58: Exemplo de um FB_RC instanciado, visto no TIA Portal

```

1 IF #SR THEN
2     #Zone := 0;
3     #Brand := '';
4     #exists := false;
5     FOR #i := 0 TO "Global_DB_Orders".Size DO
6         IF (#RFID_Code = "Global_DB_Orders".Order[#i].RFID Code) THEN
7             #Zone := "Global_DB_Orders".Order[#i].Zone;
8             #Brand := "Global_DB_Orders".Order[#i].Name;
9             FOR #j := 0 TO ("Global_DB_Zones".Number of Zones - 1) DO
10                 IF (#Zone = "Global_DB_Zones".Zone[#j].Zone Number)
11                     THEN
12                         #Zone := #Zone;
13                         #exists := true;
14                     END_IF;
15                 END_FOR;
16                 IF (#exists = false) THEN
17                     #Zone := 0;
18                 END_IF;
19             END_FOR;
20         END_IF;

```

Tranche de Código 4.3: Programação PLC para encontrar a zona de destino consoante o código RFID

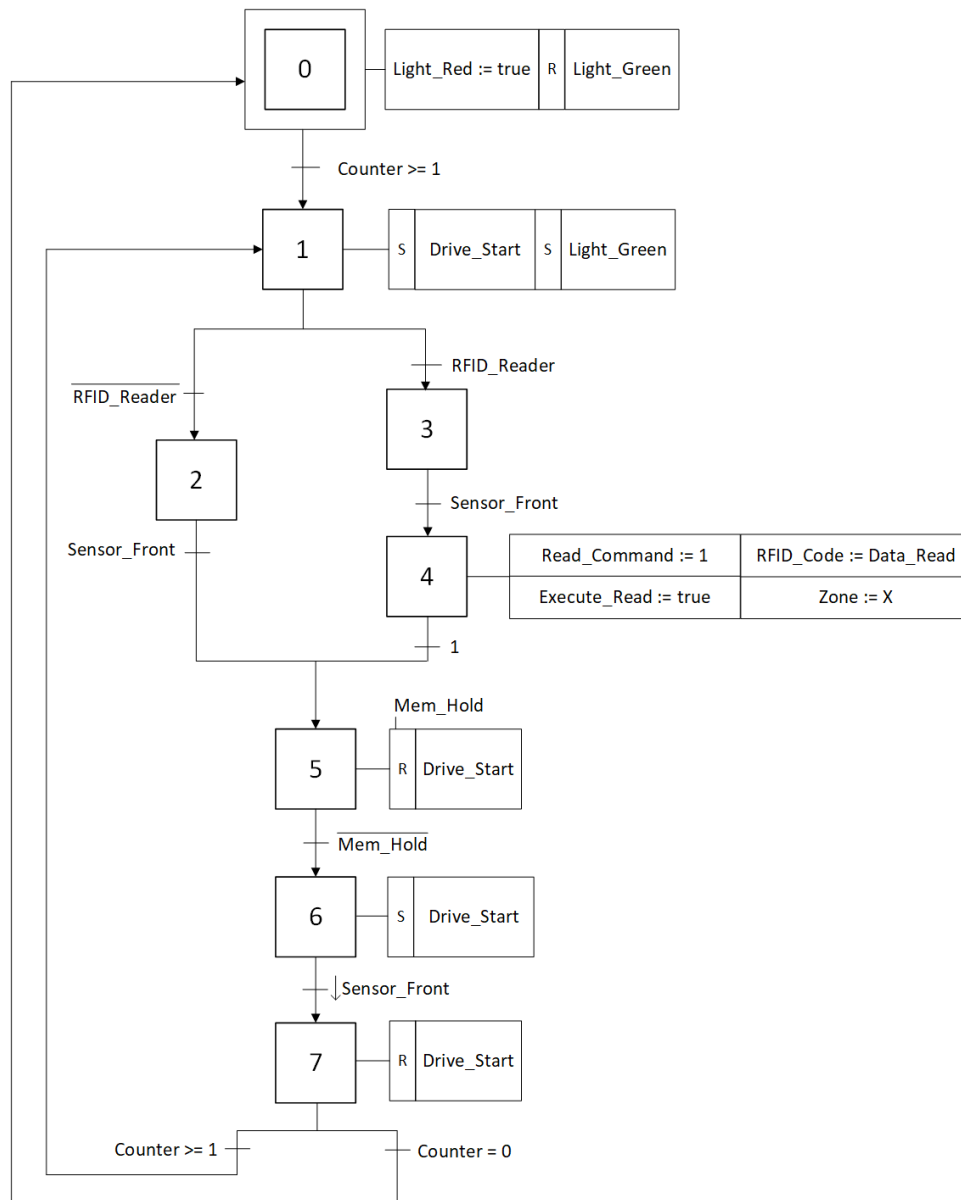


Figura 4.59: Grafcet funcional do FB_RC

Os significados das variáveis locais usadas são:

- **Sensor_Front:** sensor no final do transportador;
- **Sensor_Back:** sensor no início do transportador (nem todos os transportadores têm este sensor);
- **Sensor_Previous:** sensor da frente do transportador anterior;
- **Destiny_Conveyor_Pallets:** quantidade de paletes em trânsito no equipamento seguinte;
- **Destiny_RC:** indica se o equipamento seguinte é um transportador de rolos;
- **Destiny_Activate:** indica se o equipamento seguinte está em funcionamento;
- **Destiny_TT:** indica se o equipamento seguinte é uma mesa de transferência;

- **Destiny_RT**: indica se o equipamento seguinte é uma mesa rotativa;
- **RT_Sensor_Limit_0**: indica se a mesa rotativa seguinte (se for o caso) se encontra na posição de 0 graus, disponível para receber paletes;
- **Reset_Button**: botão que reinicia o contador;
- **RFID_Reader**: existência de um leitor RFID;
- **Data_Read**: código lido pelo leitor RFID;
- **Drive_Start**: ativação do movimento do transportador;
- **Light_Green**: sinalizador verde do transportador;
- **Light_Red**: sinalizador vermelho do transportador;
- **Execute_Read**: execução de uma leitura por parte do leitor RFID;
- **Read_Command**: modo de leitura do leitor RFID;
- **Zone**: zona de destino da encomenda em questão.

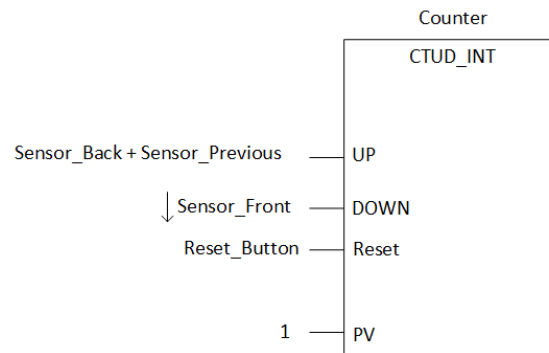


Figura 4.60: Estrutura auxiliar (contador) do grafcet do FB_RC

Na figura 4.60 encontra-se a estrutura auxiliar para o correto funcionamento do *function block*. A lógica da ativação da variável auxiliar *Mem_Hold*, que indica quando uma paleta deve esperar que o equipamento seguinte esteja disponível para recebê-la, encontra-se na figura 4.61.

Equipamento de destino	Estado da variável	Código PLC
Transportador de rolos <i>Destiny_RC</i>	<i>Set</i>	$\text{Destiny_Conveyor_Pallets} \geq 2$
	<i>Reset</i>	$\text{Destiny_Conveyor_Pallets} \leq 1 + \text{Destiny_Activate}$
Mesa de transferência <i>Destiny_TT</i>	<i>Set</i>	$\text{Destiny_Conveyor_Pallets} \geq 1 * \overline{\text{Destiny_Activate}} * \text{Sensor_Front}$
	<i>Reset</i>	$\text{Destiny_Conveyor_Pallets} = 0 + \text{Destiny_Activate}$
Mesa rotativa <i>Destiny_RT</i>	<i>Set</i>	$\text{Sensor_Front} * (\text{Destiny_Conveyor_Pallets} \geq 1 + \overline{\text{RT_Sensor_Limit_0}})$
	<i>Reset</i>	$\text{RT_Sensor_Limit_0} * \text{Destiny_Conveyor_Pallets} = 0$

Figura 4.61: Lógica da ativação da variável *Mem_Hold*

FB_TT

Function block criado para o controlo das mesas de transferência. Tal como o FB_RC, possui um contador para monitorizar a quantidade de paletes existentes no equipamento. Este contador é incrementado quando é detetada uma paleta na entrada da mesa (*Sensor*) e reduz quando deteta uma transição negativa de um dos sensores de entrada dos transportadores de rolos alimentados pela mesa (*Sensor_Front*, *Sensor_Right* ou *Sensor_Left*).

O processo de funcionamento deste equipamento começa quando uma paleta é detetada no sensor de entrada. Os rolos da mesa são ativados para posicionar a paleta no centro da mesma, para que a possível mudança de direção ocorra sem problemas. A paleta encontra-se corretamente posicionada quando é detetada uma transição negativa do sensor de entrada. Nesta altura a mesa já recebeu a informação sobre a zona de destino da paleta e já tem definida a direção que esta tem que tomar. Se for em frente, os rolos ativam-se e a paleta segue em frente. Se for para a esquerda ou direita as correntes da mesa sobem e levam a paleta para a respetiva direção. O movimento acaba quando a paleta ultrapassa totalmente o sensor de entrada do transportador de rolos que a recebe.

A decisão da direção a seguir pela paleta é tomada com base na zona de destino da mesma. Depois da mesa de transferência ter acesso a essa informação, consoante o seu próprio ID, vai encontrar a correspondência entre a zona e a direção a seguir (tranche de código 4.4). Estes dados encontram-se na *Global_DB_Zones*.

Sempre que um determinado FB deste tipo de equipamento é instanciado a aplicação C# preenche os *inputs* e *outputs* com as variáveis globais correspondentes (figura 4.62).

No *grafcet* existente na figura 4.63 encontra-se representada a lógica sequencial de operações seguida para o correto funcionamento deste tipo de equipamentos.

```
1 FOR #p := 0 TO ("Global_DB_Zones"."Number of Zones" - 1) DO
2     IF #Zone_Orders[0] = "Global_DB_Zones".Zone[#p]."Zone Number" THEN
3         FOR #i := 0 TO ("Global_DB_Zones".Zone[#p]."Number of Equipments"
4             " - 1) DO
5             IF "Global_DB_Zones".Zone[#p]."Direction Points"[#i]."
6                 Equipment ID" = #RotativeTable_ID THEN
7                 #Direction := "Global_DB_Zones".Zone[#p]."Direction
8                     Points"[#i].Direction;
9             END_IF;
10        END_FOR;
11    END_IF;
12 END_FOR;
```

Tranche de Código 4.4: Programação em SCL que associa a direção que uma paleta deve seguir com base no destino da mesma

A informação da direção que uma paleta deve tomar numa determinada mesa de transferência é enviada para os diferentes blocos instanciados de mesas de transferência quando a paleta passa pelo leitor RFID. Logo é preciso guardar a zona de destino de cada uma, para quando chegarem à mesa de transferência o programa saber que direção devem seguir. A programação deste processo encontra-se na tranche de código 4.5, que vai buscar os dados à *Global_DB_Zones*.

```
1 #f_trig_er(CLK := #Execute_Read);
2
3 IF #f_trig_er.Q THEN
4     #SR_er := true;
5 END_IF;
```

```

6
7 IF #Reset_Button THEN
8     #element := 0;
9     #Direction := ' ';
10    #SR_er := false;
11    FOR #i := 0 TO 10 DO
12        #Zone_Orders[#i] := 10;
13    END_FOR;
14 END_IF;
15
16 IF #SR_er THEN
17     FOR #l := 0 TO ("Global_DB_Zones"."Number of Zones" - 1) DO
18         IF #Zone = "Global_DB_Zones".Zone[#l].Zone Number THEN
19             FOR #k := 0 TO ("Global_DB_Zones".Zone[#l].Number of
20                 Equipments" - 1) DO
21                 IF "Global_DB_Zones".Zone[#l].Direction Points"[#k].
22                     Equipment ID" = #RotativeTable_ID THEN
23                     #Zone_Orders[#element] := #Zone;
24                     #element := #element + 1;
25                     #SR_er := false;
26                 END_IF;
27             END_FOR;
28         END_IF;
29     END_FOR;
30
31     #SR_er := false;
32 END_IF;
33
34 IF #Sensor_Limit_90 AND (#f_trig_sensor_front.Q OR #f_trig_sensor_back.Q
35     ) THEN
36     FOR #j := 0 TO #element DO
37         #Zone_Orders[#j] := #Zone_Orders[#j + 1];
38     END_FOR;
39     #element := #element - 1;
40 END_IF;

```

Tranche de Código 4.5: Programação em SCL que grava o destino de cada paleta que vai passar por uma determinada mesa de transferência ou mesa rotativa

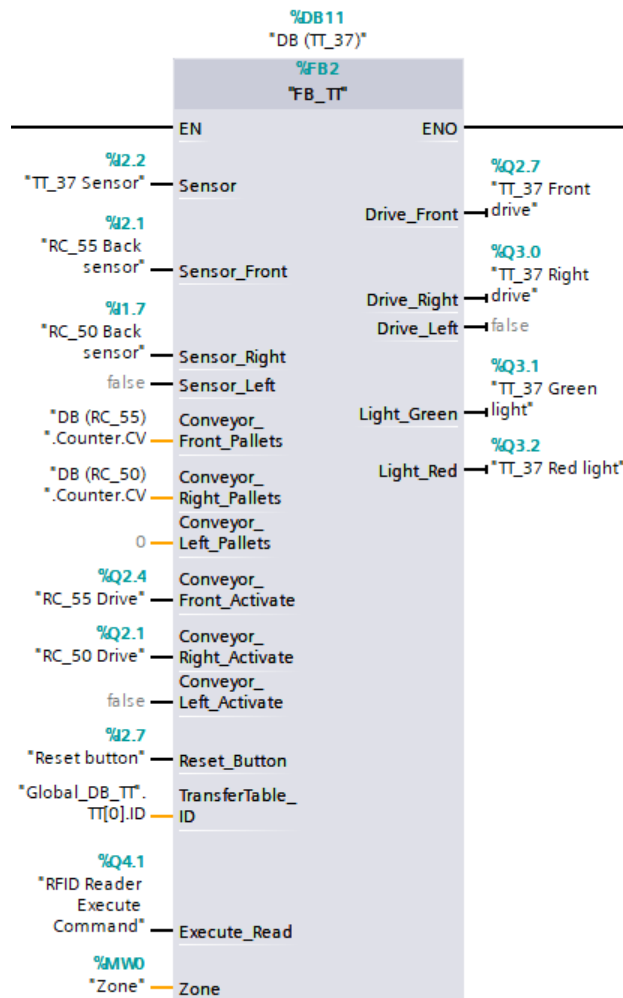


Figura 4.62: Exemplo de um FB_TT instanciado, visto no TIA Portal

Os significados das variáveis locais usadas são:

- **Sensor:** sensor na entrada da mesa de transferência;
- **Sensor_Direction:** sensor no início do transportador que se encontra na direção referida em relação à mesa de transferência;
- **Conveyor_Direction_Pallets:** quantidade de paletes existentes no transportador na direção referida;
- **Conveyor_Direction_Activate:** indica se o transportador na direção referida se encontra em funcionamento;
- **Reset_Button:** botão que reinicia o contador e os SR_Direction;
- **TransferTable_ID:** ID da transfer table em questão;
- **Execute_Read:** execução de uma leitura por parte do leitor RFID;
- **Zone:** zona de destino da encomenda em questão;
- **Drive_Direction:** ativação do movimento da mesa de transferência na direção referida, podendo ser através de rolos (frente) ou correntes (esquerda e direita);

- **Light_Green:** sinalizador verde da mesa de transferência;
- **Light_Red:** sinalizador vermelho da mesa de transferência.

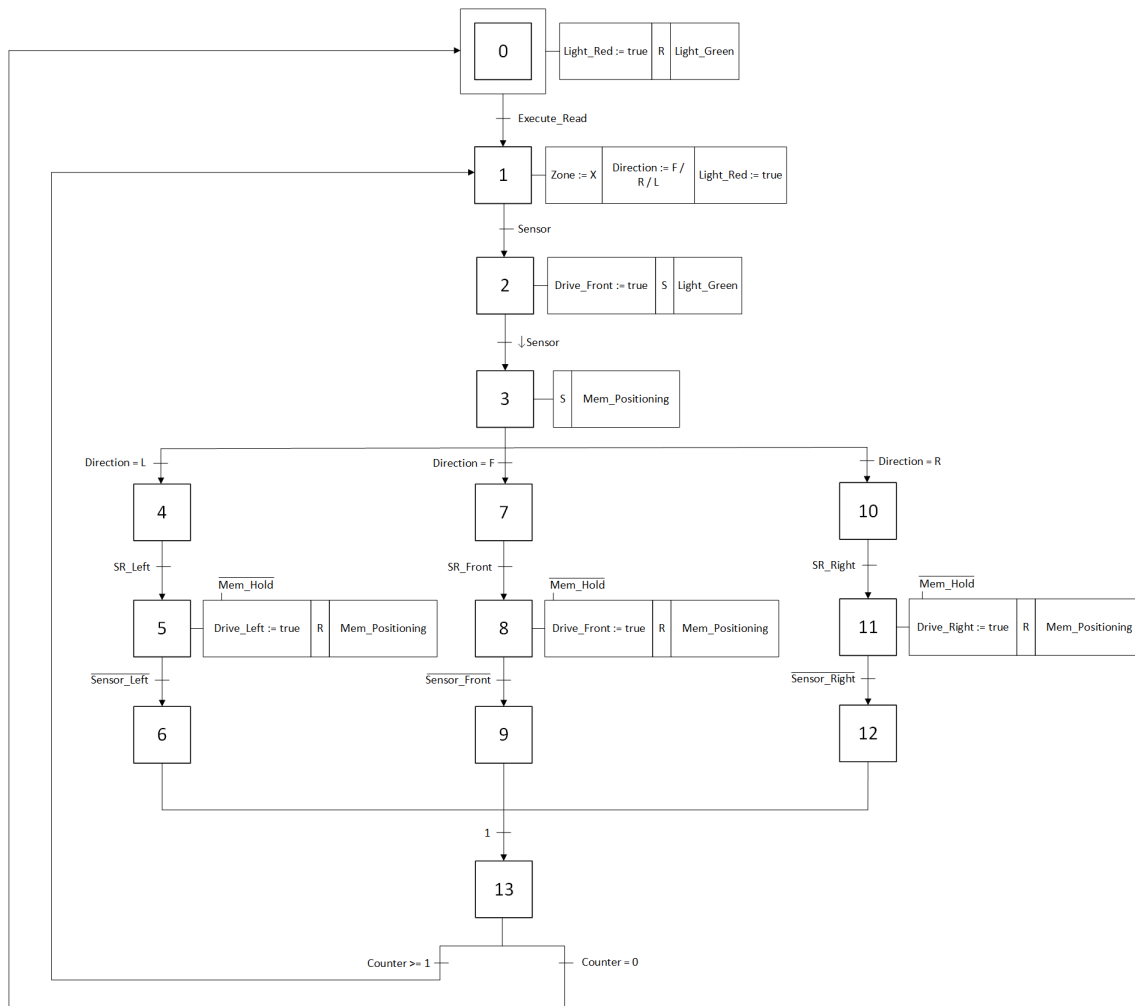


Figura 4.63: Grafcet funcional do FB-TT

Existem ainda várias variáveis auxiliares importantes, cuja definições se encontram na figura 4.64, na programação deste bloco:

- **Mem_Hold:** tal como no FB-RC, esta variável serve para reter as paletes na mesa de transferência enquanto o equipamento de destino não se encontra preparado para as receber;
- **Direction:** caractere que especifica a direção a seguir na mesa de transferência;
- **Mem_Positioning:** variável que indica quando uma paleta se encontra corretamente posicionada na mesa;
- **SR_Direction:** *set reset* que aciona os rolos da mesa (no caso da paleta ir em frente) ou as correntes para a esquerda ou direita. Existem três variáveis deste tipo, uma para cada direção possível;
- **Activate_Direction:** variável que indica quando uma paleta está pronta para tomar uma determinada direção.

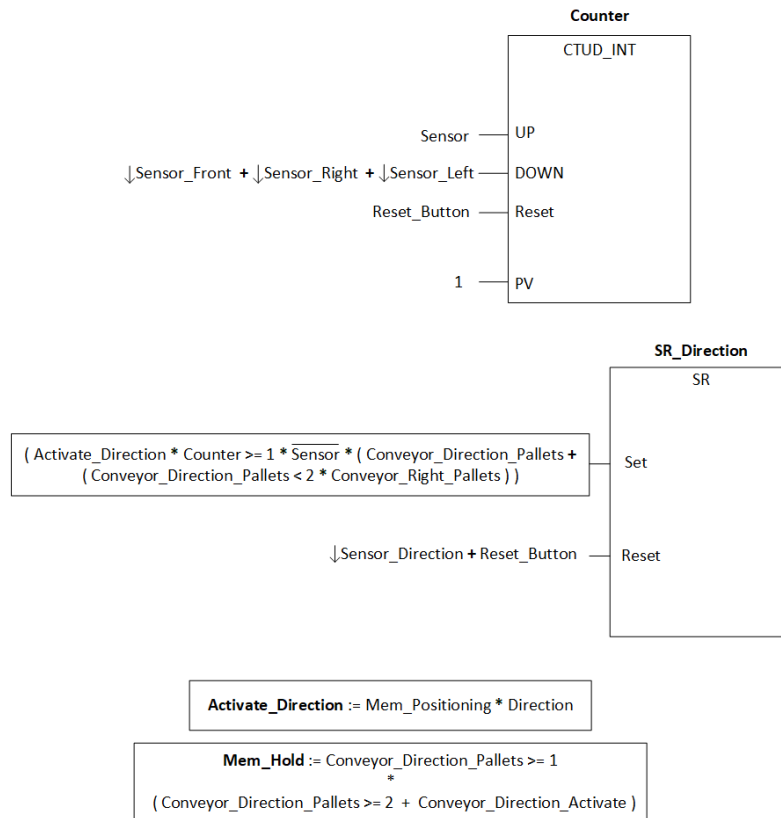


Figura 4.64: Estruturas auxiliares do bloco FB_TT

FB_RT

Function block criado para controlo das mesas rotativas. Tal como os blocos anteriores, possui um contador para controlar a quantidade de paletes existentes na mesa. Este contador é incrementado com a deteção de uma transição negativa do sensor final do transportador de rolos que alimenta a mesa rotativa (*Sensor_Previous*) enquanto a mesa se encontra na posição de ângulo 0 (posição de receção de paletes). O contador é reduzido com a deteção de transições negativas do sensor na zona inicial ou final da mesa (*Sensor_Back* e *Sensor_Front* respetivamente) enquanto esta se encontra na posição de ângulo 90.

O funcionamento deste equipamento começa quando uma paleta ainda está no transportador de rolos anterior, prestes a entrar na mesa rotativa. Os rolos da mesa são ativados para movimentar a paleta para a frente, até esta estar corretamente posicionada. A paleta encontra-se corretamente posicionada quando é detetada uma transição negativa do *Sensor_Back*. Posteriormente, se a paleta for em frente, os rolos continuam ativados. Se a paleta mudar de direção, é ativada a rotação da mesa. Importante referir que a mesa rotativa existente no Factory IO apenas permite a rotação no sentido contrário aos ponteiros do relógio. Depois da mesa se encontrar na posição de ângulo 90, se a paleta for para a esquerda os rolos movimentam-a para a frente, se for para a direita os rolos movimentam-a para trás. A mesa retorna à posição de ângulo 0 passados 0.7 segundos da paleta ter saído.

A decisão da direção que a paleta deve seguir é tomada de forma idêntica às mesas de transferência, com o código PLC representado nas tranches de código 4.4 e 4.5.

Sempre que um determinado FB deste tipo de equipamento é instanciado a aplicação C# preenche os *inputs* e *outputs* com as variáveis globais correspondentes (figura 4.65).

No *grafcet* existente na figura 4.66 encontra-se representada a lógica sequencial de operações seguida para o correto funcionamento deste tipo de equipamentos.

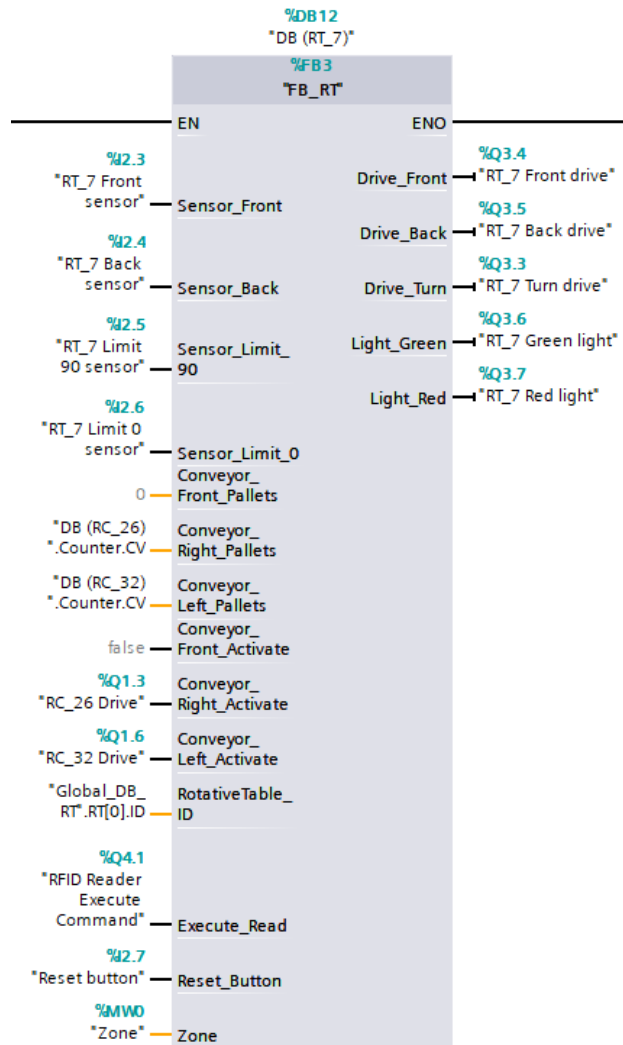


Figura 4.65: Exemplo de um FB_RT instanciado, vista no TIA Portal

O significado das variáveis locais usadas são:

- **Sensor_Front:** sensor na zona inicial da mesa rotativa;
- **Sensor_Back:** sensor na zona final da mesa rotativa;
- **Sensor_Previous:** sensor final do transportador de rolos que "alimenta" a mesa rotativa;
- **Sensor_Limit_90:** sensor que deteta a posição da mesa no ângulo de 90 graus;
- **Sensor_Limit_0:** sensor que deteta a posição da mesa no ângulo de 0 graus;
- **Conveyor_Direction_Pallets:** quantidade de paletes existentes no transportador na direção referida;
- **Conveyor_Direction_Activate:** indica se o transportador na direção referida se encontra em funcionamento;

- **RotativeTable_ID**: ID da mesa rotativa em questão;
- **Execute_Read**: execução de uma leitura por parte do leitor RFID;
- **Reset_Button**: botão que reinicia o contador e os vários *set resets* existentes neste bloco;
- **Zone**: zona de destino da encomenda em questão;
- **Drive_Front**: ativação dos rolos da mesa para movimentar a paleta em frente;
- **Drive_Back**: ativação dos rolos da mesa para movimentar a paleta para trás;
- **Drive_Turn**: ativação da rotação da mesa;
- **Light_Green**: sinalizador verde da mesa rotativa;
- **Light_Red**: sinalizador vermelho da mesa rotativa;

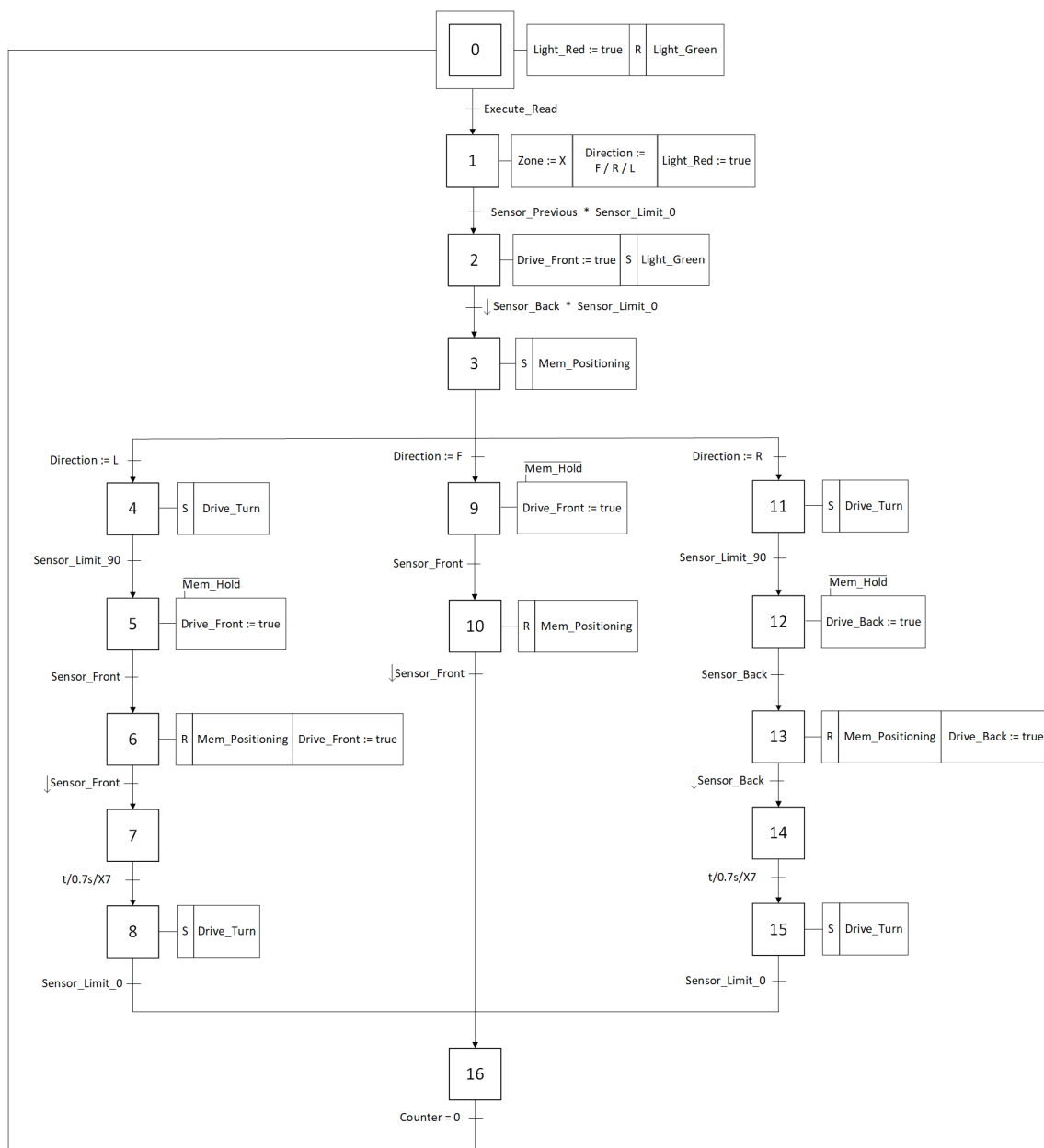


Figura 4.66: Grafcet funcional do FB_RT

Existem ainda várias variáveis auxiliares importantes, cuja definições se encontram na figura 4.67, na programação deste bloco:

- **Mem_Hold**: tal como no FB_RC, esta variável serve para reter as paletes na mesa de transferência enquanto o equipamento de destino não se encontra preparado para as receber;
- **SR_Front**: *set reset* que aciona os rolos da mesa para movimentar a paleta em frente;
- **SR_Back**: *set reset* que aciona os rolos da mesa para movimentar a paleta para trás;
- **SR_Turn**: *set reset* que aciona a rotação da mesa;
- **Timer**: temporizador, acionado pelo SR_Turn, que dá tempo à paleta para sair totalmente da mesa rotativa antes desta volta à posição inicial;
- **Activate_Direction**: variável que indica quando uma paleta está pronta para tomar uma determinada direção.

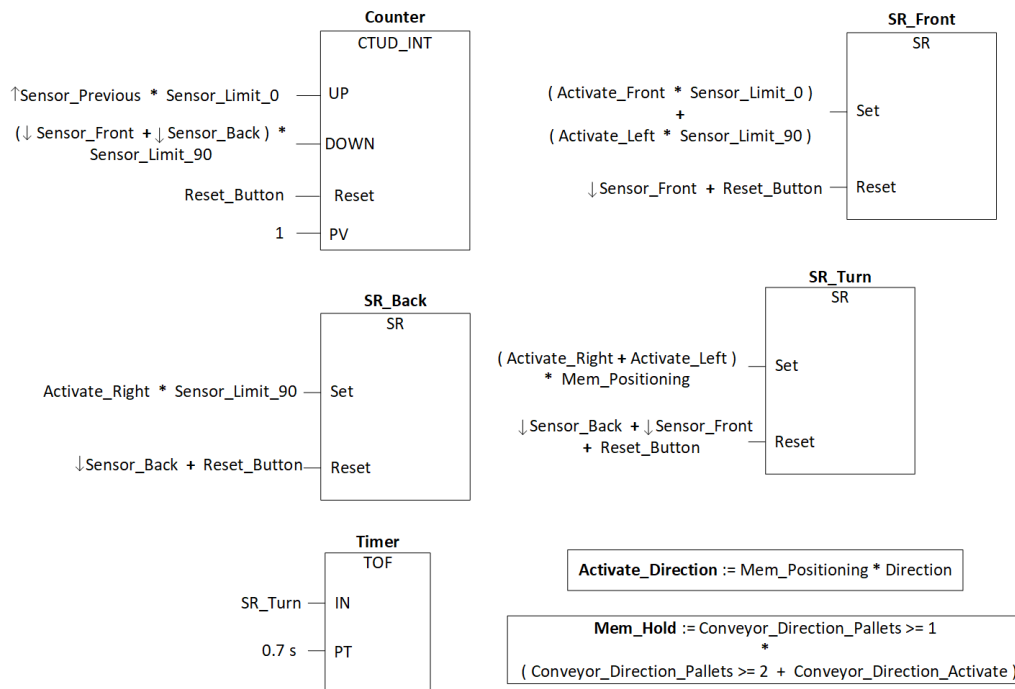


Figura 4.67: Estruturas auxiliares do bloco FB_RT

Abordagem prática - Geração do ambiente de simulação

Neste capítulo é realizada uma descrição detalhada da geração automática do ambiente virtual de simulação pela aplicação desenvolvido. Este ambiente de simulação contém a recriação fiel do sistema periférico construído na interface gráfica bem como a associação de todos os nomes de transportadores e outros aparelhos com as respectivas variáveis no programa PLC.

São abordadas as ferramentas e metodologias consideradas para a realização deste processo. Nesta abordagem foi também estudada a possibilidade de interação com o TIA Portal, uma vez que é neste software que se encontra a programação PLC para o controlo do sistema periférico.

5.1 Ferramentas e metodologias utilizadas

Nesta secção são abordadas as ferramentas e metodologias utilizadas no desenvolvimento da parte da aplicação que gera automaticamente o cenário virtual de simulação 3D. Foi explorado também o software de simulação Factory IO e como os ambientes de simulação se encontram codificados. Tal como na geração do código PLC, foram utilizados documentos XML para criar o cenário virtual correspondente ao layout construído na interface gráfica.

5.1.1 Ficheiros XML

Os ficheiros XML são utilizados, atualmente, em vários tipos de software e programas. Para a correta geração do ambiente virtual de simulação, tal como na geração do código PLC, recorreu-se a documentos XML, mais concretamente à manipulação deste tipo de ficheiros para a correta criação do cenário.

Os princípios e regras referidos em 4.1.3 aplicam-se neste caso, nomeadamente, a correta estruturação do ficheiro XML, de modo é que este seja compatível com o respetivo *schema* (neste caso trata-se do *schema* correspondente à codificação de um cenário no Factory IO) e as regras de sintaxe.

5.1.2 Factory IO

O Factory IO é um software de simulação 3D usado para aprender tecnologias de automação. Permite a criação de um ambiente virtual de uma maneira fácil e rápida através da seleção de objetos e equipamentos comuns numa instalação autónoma. O cenário de controlo mais habitual no Factory IO é o uso de PLCs para controlo de todo o ambiente, uma vez que são os aparelhos mais comuns usados para aplicações industriais. É também possível usar micro-controladores para realizar esse controlo [31].

Foi o software escolhido para executar simulações e testar código PLC gerado devido à familiaridade ganha durante o mestrado de Automação e por possibilitar o controlo de um ambiente virtual a partir do simulador PLC da Siemens.

5.2 Geração automática do cenário de simulação no Factory IO

Tal como o código PLC, também o ambiente virtual de simulação é gerado automaticamente a partir do desenho em Visio. Este processo é conseguido através da edição de um documento XML proveniente de outro ambiente virtual em Factory IO, o cenário modelo.

O uso de um ambiente de simulação é uma fase importante na programação de algumas partes de armazéns automáticos uma vez que permite ao engenheiro testar o seu código antes deste ser aplicado no terreno. O facto deste ambiente ser gerado automaticamente permite ao programador poupar uma grande quantidade de tempo que seria gasta na preparação manual de um armazém automático no simulador. Tarefa esta que seria muito extensa devido ao elevado número de equipamentos constituintes deste tipo de armazéns.

Na figura 5.1 temos um exemplo de um sistema periférico criado no Visio e o respetivo ambiente de simulação virtual gerado.

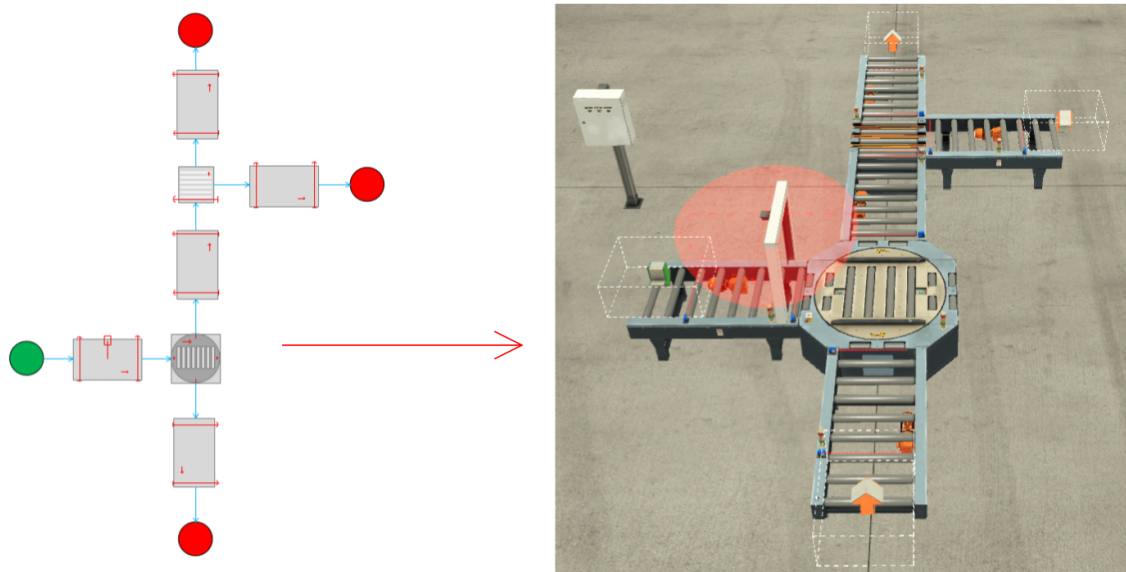


Figura 5.1: Sistema periférico criado no Visio e respetivo cenário virtual

5.2.1 Manipulação dos ficheiros XML

A primeira fase na geração automática do ambiente virtual foi descobrir como os cenários se encontram codificados. Uma vez que a codificação de ficheiros em estrutura XML é muito comum hoje em dia, foi esse o primeiro teste. Foi possível verificar que a codificação de um cenário de Factory IO era de facto em estrutura XML (figura 5.2).



Figura 5.2: Estrutura XML de um cenário de Factory IO vista no *Bloco de notas*

Uma vez que não existe documentação por parte da Real Games (empresa detentora pelo software Factory IO) referente à estrutura deste tipo de ficheiros, a sua análise começou "do zero" ao tentar perceber a função de cada elemento e atributo existente no ficheiro. Na figura 5.3 temos o significado e posição dos elementos mais superficiais do ficheiro XML.

Para verificar se era possível alterar um cenário virtual através da modificação de informação do ficheiro XML, uma vez que não existe documentação nem provas anteriores da eficácia deste processo aplicado ao software em questão, procedeu-se a um simples teste. No ficheiro XML foram alteradas as coordenadas correspondentes a um objeto de um cenário no Factory IO. Posteriormente esse ficheiro foi guardado com a extensão *.factoryio*. Quando o cenário foi aberto no simulador verificou-se que o objeto tinha mudado de posição, para as coordenadas alteradas. Ficou então provada a viabilidade deste procedimento.

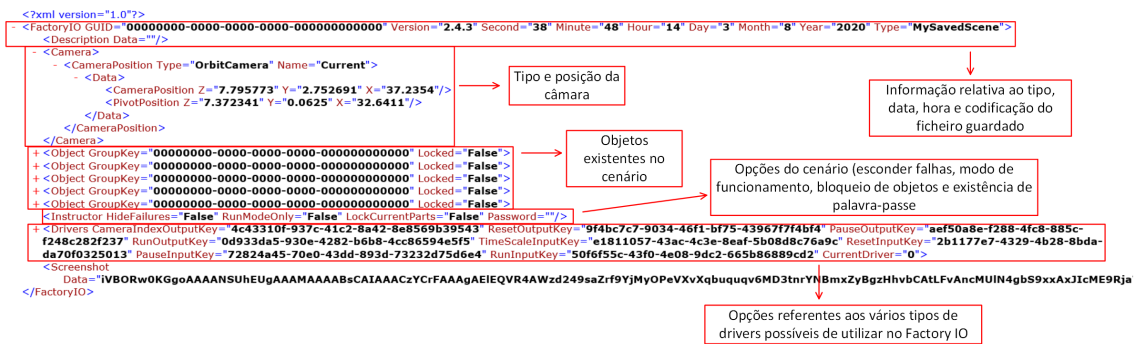


Figura 5.3: Significado de alguns elementos do ficheiro XML

Sabendo isto, a geração automática do cenário virtual de simulação segue um princípio semelhante à geração automática de código PLC, com a edição de documentos XML a ser o caminho a seguir para realizar esta tarefa. A manipulação deste tipo de ficheiros foi feita utilizando *XmlDocument*, tal como nos documentos XML a serem inseridos no TIA Portal.

5.2.2 Ficheiro XML original

O cenário virtual inicial, bem como o respetivo ficheiro XML, já possui um quadro elétrico com três botões (*Start button*, *Stop Button* e *Reset button*) uma vez que todos os cenários gerados contemplam estes objetos. Os cinco elementos com o nome *Object* que se encontram no ficheiro da figura 5.3 correspondem à coluna de suporte, quadro elétrico e os referidos três botões.

Este cenário inicial foi construído no Factory IO manualmente e posteriormente foi extraído o respetivo ficheiro XML. Como foi dito em 4.3, cada ambiente virtual é gerado ao adicionar informação a este ficheiro XML, o documento XML modelo. Logo, todos os cenários criados em Factory IO vão ter os objetos codificados no ficheiro. De cada vez que é necessário adicionar um novo objeto, é criado um novo elemento *Object*, que é colocado depois do último *Object* existente.

5.2.3 Colocação dos equipamentos no ambiente virtual

Na aplicação desenvolvida foram contemplados três tipos de equipamentos, como já foi referido: transportadores de rolos, mesas de transferência e mesas rotativas. No cenário virtual, para além destes equipamentos em si, existem outros objetos que se encontram associados para o correto funcionamento do sistema. Portanto os objetos que são ou podem ser colocados, além dos pré-existentes, nos cenários virtuais encontram-se ilustrados na figura 5.4.

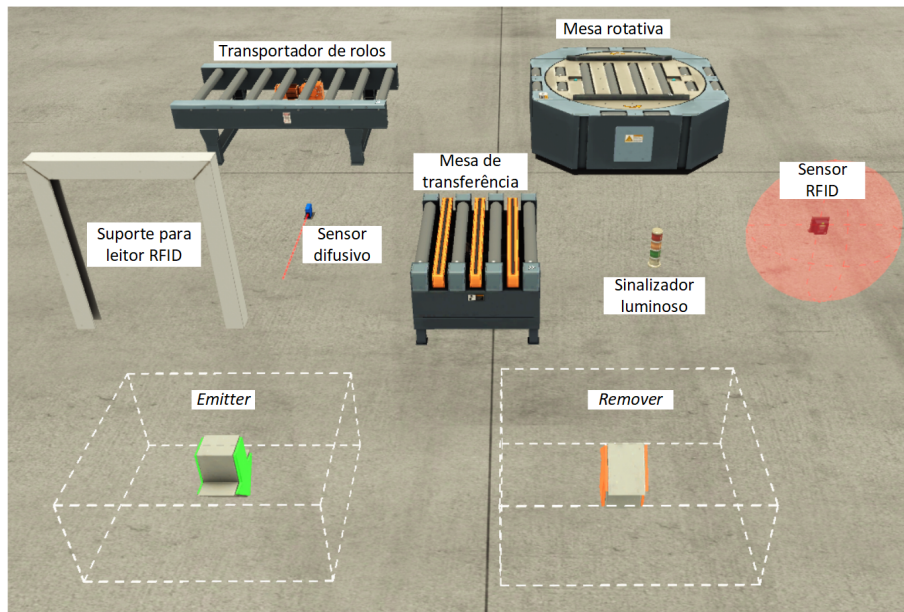


Figura 5.4: Equipamentos existentes num cenário virtual

Quando é colocado um novo objeto no documento XML, existem parâmetros que têm que ser preenchidos. Todos os objetos possuem o elemento *Proxy*, onde se encontra a definição da sua orientação e posição. A seguir a este existem outros elementos que dizem respeito aos *inputs* e *outputs* do objeto em questão, bem como os endereços e outras configurações. Na figura 5.5 encontra-se o elemento referente à colocação de um transportador de rolos.

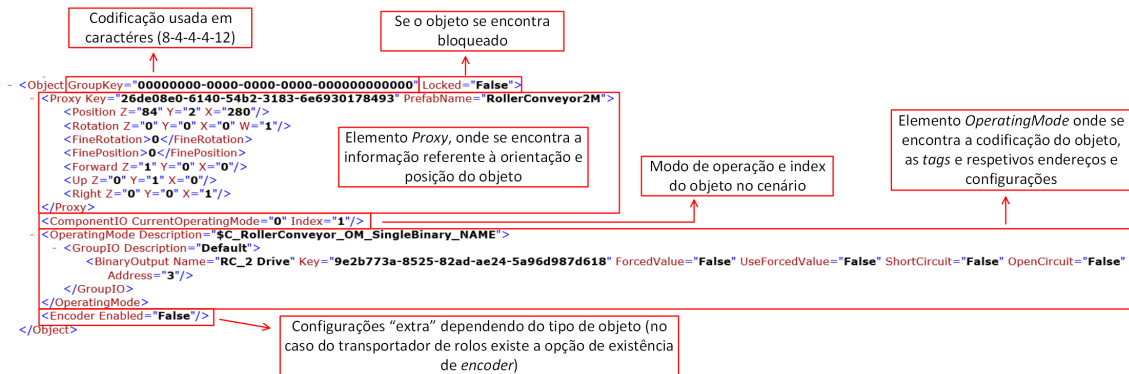


Figura 5.5: Exemplo do elemento correspondente à colocação de um transportador de rolos

5.2.3.1 Orientação e posição

Como foi dito anteriormente, dentro do elemento *Object* existe o elemento *Proxy* com a informação acerca da orientação e posição do objeto, representado na figura 5.6. Para além disso, ainda possui dois atributos:

- **Key:** código com 36 caracteres (na sequência 8-4-4-4-12) único para cada objeto;
- **PrefabName:** nome do tipo de objeto no Factory IO.

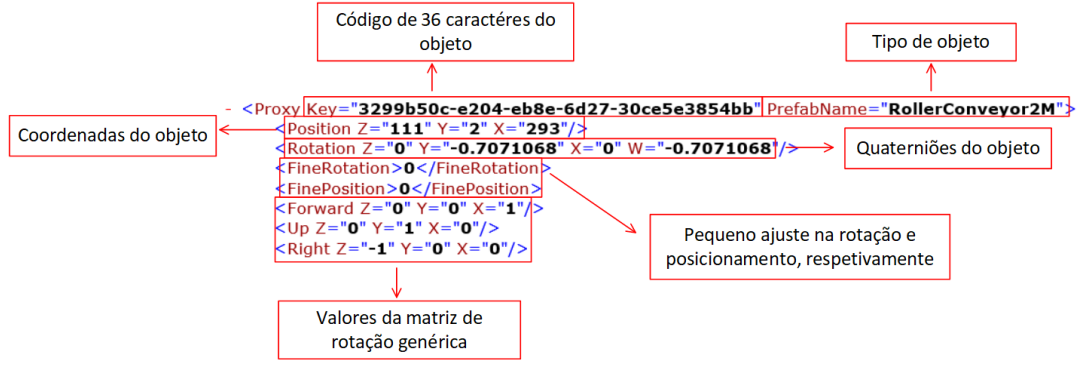


Figura 5.6: Significado dos elementos filho do elemento *Proxy*

No que diz respeito à posição do objeto a sua atribuição é bastante simples. Dado que o cenário de simulação possui 3 eixos, no elemento *Position* encontram-se as coordenadas do objeto em questão. O atributo Y é a cota, e os atributos Z e X são o comprimento e largura, respetivamente.

No que toca à orientação, esta é representada por quaterniões. Os quaterniões são uma extensão dos números complexos e representam um modo mais fácil e compacto da matriz de rotação. A matriz de rotação genérica, representada pelos atributos dos elementos *Forward*, *Up* e *Right* tem a seguinte estrutura.

$$\begin{pmatrix} \mathbf{Z}_{Forward} & \mathbf{Y}_{Forward} & \mathbf{X}_{Forward} \\ \mathbf{Z}_{Up} & \mathbf{Y}_{Up} & \mathbf{X}_{Up} \\ \mathbf{Z}_{Right} & \mathbf{Y}_{Right} & \mathbf{X}_{Right} \end{pmatrix}$$

O quaterniões calculados a partir da matriz de rotação genérica encontram-se como atributos do elemento *Rotation*, e são calculados através das seguintes expressões:

$$\begin{aligned} \mathbf{W} &= \frac{\sqrt{\mathbf{Z}_{Forward} + \mathbf{Y}_{Up} + \mathbf{X}_{Right} + 1}}{2} & \mathbf{Z} &= \frac{\sqrt{\mathbf{Z}_{Forward} - \mathbf{Y}_{Up} - \mathbf{X}_{Right} + 1}}{2} \\ \mathbf{Y} &= \frac{\sqrt{\mathbf{Y}_{Up} - \mathbf{Z}_{Forward} - \mathbf{X}_{Right} + 1}}{2} & \mathbf{X} &= \frac{\sqrt{\mathbf{X}_{Right} - \mathbf{Z}_{Forward} - \mathbf{Y}_{Up} + 1}}{2} \end{aligned}$$

E os respetivos sinais são:

$$\begin{aligned} \text{sin}(\mathbf{Z}) &= \text{sin}(\mathbf{Y}_{Right} - \mathbf{X}_{Up}) \\ \text{sin}(\mathbf{Y}) &= \text{sin}(\mathbf{X}_{Forward} - \mathbf{Z}_{Right}) \\ \text{sin}(\mathbf{X}) &= \text{sin}(\mathbf{Z}_{Up} - \mathbf{Y}_{Forward}) \end{aligned}$$

O quaternião W pode ser representado de maneira normalizada ou não normalizada, não permitindo ou permitindo rotações negativas, respetivamente. Os sinais dos restantes quaterniões são definidos consoante a representação do quaternião W. As expressões anteriores correspondem à representação normalizada, onde o quaternião W é sempre positivo.

Os elementos *FineRotation* e *FinePosition* apenas tomam valores diferentes de zero quando existem mudanças de inclinações ou posições muito pequenas, como por exemplo, inclinar um sensor por 1°.

A posição do transportador de rolos que recebe as paletes, ou seja, do primeiro transportador que existe no circuito, é definida "manualmente". É sempre a mesma posição para qualquer circuito criado, pois é uma posição central no ambiente virtual, para que todo o circuito se enquadre corretamente no cenário. Todos os equipamentos têm cota de 2 ($Y = 2$). Quanto às coordenadas X e Z, as do primeiro transportador são 280 e 84, respectivamente. Todos os restantes equipamentos são colocados tendo em conta o primeiro transportador inserido.

A lógica da definição da posição (coordenadas) de cada equipamento também se encontra definida na aplicação C#. Sempre que é colocado qualquer equipamento, o programa verifica a orientação do mesmo (frente, trás, baixo ou cima), e os valores dos quaterniões são inseridos com base nisso.



Figura 5.7: Lógica da colocação de equipamentos no ambiente virtual

Resumindo a lógica do código representada na figura 5.7, o programa percorre cada equipamento existente no desenho em Visio, e caso o equipamento já tenha sido colocado no cenário, o programa vai buscar o ID do equipamento(s) seguinte(s), dependendo se o equipamento só tenha um destino (caso dos transportadores de rolos) ou exista a possibilidade de ter mais que um (caso das mesas de transferência e mesas rotativas). Posteriormente, caso o equipamento seguinte ainda não esteja colocado no cenário, o programa procede à sua colocação, e adiciona o seu ID na lista de equipamentos que já se encontram no cenário virtual. Esta lógica funciona até todos os equipamentos se encontrarem no cenário.

5.2.3.2 *Tags*

Estando a posição e orientação de um objeto definida, existem outros elementos que precisam de ser especificados. Depois do elemento *Proxy* temos o elemento *ComponentIO* com dois atributos:

- **CurrentOperatingMode:** *index* da configuração do objeto no Factory IO;
- **Index:** *index* do objeto em questão, em relação a todos os objetos existentes do mesmo tipo, no cenário de Factory IO;

A seguir temos o elemento *OperatingMode*. Neste elemento temos o modo de operação que se aplica ao objeto em questão (atributo *Description*) e os *inputs* e *outputs* associados. A maioria dos objetos têm diferentes tipos e quantidades de *tags*, porém a definição de cada uma encontra-se sempre dentro do elemento destinado a armazenar esta informação, de nome *GroupIO*. Já no elemento correspondente a uma determinada *tag*, existem vários parâmetros a definir:

- Tipo de *tag* (*Input* ou *Output*);
- Tipo de variável associada (*Binary*, *Int* ou *Real*);
- Nome atribuído;
- Código de 36 caracteres, único para cada *tag*;
- Endereço (número inteiro entre 0 e o número total de *inputs* ou *outputs*);
- Parâmetros relacionados com a ativação ou desativação durante a simulação.

Se existirem outras funcionalidades adicionais que só o tipo de objeto em questão contemplem, como a existência de *encoder* num transportador ou o *range* de um sensor, estas aparecem depois do elemento *OperatingMode*. Caso contrário, a definição do objeto está concluída.

Toda a informação necessária no documento XML referente aos elementos *ComponentIO*, *OperatingMode* e funcionalidades adicionais encontra-se representada na figura 5.8.

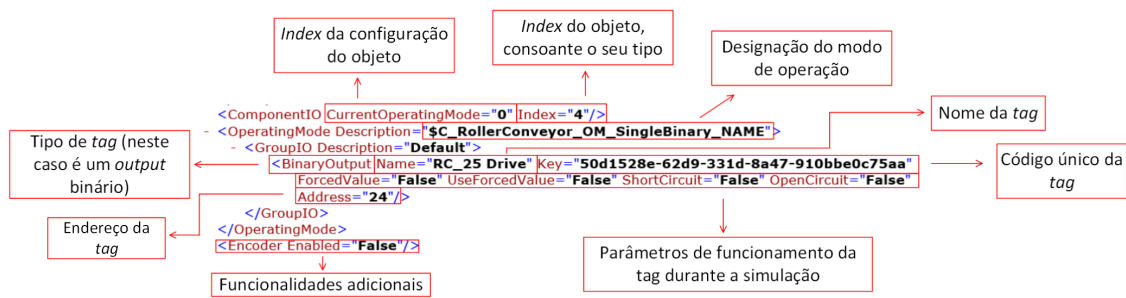


Figura 5.8: Significado dos restantes elementos de um objeto

5.2.4 Associação de *tags* e variáveis

Depois do cenário virtual estar totalmente gerado, com todos os equipamentos corretamente posicionados e os seus nomes atribuídos, procede-se à associação das *tags* do Factory IO com as respetivas variáveis no TIA Portal. Esta operação é executada pela aplicação C#, ainda na edição do documento XML. No documento existe o elemento *Drivers* com elementos filhos correspondentes aos diferentes PLCs ou simuladores com que o Factory IO pode interagir (figura 5.9).

```
- <Drivers CameraIndexOutputKey="4c43310f-937c-41c2-8a42-8e8569b39543"
ResetOutputKey="9f4bc7c7-9034-46f1-bf75-43967f7f4bf4" PauseOutputKey="aef50a8e-f288-4fc8-
885c-f248c282f237" RunOutputKey="0d933da5-930e-4282-b6b8-4cc86594e5f5"
TimeScaleInputKey="e1811057-43ac-4c3e-8eaf-5b08d8c76a9c" ResetInputKey="2b1177e7-4329-
4b28-8bda-da70f0325013" PauseInputKey="72824a45-70e0-43dd-893d-73232d75d6e4"
RunInputKey="50f6f55c-43f0-4e08-9dc2-665b86889cd2" CurrentDriver="0">
+ <Advantech4750>
+ <Advantech4704>
+ <AllenBradleyLogix5000>
+ <AllenBradleyMicro800>
+ <AllenBradleyMicroLogix>
+ <AllenBradleySLC5>
+ <Automgen>
+ <ControlIO>
+ <MHJ>
+ <ModbusTCPClient>
+ <ModbusTCPServer>
+ <OPCCientDA>
+ <SiemensLOGOTCP>
+ <SiemensS7300S7400TCP>
+ <SiemensS71200S71500TCP>
+ <SiemensS7PLCSIM>
</Drivers>
```

Figura 5.9: Elemento *Drivers* e respetivos elementos filhos

Dentro de cada elemento correspondente a um dos PLCs ou simuladores encontram-se todos os dados referentes às ligações entre o Factory IO e o respetivo PLC ou simulador. Na aplicação em questão é usado o simulador PLC da Siemens de nome *SiemensS7PLCSIM*. Na página do Factory IO correspondente a este simulador (figura 5.10) é possível associar a cada endereço das variáveis do TIA Portal uma *tag* do cenário de simulação. A mesma operação pode ser feita diretamente no ficheiro XML do cenário, ao colocar no atributo *PointIOKey* de cada variável o código de 36 caracteres da *tag* correspondente, como se encontra representado na figura 5.11.

Significado de cada atributo:

- Elemento *PointIOCount*:
 - **NumericOutputCount**: número de *outputs* do tipo DWORD (número inteiro entre 0 e 4294967295);
 - **NumericInputCount**: número de *inputs* do tipo DWORD;
 - **IntOutputCount**: número de *outputs* do tipo Int (número inteiro entre -2147483648 e 2147483648);
 - **IntInputCount**: número de *inputs* do tipo Int;
 - **FloatOutputCount**: número de *outputs* do tipo Float (número decimal);
 - **FloatInputCount**: número de *outputs* do tipo Float;
 - **BitOutputCount**: número de *outputs* do tipo Bool (variável binária);
 - **BitInputCount**: número de *inputs* do tipo Bool;
- Elemento *PointIOOffset*:
 - **NumericOutputOffset**: *offset* dos *outputs* do tipo DWORD;
 - **NumericInputOffset**: *offset* dos *inputs* do tipo DWORD;
 - **IntOutputOffset**: *offset* dos *outputs* do tipo Int;
 - **IntInputOffset**: *offset* dos *inputs* do tipo Int;
 - **FloatOutputOffset**: *offset* dos *outputs* do tipo Float;
 - **FloatInputOffset**: *offset* dos *inputs* do tipo Float;
 - **BitOutputOffset**: *offset* dos *outputs* do tipo Bool;
 - **BitInputOffset**: *offset* dos *inputs* do tipo Bool;

As funcionalidades da aplicação desenvolvida reforçam as vantagens da flexibilização da programação de PLCs para situações complexas e padronizadas e provam que é possível gerar automaticamente um ambiente de simulação virtual através da sua representação gráfica. Estes dois componentes, postos em prática numa empresa como a Consoveyo S.A., poderiam aumentar a produtividade de uma empresa pois diminuem o tempo de programação e erros adjacentes à programação manual, enquanto facilitam o comissionamento e posterior manutenção de um armazém automático, devido à utilização de blocos de programação previamente verificados e à existência de um standard bem definido.

A geração automática do ambiente de simulação justifica a utilização de emuladores para o teste da solução desenvolvida, uma vez que reduz significativamente o tempo necessário para a recriação de um sistema periférico. Para além disso, a verificação do código PLC numa fase anterior à implementação da solução permite a deteção de erros e à correção dos mesmos ainda na fase de desenvolvimento.

6

Conclusões e perspetivas de trabalhos futuros

6.1 Conclusões

O tema da presente dissertação apelava à implementação da estruturação automática de programas PLC de controlo de transportadores seguindo a metodologia do grupo Körber na Consoveyo S.A.. O objetivo passou pelo auxílio ao departamento de Automação da Consoveyo S.A. a transitar dos procedimentos levados a cabo atualmente para os do grupo Körber, ao documentar os processos estudados e ajudar na consolidação deste novo standard na empresa.

Segundo a pesquisa bibliográfica geral, a geração automática de código é uma temática com várias abordagens, porém nenhuma se destaca por se encontrar num patamar mais avançado ou uso em ambiente industrial ou empresarial. Na bibliografia industrial temos a metodologia do grupo Körber que usou a geração de código para melhorar a eficácia da empresa.

A abordagem prática desta dissertação passou pelo desenvolvimento de uma aplicação que fosse de encontro com a metodologia estudada, para um possível uso a nível empresarial. Foram definidos como objetivos a geração automática de código e do ambiente virtual de simulação.

A presente dissertação reforça as vantagens da utilização da geração de código para a programação de sistemas complexos e padronizados.

É demonstrado o potencial de um software como o Visio, para as gerações de código e ambiente de simulação, devido à informação existente neste tipo de ficheiros e à estrutura formada para a extrair. Permite ainda facilmente alterar essa mesma estrutura para gerar outros documentos com o objetivo de alterar a metodologia, caso seja necessário.

A utilização de ficheiros XML para a geração de ambos os componentes demonstra como a automação beneficia do esforço de standardização do campo da informática, permitindo assim várias maneiras de atingir um determinado objetivo.

Por último, conclui-se que a geração e reutilização de código devidamente testado em processos de elevada complexidade e repetitivos, como é o caso dos armazéns automáticos

desenvolvidos na Consoveyo S.A., representa uma enorme vantagem na forma de trabalhar de uma empresa, diminuindo significativamente o tempo de programação dos equipamentos e de recriação do ambiente de simulação.

6.2 Trabalhos Futuros

Na presente dissertação é feita uma abordagem à geração automática de código PLC e de um ambiente virtual de simulação, inspirada na metodologia estudada. Existem várias possibilidades de continuar o trabalho desenvolvido ou usá-lo para introduzir abordagens semelhantes, que se encontram sugeridos de seguida.

Aumentar o número de equipamentos e situações abrangidos pela aplicação, de modo a maximizar o uso da mesma e assim tornar mais intuitiva e fácil a programação deste tipo de sistemas e posterior manutenção.

Na aplicação desenvolvida foi escolhido o software Visio como interface com o utilizador. Seria interessante explorar as possibilidades de usar outros softwares que permitam uma perspetiva ilustrativa da solução a implementar, como por exemplo o AutoCAD, como interface. Deste modo, uma empresa que queira implementar esta metodologia não tem que mudar totalmente as ferramentas que usa, facilitando assim a transição.

Melhorar a geração do ambiente virtual de simulação para que este seja o mais idêntico possível ao sistema real.

Aumentar a quantidade de equipamentos, aparelhos e situações abrangidas pela interface gráfica, e melhorar a geração do ambiente de simulação virtual para que tanto o sistema periférico construído como a respetiva simulação sejam o mais idênticos possíveis ao sistema real.

Permitir uma maior customização do primeiro desenho elaborado no Visio dando mais opções de disposição dos equipamentos e adição de outros aparelhos que possam interagir com os mesmos.

Aumento das funcionalidades da aplicação, como a adição de hardware no TIA Portal.

Bibliografia

- [1] A. P. F. Industry, “*The 21st Century Warehouse.*” Disponível em: <https://apfoodonline.com/industry/the-21st-century-warehouse/>. Último acesso em 07/09/2020.
- [2] A. P. Magalhães, *Documentação de Computação Industrial, FEUP*.
- [3] D. H. Hanssen, “Programmable Logic Controllers: A practical approach to IEC 61131-3 using CODESYS,” 2015.
- [4] E. Estevez, M. Marcos, and D. Orive, “Automatic generation of PLC automation projects from component-based models,” *International Journal of Advanced Manufacturing Technology*, vol. 35, no. 5-6, pp. 527 – 40, 2007/12.
- [5] K. Sacha, “Automatic code generation for plc controllers,” *Computer Safety, Reliability, and Security. 24th International Conference, SAFECOMP 2005. Proceedings (Lecture Notes in Computer Science Vol. 3688)*, 2005.
- [6] O. Duran and A. Batocchio, “Automatic PLC software generator with a natural interface,” *International Journal of Production Research*, vol. 37, no. 4, pp. 805 – 19, 1999/03/10.
- [7] A. Pereira, C. Lima, and J. F. Martins, “The use of IEC 61131-3 to enhance PLC control and Matlab/Simulink process simulations,” pp. 1243–1247, 2011.
- [8] D. Thapa, C. M. Park, S. Park, and G.-N. Wang, “Auto-Generation of IEC Standard PLC Code Using t-MPSG,” *International Journal of Control, Automation, and Systems*, vol. 7, no. 2, pp. 165 – 74, 2009/04.
- [9] M. Schaumlfer, P. Moll, L. Brocke, S. Coutandin, and J. Fleischer, “Model for Web-Application based Configuration of Modular Production Plants with automated PLC Line Control Code Generation,” *Procedia CIRP*, vol. 83, pp. 292 – 7, 2019//.
- [10] M. R. Barbosa, *Documentação de Automação Flexível, FEUP*.
- [11] M. Zhou, H. Wan, R. Wang, X. Song, C. Su, M. Gu, and J. Sun, “Formal component-based modeling and synthesis for PLC systems,” *Computers in Industry*, vol. 64, no. 8, pp. 1022 – 1034, 2013.
- [12] G. Frey and L. Litz, “Formal methods in PLC programming,” vol. 4, pp. 2431–2436 vol.4, Oct 2000.
- [13] J. Farines, M. H. de Queiroz, V. G. da Rocha, A. M. M. Carpes, F. Vernadat, and X. Crégut, “A model-driven engineering approach to formal verification of PLC programs,” pp. 1–8, Sep. 2011.

- [14] M. Jbair, B. Ahmad, M. Ahmad, D. Vera, R. Harrison, and T. Ridler, “Automatic plc code generation based on virtual engineering model,” *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS). Proceedings*, pp. 675 – 80, 2019.
- [15] B. Lydon, “Automatic PLC Code Generation & Design Interchange Standards.” Disponível em <https://www.automation.com/en-us/articles/2003-1/automatic-plc-code-generation-design-interchange-s>. Último acesso em 06/05/2020.
- [16] RIVOPS, “AutoGen, an automatic code generator for the process industry.” Disponível em: <http://www.rivops.com/autogen>. Último acesso em 17/02/2020.
- [17] MathWorks, “Simulink PLC Coder.” Disponível em: <https://www.mathworks.com/products/simulink-plc-coder.html>. Último acesso em 18/02/2020.
- [18] Siemens, “TIA Portal Openness: Introduction and Demo Application.” Disponível em: <https://support.industry.siemens.com/cs/document/108716692/tia-portal-openness%3A-introduction-and-demo-application?dti=0&lc=en-WW>. Último acesso em 18/02/2020.
- [19] Siemens, *Automation projects with scripts*. 2017.
- [20] H. M. A. Domingos, *Desenvolvimento de uma aplicação para geração automática de código para o PLC de controlo de um shuttle car*. 2018.
- [21] Microsoft, “Visio.” Disponível em: <https://www.microsoft.com/pt-pt/microsoft-365/visio/flowchart-software>. Último acesso em 04/03/2020.
- [22] E. S. . Service, “EPLAN Electric P8: Potência para o planeamento e engenharia elétrica.” Disponível em: <https://www.eplan.pt/pt/solucoes/engenharia-electrica/eplan-electric-p8/>. Último acesso em 09/09/2020.
- [23] Autodesk, “O que é o AutoCAD?” Disponível em: <https://www.autodesk.pt/products/autocad/overview>. Último acesso em 25/03/2020.
- [24] Microsoft, “Microsoft Excel.” Disponível em: <https://www.microsoft.com/pt-pt/microsoft-365/excel>. Último acesso em 16/03/2020.
- [25] P. Center, “Overview of STEP 7.” Disponível em: <http://www.plccenter.cn/Siemens-Step7/bas00014.htm>. Último acesso em 18/05/2020.
- [26] Siemens, “Overview TIA Portal.” Disponível em: https://cache.industry.siemens.com/dl/dl-media/707/106656707/att_961750/v9/106656707_TIA_Portal_-_Tutorial_Center_web/start.htm#!/en/4473. Último acesso em 18/02/2020.
- [27] Prologistik, “Material Flow Control PL-MAT.” Disponível em: <https://www.prologistik.com/en/warehouse-management-software/material-flow-control/>. Último acesso em 16/02/2020.
- [28] T. Target, “Warehouse Management System.” Disponível em: <https://searcherp.techtarget.com/definition/warehouse-management-system-WMS>. Último acesso em 17/03/2020.
- [29] w3schools.com, “XML Tree.” Disponível em: <https://www.w3schools.com/xml/xml-tree.asp>. Último acesso em 04/05/2020.

- [30] Siemens, “*How is the XML file structured for blocks?*” Disponível em: [https://support.industry.siemens.com/cs/document/109480446/how-is-the-xml-file-structured-for-blocks-\(export-import-in-tia-portal-openness-v13-sp1\)-?dti=0&lc=en-CH](https://support.industry.siemens.com/cs/document/109480446/how-is-the-xml-file-structured-for-blocks-(export-import-in-tia-portal-openness-v13-sp1)-?dti=0&lc=en-CH). Último acesso em 21/07/2020.
- [31] R. Games, “*About Factory IO.*” Disponível em: <https://docs.factoryio.com/>. Último acesso em 20/07/2020.