Desenvolvimento de uma aplicação para geração automática de código para o PLC de controlo de elevadores

Cristina Lírio Ferreira

Dissertação de Mestrado

Orientador na FEUP: Professor Doutor Germano Correia dos Santos Veiga

Orientador na Consoveyo, S.A.: Engenheiro Vitor Vaz

Mestrado Integrado em Engenharia Mecânica

Setembro 2020

"All our dreams can come true, if we have the courage to pursue them."

Walt Disney

Resumo

A complexidade das instalações de produção e dos sistemas automáticos modernos tem criado novos e diferentes desafios às indústrias. Além disso, a crescente pressão sobre os custos e tempos de execução leva os fabricantes a procurarem novas formas de responder a esses desafios. Para encurtar a fase de programação dos referidos sistemas propõe-se a geração automatizada de código para Controladores Lógicos Programáveis (*Programmable Logic Controller* - PLCs).

A geração de código para PLC é uma técnica que introduz na indústria novos métodos e tecnologias que possibilitam a realização de reconfigurações de alto nível com vista à melhoria da performance global dos processos de programação de máquinas industriais, nomeadamente em armazéns automáticos. Sendo uma realidade complexa e inovadora, não há, até ao momento, relevante implementação ao nível industrial. No entanto, diversas vantagens na utilização desta técnica são apontadas pela comunidade científica. A presente dissertação teve como objetivo investigar esta realidade, de forma a ser possível desenvolver uma aplicação para a geração automática de código para um PLC de controlo de um elevador automático. Foi desenvolvida em colaboração com a Consoveyo, S.A.. Este projeto foi testado num cenário simples, mas realista, para propósitos de observação.

O trabalho inicia-se com a análise de diferentes ferramentas, como o TIA Portal® e TIA Portal Openness® de forma a identificar os requisitos funcionais e de engenharia a adotar no projeto, bem como avaliar a possibilidade de concretizar o objetivo final do trabalho. Com base nos conhecimentos obtidos e nos princípios de funcionamento dos armazéns concebidos pela Consoveyo, S.A., é explorada uma abordagem que possibilite o desenvolvimento de uma aplicação de software.

Um programa modular para o controlo de um elevador automático foi desenvolvido, no TIA Portal®, que foi testado no simulador FactoryIO® e trabalhado usando métodos de manipulação de ficheiros XML (Extensible Markup Language) e o programa TIA Portal Openness®. No final do prazo da presente dissertação não foi possível obter uma aplicação operacional para a geração de código de controlo do PLC de um elevador. No entanto, com vista ao desenvolvimento da mesma, um layout para a interface foi proposto e concluíram-se os melhores métodos de manipulação dos ficheiros XML gerados, provenientes do programa modular referido anteriormente.

Palavras-chave: Armazém automático, elevador automático, programação de PLC, geração automático de código PLC.

Development of a software application for automatic generation of PLC code to control elevators

Abstract

More and more industries face new and different challenges, with the rising pressure complexity of modern production facilities and automated systems. In addition, the increasing pressure on costs and lead times also means that manufacturers need to find new ways to respond to these challenges. In order to shorten the programming phase of this type of system, automated code generation for Programmable Logic Controllers (PLCs) is proposed.

PLC code generation is a technique that introduces to the industry new intelligent engineering methods and technologies that allow high level reconfigurations to improve the overall performance of industrial machine programming processes, such as in automated warehouses. Being a complex and innovative reality, there is, to date, no relevant implementation at the industrial level. However, several advantages in the use of this technique are pointed out by the scientific community. This dissertation aims to investigate this reality in order to ease the development of a software application to automatically generate the PLC code that controls an elevator, and it was developed with the support of Consoveyo, S.A.. This project was tested in a simple but realistic scenario for observation purposes.

The work begins with an analysis of different tools, such as TIA Portal® and TIA Portal Openness® in order to identify the functional and engineering requirements to be adopted in the project, as well as to evaluate the possibility of achieving the final work objective. Based on the obtained knowledge and operating principles of the warehouses designed by Consoveyo, SA, an approach that enables the development of a software application is explored.

A modular program to control an automatic elevator was developed using TIA Portal®, which was tested in the FactoryIO® simulator and, later on, tooled using XML files manipulation methods and the TIA Portal Openness® software. At the end of the term of this dissertation, it was not possible to obtain an operational application for the generation of an elevator PLC control code. However, in order to its development, a layout for an interface was proposed and the best methods for manipulating the generated XML files, from the modular program mentioned above, are exposed.

Keywords: Automatic warehouse, automatic elevator, PLC programming, automatic PLC code generation.

Agradecimentos

A elaboração deste documento não seria possível sem o apoio e incentivo de determinadas pessoas, às quais me sinto extremamente grata pela ajuda que me proporcionaram neste percurso.

À minha mãe, que me possibilitou uma vida tão confortável, pela preocupação comigo e com o meu futuro, bem como por todo o suporte financeiro que disponibilizou de forma a que tal fosse possível.

À minha família, com ênfase na minha avó e avô, por todo o apoio e confiança que me transmitiram ao longo do meu percurso académico.

Ao Professor Douto Germano Veiga, pela partilha de conhecimentos e pela sua orientação ao longo do projeto.

Ao Engenheiro Vitor Vaz, pela oportunidade de realizar este projeto em colaboração com a Consoveyo S.A..

Ao meu coorientador, Henrique Domingos, por toda a paciência, dedicação, disponibilidade e acompanhamento prestado ao longo deste semestre.

À Consoveyo S.A. pela oportunidade que me deram de realizar este projeto, que em muito contribuiu para o meu crescimento como futura engenheira e por me terem recebido tão bem.

Aos colaboradores da Consoveyo S.A, em especial aos Engenheiros Vasco Sá e Miguel Pinto, pela sua constante disponibilidade e ajuda em todas as dúvidas que surgiram ao longo deste percurso.

Ao meu namorado, pela paciência, apoio constante, por nunca me ter deixado desistir, e ser o meu escape quando mais precisei.

Os meus sinceros agradecimentos a todos os que me acompanharam neste percurso de cinco anos e que, direta ou indiretamente contribuíram para o meu sucesso.

Índice de Conteúdos

Siglas

AGV - Automated Guided Vehicle
API - Application Programming Interface
CAD – Computer Aided Design
CAM – Computer Aided Manufacturing
CNC - Comando Numérico Computorizado
CPU – Central Power Unit
DLL - Dynamic-Link Library
FB – Function Block
FBD - Function Block Diagram
HMI - Human Machine Interface
HTML - HyperText Markup Language
IEC - International Electrotechnical Commission
IL – Instruction List
LD – Ladder Diagram
MDE - Model Driven Engineering
PIM - Platform Independent Model
PLC – Programmable Logic Controller
PSM - Platform Specific Model
R&D – Research and Development
RGV – Rail Guided Vehicle
SFC - Sequential Function Chart
SSI - Synchronous Serial Interface
ST – Structured Text
UML - Unified Modeling Language
VBA - Visual Basic for Applications
W3C - World Wide Consortium
XML – Extensible Markup Language

Índice de Figuras

1 Introdução

A evolução da indústria apresenta requisitos cada vez mais complexos para os fornecedores de soluções automáticas. Neste sentido, o uso de diferentes e mais poderosas abordagens à programação de PLCs torna-se naturalmente uma necessidade

A programação está na base do surgimento da era das tecnologias de informação. É uma ferramenta tão poderosa que parece que um computador é capaz de fazer tudo, estando limitado apenas pelos recursos de hardware e pelo conhecimento do programador. A introdução da programação de alto nível na indústria contribuiu para o desenvolvimento de soluções mais poderosas para os processos industriais. Permitiu não só enfrentar a crescente pressão sobre os preços e tempos de execução de soluções automáticas, mas também resolver alguns dos maiores problemas do sector.

Assim, o objetivo deste projeto passa por analisar como a geração automática de código estruturado pode ser uma vantagem para a indústria, usando-a para desenvolver programas para PLC que possam ser implementados diretamente e testados em cenários de acordo com os requisitos do cliente.

1.1 Enquadramento da dissertação e motivação

No contexto referido, surge o tema da presente dissertação: Desenvolvimento de uma aplicação para geração automática de código para o PLC de um elevador.

Este projeto foi desenvolvido em colaboração com a Consoveyo, S.A., especificamente no departamento de Automação de R&D (*Research and Development*). Realizada durante o segundo semestre do ano letivo de 2019/2020, contou com a orientação do Professor Doutor Germano Veiga e do Engenheiro Vitor Vaz da Consoveyo, S.A., e a coorientação de Henrique Domingos.

Os armazéns automáticos da Consoveyo, S.A. são constituídos por uma variedade de equipamentos e são concebidos de acordo com os diferentes requisitos de cada cliente, resultando numa elevada diversidade de configurações possíveis destes mecanismos. Como tal, o objetivo da empresa passa por reduzir os custos de engenharia e o tempo de execução, visando flexibilizar e agilizar o desenvolvimento de programas de PLC, de maneira a que estes possam ser adaptados às variações impostas em cada projeto e de acordo com as necessidades de cada cliente. A aplicação permitia endereçar estes problemas e, como tal, deverá ser o mais simples possível de maneira a facilitar a manipulação por parte do utilizador, evitando a necessidade de mão de obra especializada.

De entre os diferentes componentes de um armazém automático, esta tese incide sobre os elevadores, que são equipamentos geralmente utilizados em armazéns e permitem transportar cargas entre níveis diferentes do sistema de periféricos.

1.2 Desenvolvimento de uma aplicação para geração automática de código para o PLC de controlo de um elevador na Consoveyo, S.A.

Esta dissertação resulta de um projeto realizado em conjunto com o departamento de automação da empresa Consoveyo, S.A. e centra-se no desenvolvimento de uma aplicação que gera automaticamente o código do programa de controlo para PLC de um elevador (Consoveyo 2020a). A Consoveyo, S.A. integra o grupo Körber que é uma empresa líder de aplicações integradas para a otimização de processos complexos de logística interna ou externa (Consoveyo 2019a).

Líder mundial, a Consoveyo, S.A., é especialista em sistemas automáticos de transporte e armazenamento de materiais. Concebe e constrói soluções automáticas "chave na mão" para aplicações de intralogística tais como armazéns automáticos, sistemas de manuseamento e transporte interno e de preparação de encomendas. Como fornecedor de produtos, disponibiliza uma vasta gama de equipamentos, totalmente personalizados de acordo com os requisitos do cliente, e que são totalmente desenvolvidos nas instalações da empresa, desde o planeamento à integração no cliente, passando pelas fases de design, montagem e desenvolvimento do software necessário (Consoveyo 2019a). Dentro da gama de produtos podemos encontrar: elevadores e transelevadores, veículos guiados automaticamente (*Automated Guided Vehicles* – AGVs), veículos guiados sobre carris (*Rail Guided Vehicles* - RGVs), eletrovias, transportadores (*conveyors*) e carros de transferência (*shuttle car*) (Consoveyo 2020a). Para além de atuar como fornecedor de equipamentos e de produtos a Consoveyo, S.A. dispõe ainda de uma variada gama de serviços ao longo de todo o ciclo de vida do seu equipamento (Consoveyo 2019a).

O departamento de Investigação e Desenvolvimento, R&D, da Consoveyo, S.A. é responsável pelos novos projetos abraçados pela empresa, e está divido em Automação e Engenharia. É no departamento de Automação que vamos focar a nossa atenção: encarregue pelo projeto elétrico dos equipamentos e pelo desenvolvimento do software de controlo dos mecanismos. De momento, uma vez que a Consoveyo S.A. produz soluções variadas de acordo com os requisitos de cada cliente, cada projeto resulta na implementação de sistemas com diferentes layouts e princípios de funcionamento distintos, sendo que a seleção da "melhor solução" para cada sistema depende de vários fatores como a experiência e o estilo de trabalho do programador. Isto implica que, usando uma estratégia de reutilização de código, a cada novo projeto acolhido, é necessário interpretar e modificar manualmente os programas de controlo dos sistemas anteriormente implementados. Esta tarefa é bastante morosa e suscetível a erros. Posto isto, um dos objetivos do departamento passa por padronizar o estilo de programação utilizado nos seus sistemas e tem como finalidade flexibilizar e agilizar a programação dos vários equipamentos concebidos, de forma a diminuir os tempos de execução dos projetos e, consequentemente, aumentar a produtividade da empresa. Neste sentido, foi sugerido que se desenvolvesse uma aplicação de software que seja capaz de gerar, de forma automática, o código de controlo para PLC de um elevador, de acordo com os requisitos impostos e os seus princípios de funcionamento.

1.3 Objetivos do projeto

Como referido, os objetivos do projeto dizem respeito ao desenvolvimento de uma aplicação de software de geração automático de código para controlo do PLC para o

equipamento Elevadores da Consoveyo, S.A.. Como tal, referem-se os seguintes objetivos específicos:

- Analisar as funcionalidades operacionais de um elevador Consoveyo, incluindo a identificação de elementos reutilizáveis e opções mais frequentes.
- Estudar estratégias de modularidade para programação de PLCs.
- Estudar ferramentas de geração automática de código para PLCs Siemens.
- Desenvolver programa PLC modular para o controlo de elevadores Consoveyo.
- Desenvolver programa em linguagem de alto nível para a geração de código automática para PLCs Siemens.

1.4 Método seguido no projeto

Antes de abordar qualquer cenário é necessário perceber porque a geração automática de código é importante e vantajosa para a indústria de sistemas automáticos: o crescimento das empresas faz aumentar a competitividade e a necessidade de reduzir os custos e tempos de produção bem como o preço de fabrico dos seus produtos.

Devido ao caráter inovador do tema em questão e à reduzida quantidade de soluções que se mostram capazes de solucionar o problema apresentado pela empresa, havia alguma incerteza em relação à possibilidade de concretizar o último objetivo apresentado na secção anterior. Além disso, a Consoveyo, S.A., que se encontra de momento numa fase de padronização dos programas para as suas soluções, pretende passar toda a sua programação de Sharp 7 para TIA Portal®, sendo crucial que a ferramenta desenvolvida no final da presente dissertação seja compatível com as ferramentas e os procedimentos utilizados na empresa.

No início da presente dissertação, efetuou-se uma pesquisa bibliográfica acerca das possíveis abordagens ao problema da geração automática de código. A informação recolhida facilitou a contextualização do tema e permitiu identificar duas possíveis soluções: a geração de código automático a partir de uma aplicação em Microsoft Excel, programada em VBA (Visual Basic for Applications) e a geração de código recorrendo à manipulação de ficheiros XML.

Na exploração das opções encontradas, foi considerada, para cada ferramenta, a sua compatibilidade com as ferramentas e procedimentos utilizados na Consoveyo, S.A.. Assim, optou-se por uma solução cujo software de programação fosse o TIA Portal® e com utilização do TIA Portal Openness®.

O recurso ao TIA Portal Openness® implicou a aquisição de conhecimentos de linguagens de programação de alto nível como o C# e o XML. Além disso, foi necessária a familiarização com um dos softwares de programação de PLCs da Siemens, utilizado na Consoveyo, S.A., o TIA Portal®.

Paralelamente, um estudo sobre os sistemas de armazenamento desenvolvidos pela Consoveyo, S.A. foi iniciado, tendo-se desenvolvido um novo programa de controlo para o elevador, dado que a implementação do TIA Portal Openness® obrigava à existência de um programa exclusivamente em TIA Portal®. Esse novo programa foi desenvolvido tendo em conta a modularidade mais tarde necessária à geração de código, separando-se o código por blocos que poderiam ser reutilizados ao longo do programa, e tendo em conta, não só os princípios de funcionamento dos sistemas desenvolvidos pela empresa, mas também as variações mais comuns na configuração dos mesmos.

Posto isto, e uma vez desenvolvido o programa de controlo do elevador e as competências necessárias à utilização do software TIA Portal Openness®, iniciou-se o desenvolvimento da aplicação, tendo por base a possibilidade de alterar o "programa base" a partir da manipulação de documentos XML. Assim sendo, a aplicação resultante deverá partir de um programa base em TIA Portal®, manipulá-lo e gerar o código do programa de controlo do elevador de acordo com a configuração única desejada, dentro de uma gama de variações.

Apesar da aplicação em C# não ter sido concluída de maneira a operar corretamente até ao final do prazo da presente dissertação, uma interface para a mesma é proposta e várias funções e procedimentos necessários à sua conclusão são explicitados, tornando possível que a mesma seja futuramente concluída.

1.5 Estrutura da dissertação

Uma breve descrição de todos os capítulos encontrados nesta tese será apresentada a seguir. No final do documento estão incluídas as várias referências bibliográficas consultadas e os anexos.

Capítulo 2

Neste capítulo introduzem-se os dois componentes importantes do sistema tratado no projeto: elevadores integrados em armazéns automáticos e PLCs. Para além disso, uma pesquisa é feita e são estudadas abordagens e ferramentas existentes, que possibilitem atingir o objetivo final da dissertação (geração automática de código).

Capítulo 3

O elevador, sistema cujo controlo é pretendido é, neste capítulo, descrito e o seu funcionamento explicado. É aqui também que se explora de forma mais profunda algumas ferramentas consideradas com potencial para a resolução do problema proposto. Realça-se uma dissertação realizada em 2018, com um tema bastante semelhante ao apresentado nesta dissertação.

Capítulo 4

Neste capítulo é explicado o programa concebido no TIA Portal® para o controlo de um cenário "tipo" de um elevador com 3 ou 2 pisos, desenvolvido no ambiente de simulação Factory IO®.

Capítulo 5

Este capítulo é focado no desenvolvimento da aplicação em C# para a geração de código propriamente dita. Podem encontrar-se aqui os objetivos da aplicação, assim como a interface proposta e o funcionamento pretendido para a mesma.

Capítulo 6

As principais conclusões retiradas desta tese são apresentadas neste capítulo, seguidas de algumas sugestões de trabalhos futuros.

2 Estado da Arte

O PLC e as suas práticas de programação sofreram grandes transformações desde 1960. Compreender a evolução e a história dos PLCs e suas metodologias de programação é essencial para a organização, execução e manutenção de projetos baseados em PLCs. Este equipamento evoluiu para acompanhar as necessidades do mercado procurando melhores soluções de hardware e software. Contudo, não foi ainda desenvolvido, a nível comercial, um método de programação de PLCs formal e sistemático, de forma a que o programa não esteja sujeito à experiência do programador.

Neste capítulo, em primeiro lugar, optou-se por fazer uma breve exposição sobre sistemas de elevação em armazéns automáticos, uma vez que o elevador alvo desta dissertação é parte integrante de um desses sistemas. Em segundo lugar, enquadram-se os PLCs no ambiente da automação industrial, fazendo uma breve descrição do seu funcionamento geral e história, para além de um enquadramento da norma vigente – o IEC61131 - no desenvolvimento de PLCs. Por último, abordam-se alternativas existentes de soluções para geração e flexibilização da programação de PLCs.

2.1 Sistemas de elevação em armazéns automáticos

A integração de sistemas de armazenamento automáticos nas indústrias tem vindo a aumentar, uma vez que esta promove o aumento da produtividade, reduzindo o número de trabalhadores necessários no armazenamento e recolha de objetos do armazém. Ao contrário dos sistemas manuais, constantemente sujeitos a erros humanos, estes sistemas conferem uma elevada taxa de precisão e fiabilidade (Parsons et al. 1992).

Os elevadores automáticos (Figura 1), foco da presente dissertação, fazem parte do conjunto de equipamentos que podem ser integrados nos armazéns desenvolvidos pela Consoveyo, S.A..

Os elevadores produzidos pela Consoveyo, S.A. são um elemento de transporte vertical de unidades de carga (subida e descida de paletes entre dois níveis distintos do sistema de transportadores), estando dispostos em dois ou mais pisos. Estes equipamentos estão geralmente presentes num circuito de tapetes de rolos de um armazém de logística, servindo de ligação entre os diferentes níveis de altura do mesmo.

São equipamentos com dois modos de operação possíveis: automático ou manual. Quando o modo automático está selecionado, o elevador tem um gestor (implementado pelo PLC) que controla os sistemas e recebe os comandos que determinam que tarefa vai ser executada. Por outro lado, quando o modo manual é selecionado, o controlo do sistema é local e feito através de uma consola (Consoveyo 2019b).

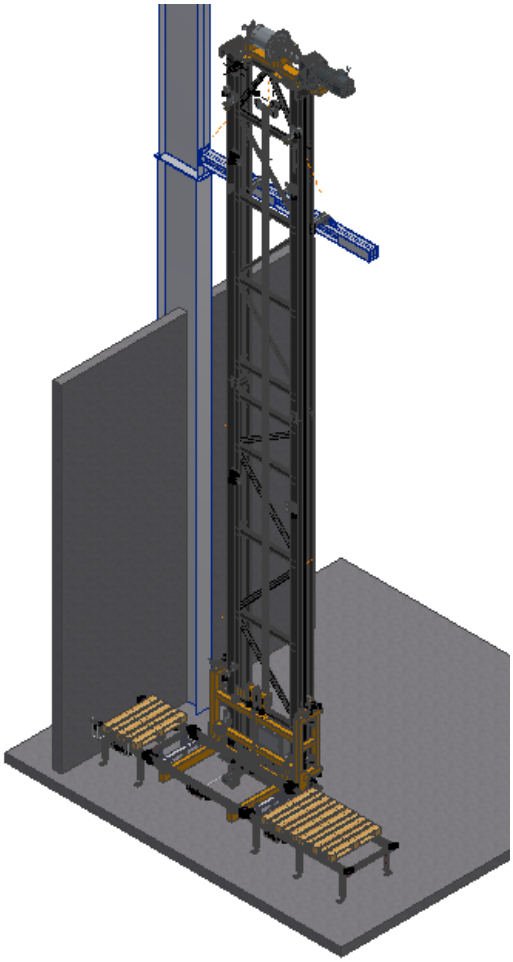


Figura 1 - Exemplo de elevador produzido pela Consoveyo.

2.2 Controladores Lógicos Programáveis

PLCs são computadores industriais, concebidos de forma robusta e capazes de controlar vários tipos de equipamentos industriais e sistemas automatizados, como linhas de montagem, dispositivos robóticos ou sistemas de armazenamento (Alphonsus e Abdullah 2016). Este equipamento monitoriza continuamente o estado dos dispositivos de entrada e toma decisões baseadas num programa personalizado de forma a controlar o estado dos dispositivos de saída (Cheng et al. 2014).

Estes equipamentos, originalmente desenvolvidos pela indústria automóvel, no fim de 1960, com o objetivo de substituir os circuitos cablados (com relés e temporizadores) por um sistema mais flexível, são atualmente uma das peças mais importante de um sistema de automação (Alphonsus e Abdullah 2016).

Os PLCs têm como vantagens: a robustez, imunidade ao ruído elétrico, resistência à vibração, ao impacto e a amplos intervalos de temperatura e humidade. Para além disso, são considerados sistemas em tempo real: os sinais de saída são gerados ou alterados em resposta a condições de entrada num intervalo de tempo limitado (Advanced Micro Controls 2020). São agora indispensáveis e o dispositivo de preferência para controlo de sistemas industriais em muitas empresas (Hajarnavis e Young 2008).

2.2.1 Norma IEC 61131

No passado, vários fabricantes, desenvolviam PLCs com diferentes tempos de execução, sistemas operacionais e linguagens de programação. Em 1992, para reduzir a complexidade da utilização de PLCs e harmonizar as diferentes visões e filosofias dos diferentes fabricantes, a Comissão Eletrotécnica Internacional (IEC) elaborou a norma IEC 61131, a principal norma de referência dos PLCs (Otto e Hellmann 2010).

A norma está dividida em diversas partes (International Electrotechnical Commission 2020):

- Parte 1: Informação geral;
- Parte 2: Requisitos e testes para os equipamentos;
- Parte 3: Linguagens de programação;
- Parte 4: Guia de implementação para utilizadores;
- Parte 5: Especificações para a comunicação;
- Parte 6: Segurança funcional;
- Parte 7: Programação *Fuzzy Control*;
- Parte 8: Guias de aplicação e implementação das linguagens de programação;
- Parte 9: Interface de comunicação digital *Single-Drop* para pequenos sensores e atuadores;
- Parte 10: Formatos de intercâmbio XML para PLC.

2.2.1.1 IEC 61131-3: Arquitetura do software

A norma IEC tinha o objetivo de resolver as diferenças na abordagem de programação dos sistemas de controlo de PLC oferecidos pelos principais fornecedores, tendo em conta uma previsão do impacto que tecnologias emergentes e futuros requisitos de controlo viriam a ter sobre o software do PLC e as estruturas de linguagem. A arquitetura do software proposta pela norma (Figura 2) deve ser capaz de acomodar esses requisitos futuros sem grandes mudanças estruturais (IDC Technologies 2002).

De acordo com a norma, um programa a ser executado por um PLC requer os seguintes tipos de interfaces:

- Interface I/O: conexão do PLC ao mundo físico;
- Interface de sistema: serviços necessários para garantir a execução de um programa PLC;
- Interface de comunicação: troca de dados com outros dispositivos e PLCs.

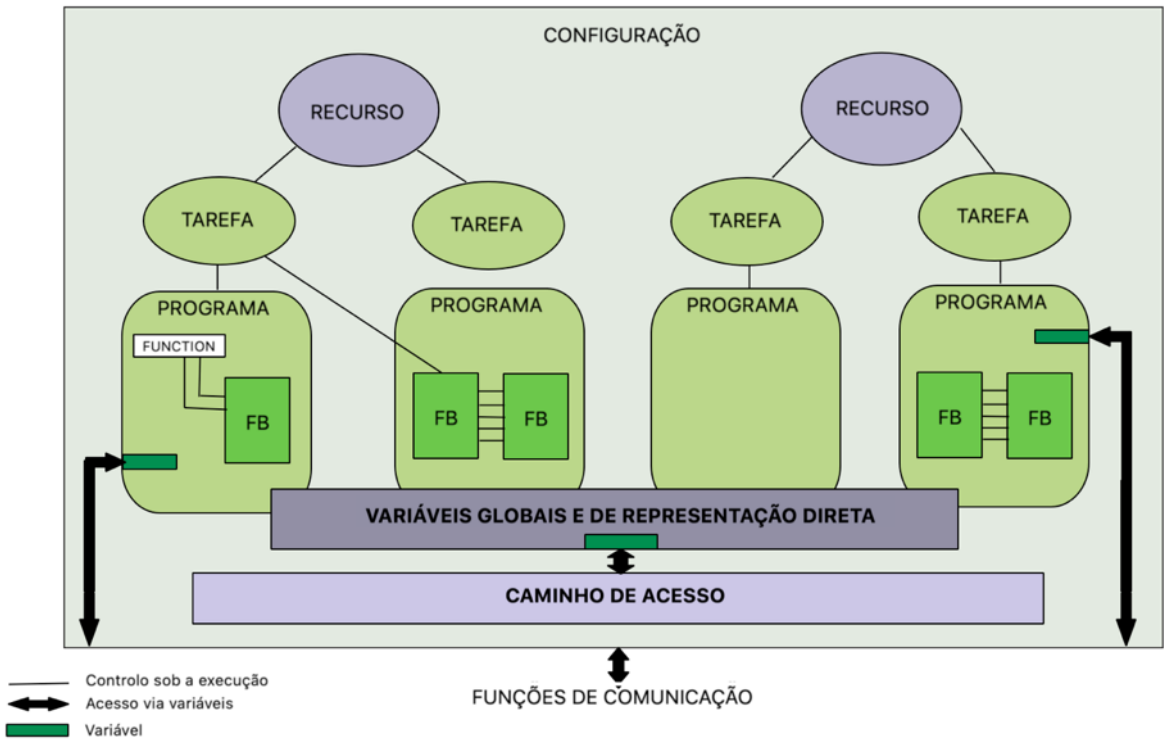


Figura 2 - Arquitetura do software descrita pela norma IEC 61131-3. In Andreas Otto and Klas Hellmann, IEC 61131: A General Overview and Emerging Trends, 2010.

Como se pode verificar a partir da observação da Figura 2, o modelo de software sugerido contém vários elementos organizados hierarquicamente. Segue-se a descrição de cada um desses elementos.

Configuração

Pode ser equiparado ao software da aplicação de controlo de um PLC. É constituído por vários recursos, que agrupa no sistema de controlo do PLC, e fornece meios para troca de dados entre eles.

Recurso

São Unidades Central de Processamento (*Central Process Units* – CPU), mecanismo que suporta e direciona a execução de uma ou mais tarefas. Uma configuração pode ter um ou vários recursos.

Tarefa

Elementos de software utilizados pelo recurso e que podem invocar um ou mais blocos de funções ou programas a serem executados de acordo com as suas especificações ou com uma certa periodicidade.

Unidades de Organização do Programa

Program Organisation Units (POUs), blocos base da estrutura da programação, incluem programas, *functions* e *function blocks.*

Programa

Conjunto lógico de todos os elementos de programação, constituído por todos os componentes e construções da linguagem de programação, funções (sem memória interna) e

blocos de funções (com memória interna). Contém instruções a serem executadas pelo hardware do PLC numa sequência predefinida.

Function

Pode ter vários *inputs,* mas fornece um único *output*. As *functions* não possuem memória interna.

Function Block

Ajuda na estruturação do programa, organizando um problema complexo em várias etapas mais simples. Além disso, em casos de sequências lógicas idênticas, podem ser criadas instâncias de um *function block* específico e utilizados com diferentes paramentos. Estes elementos definem um conjunto de *inputs, outputs* e variáveis internas, que, ao contrário das *functions* podem ser armazenadas numa memória interna.

Variáveis locais e globais

As variáveis locais podem ser chamadas dentro de uma configuração, programa, *function block* ou *function*, dependendo do elemento de software no qual são declaradas. Se a variável for global, esta estará acessível a todos os elementos de software dentro desse recurso ou configuração e fornecem um método de troca de dados entre programas ou entre elementos de um programa.

Variáveis de representação direta

Usadas para endereçar diretamente aos locais da memória de um PLC.

Caminho de acesso

Variáveis para fins de comunicação, que podem ser acedidas por configurações remotas.

2.2.1.2 IEC 61131-3: Linguagens de programação

Apesar das capacidades do PLC terem melhorado ao longo do tempo, o seu histórico como substituto dos sistemas elétricos cablados proporcionou a que uma das linguagens dominantes de programação deles permanecesse a linguagem ladder por esta se assemelhar bastante a diagramas de fios elétricos (Figura 3), familiar aos engenheiros elétricos (Hajarnavis e Young 2008). No entanto, dada a imensa evolução destes equipamentos, as capacidades básicas de programação desta linguagem deixaram de ser suficientes, obrigando os fabricantes a desenvolver novas opções. Este progresso resultou numa multiplicidade de linguagens de programação, tornando-se complexo para os utilizadores a aprendizagem das mesmas e difícil a operabilidade entre PLCs de diferentes fabricantes (IDC Technologies 2002).

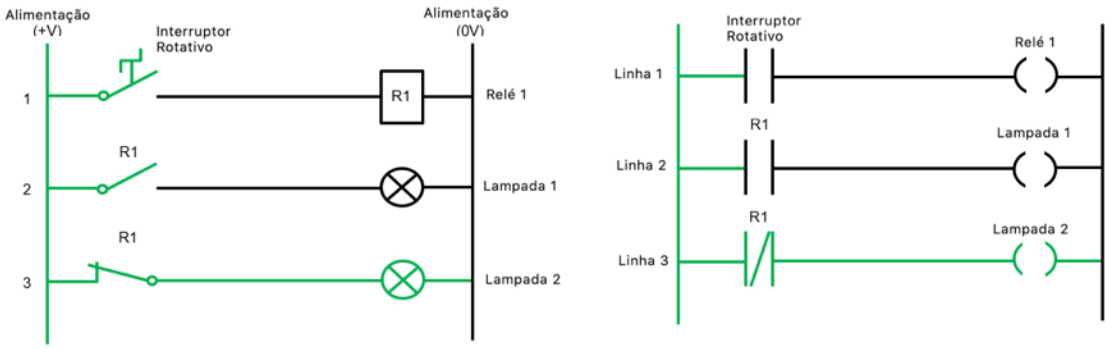


Figura 3 - Esquerda: Diagrama elétrico; Direita: Linguagem Ladder. In https://ladderlogicworld.com/relay-logic-vs-ladder-logic/

Os fabricantes de PLC rapidamente perceberam que o futuro crescimento destes equipamentos e seu amplo uso na indústria não seria possível, a menos que a questão fundamental da portabilidade do programa fosse abordada. Assim, houve uma mudança para uma abordagem de programação uniforme a ser adotada por todos os fornecedores por meio de um padrão básico de programação (IDC Technologies 2002). Em resultado desse esforço, conhecida como IEC 61131-3, surge a norma que define um modelo de desenvolvimento comum de programação de PLC, onde se incluem 5 linguagens de programação (Hajarnavis e Young 2008; IDC Technologies 2002):

- Textual:
  - Lista de Instruções (*Instructions List* - IL) – linguagem de baixo nível que consiste em códigos de operação simples; cada instrução está localizada numa linha diferente;
  - Texto Estruturado (*Structured Text* - ST) - linguagem de alto nível (sintaxe semelhante a C ou PASCAL), mais acessível a programadores familiarizados com as linguagens de programação tradicionais; definido por comandos e expressões;
- Gráfica
  - Diagrama de Blocos (*Function Block Diagram* - FBD) – baseada na representação de um sistema em termos do fluxo de sinais entre os elementos de processamento;
  - Diagrama de Contactos Elétricos (*Ladder Diagram* - LD) – representação gráfica semelhante a conexões elétricas, de variáveis booleanas (relés e bobinas);
- *Sequential Function Chart* (SFC) – linguagem simples e intuitiva, derivada das redes de Petri; constituídos por etapas - estado do sistema - e transições - um elemento que valida a mudança de uma etapa para a seguinte; normalmente usada para descrever comportamentos sequenciais.

Apesar de os fabricantes de PLC terem adotado estas linguagens, geralmente a utilização das mesmas não segue nenhuma forma de utilização padrão no que toca à perspetiva da estrutura de codificação, o que resulta em programas possivelmente confusos e não coerentes (Jbair et al. 2019).

A padronização das linguagens de programação tenta incluir diferentes opções de modo a abranger e adaptar, tanto quanto possível, o conhecimento já adquirido por profissionais do setor (Garcia 2017).

Fruto da elevada variada escolha de linguagens normalizadas, o programador tem a possibilidade de selecionar a mais apropriada para a sua aplicação. Essa escolha vai depender da habilidade do programador, a natureza da aplicação a ser programada, o nível e a estrutura do problema assim como a frequência da necessidade de correção de erros e diagnósticos (Hajarnavis e Young 2008).

2.2.1.3 *PLCOpen* – Expandindo a norma

Formada em 1992, imediatamente após o lançamento da norma IEC 61131-3, o *PLCOpen* é uma organização cujo objetivo é apoiar e fornecer eficiência em automação industrial com base nas necessidades dos utilizadores, focando-se na harmonização da programação de PLCs e desenvolvimento de software e interfaces (PLCOpen 2020).

Desde o lançamento da norma 61131-3, vários ambientes de desenvolvimento, usados para criar e editar código de controlo de PLC de acordo com a norma, foram criados por uma vasta gama de fornecedores. Com isto, surge o desejo dos utilizadores e programadores de poderem trocar os seus programas, bibliotecas e projetos entre esses diferentes ambientes de desenvolvimento (PLCOpen 2009).

2.3 Importação/Exportação em XML

Apesar de prever a portabilidade de código, a norma IEC 61131-3 não define nenhum formato para a importação/exportação de projetos entre diferentes ambientes de desenvolvimento. Com o principal objetivo de transferir um projeto, de acordo com a norma e sem trabalho adicional, de uma ferramenta de desenvolvimento para outra sem perder informações, a *PLCOpen* propôs, em 2005, um *schema* XML, que foi rapidamente implementado pela indústria (PLCOpen 2020).

O XML é uma linguagem *markup*, desenvolvida pelo *World Wide Consortium* (W3C) semelhante ao HTML (*Hypertext Markup Language*), que define um conjunto de regras para codificação de dados num formato de texto, numa estrutura hierárquica, legível tanto por humanos como por máquinas (Smith 2019b).

Esta linguagem permite a troca, armazenamento e partilha de informação de forma independente de qualquer software e hardware, entre sistemas incompatíveis (ou para versões atualizados dos mesmos). Um documento XML é apenas um ficheiro com informação organizada em *tags*, não contendo qualquer instrução acerca da apresentação dos dados (W3Schools 2020).

2.3.1 Estrutura e sintaxe do XML

O documento XML é construído segundo uma estrutura em forma de árvore (Figura 4a)). A árvore começa num elemento raiz (*root element*) e ramifica dessa raiz para elementos filhos (*child elements*). Os elementos, que tanto podem estar vazios como podem incluir texto, podem conter atributos, que servem para conter dados relacionados com um elemento específico, ou outros elementos (W3Schools 2020). Na Figura 4b) pode observar-se um exemplo de um documento XML de acordo com a árvore definida no esquema da Figura 4a).

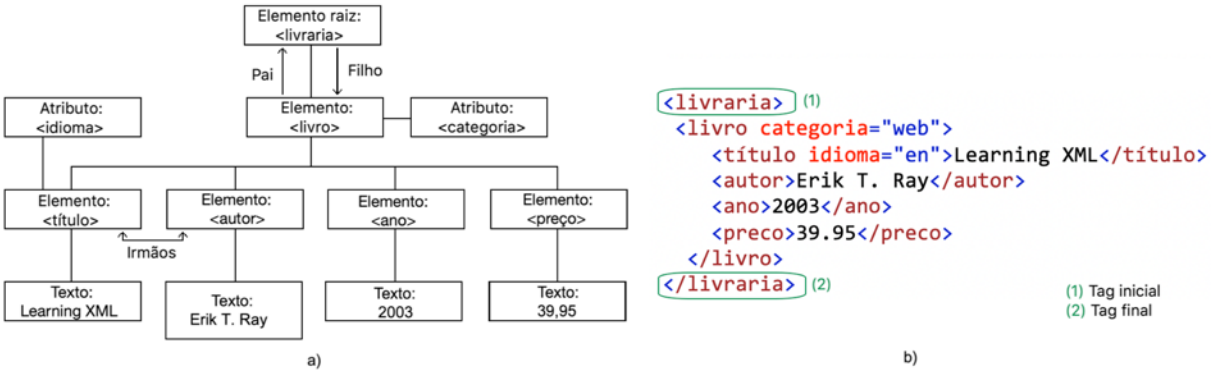


Figura 4 - Ficheiros XML: a) Estrutura em árvore de um documento XML; b) Exemplo documento XML

A construção de documentos XML é feita respeitando algumas regras de sintaxe, podendo-se destacar as seguintes (Walsh 1998; W3C 2008):

- Geralmente, o documento começa com um prólogo (Figura 5), designado por declaração XML. Esta linha, não obrigatória, identifica o documento como um documento XML e indica a versão e a codificação usados na escrita do ficheiro.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Figura 5 - Declaração XML

- Todos os elementos de um XML têm uma *tag* inicial e uma *tag* final (Figura 4b).
- As tags num ficheiro XML são sensíveis a letras maiúsculas e minúsculas. A *tag* <Letra> é diferente da *tag* <letra>. Como tal, as *tag* inicial e final devem ser escritas no mesmo estilo.
- Os comentários começam com <!-- e acabam com -->, como exemplificado na Figura 6.

```
<!-- Isto é um comentário -->
```

Figura 6 - Exemplo de comentário num documento XML

2.3.2 XML Schema

Um XML *schema* (Figura 7), igualmente escrito em linguagem XML, consiste numa descrição de um tipo de documento XML, que inclui os termos de validação de tais documentos, indicando características como (PLCOpen 2009):

- Estrutura do documento: quais elementos são elementos filhos de outros, a ordem em que os elementos filhos podem estar, o número de elementos filhos;
- Tipo de conteúdo/dados;
- Imposições sintáticas e gramaticais.

Através destas características é possível submeter os ficheiros a um processo de validação, aceitando apenas documentos XML que obedeçam aos requisitos funcionais para o sistema computacional funcionar adequadamente (W3Schools 2020).

```
<xs:element name="note">

<xs:complexType>
  <xs:sequence>
    <xs:element name="to" type="xs:string"/>
    <xs:element name="from" type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

</xs:element>
```

Figura 7 - Exemplo de XML Schema

2.4 Automatização dos processos de programação na indústria

Ao longo das últimas décadas, o nível de exposição das indústrias ao mercado global tem vindo a aumentar, o que se traduz num aumento da concorrência. Simultaneamente, a procura do cliente por produtos customizados com curtos prazos de fornecimento tem também aumentado.

Neste sentido, as indústrias têm apostado na integração de soluções automatizadas como forma de reforçarem a sua competitividade. Estas soluções contribuem para a redução dos custos operacionais e para um maior dinamismo face aos desafios e condicionantes do mercado atual. Através da sua implementação, verifica-se um aumento da eficiência, flexibilidade/customização e produtividade das estruturas de produção, bem como uma redução dos custos do produto (Armentia et al. 2018); (Cheng et al. 2014). Como os custos de hardware são considerados fixos, a contribuição das soluções automatizadas foca-se preferencialmente na redução dos custos nas fases de instalação e projeto de software (Mastilovich 2010). Além disso, segundo um estudo da empresa AIDA (2005), os custos e tempos de implementação destas fases representam a maioria dos mesmos num projeto de conceção de um sistema industrial de produção.

A reutilização de soluções e código é considerada uma das soluções mais eficientes na redução dos custos anteriormente referidos, contribuindo para a melhoria do método de programação manual aplicado atualmente pelos profissionais da área (Takada, Nakata e Horiike 2004). A seleção das soluções a reutilizar estão dependentes da experiência dos técnicos e da qualidade das soluções desenvolvidas anteriormente, por outros intervenientes. A desvantagem desta prática reside no esforço significativo de trabalho manual ainda assim requerido (Güttel, Weber e Fay 2008).

A geração de código de forma automatizada é outra alternativa explorada na indústria com vista à redução dos custos e tempos de implementação. Embora não seja uma novidade neste meio, para o caso particular dos PLCs começam apenas a surgir as primeiras abordagens, não havendo nenhuma metodologia amplamente utilizada (Han 2010).

Contrariamente, para equipamentos como máquinas de comando numérico computorizado (CNC), a criação de programas pode ser conseguida automaticamente com recurso a geradores de código que, através de ficheiros CAD (Computer Aided Design), sistemas CAM (Computer Aided Manufacturing) e pós-processadores são capazes de gerar todo o código G necessário para o fabrico de uma peça (Barbosa 2019). Também na área de IT, as ferramentas de geração de código são amplamente utilizadas com várias finalidades. É o caso dos compiladores, que através de um gerador de código, convertem o código fonte (representado por linguagens de programação modernas) em código de máquina. Além disso,

utilizam-se ainda geradores de código para gerar classes e métodos de modo a evitar a repetição de processos por parte do programador, facilitando e agilizando o seu trabalho (Techopedia 2016).

2.4.1 Geração automática de código para PLCs

Como referido anteriormente, a geração automática de código com vista à agilização da programação de PLCs e do projeto de sistemas industriais é uma abordagem recente. Deste modo, verificou-se uma escassez de publicações científicas que apresentassem metodologias concretas para fazer face a este problema.

Assim optou-se por apresentar brevemente diversos estudos que abordam o problema de um ponto de vista conceptual e posteriormente complementar com exemplos de ferramentas existentes que procuram contribuir para a problemática em causa. Neste último ponto, será abordado em maior detalhe um estudo elaborado por Domingos (2018) focado inteiramente nesta temática.

Ressalve-se que grande parte das abordagens apresentadas não foram consideradas por não sugerirem soluções completas e práticas. Jbair et al. (2019) alerta para a mesma problemática, acrescentando que a maioria dos estudos existentes carecem de implementação industrial, ignoram as *best practices* da programação de PLCs e não preenchem os requisitos operacionais e de manutenção das aplicações industriais.

2.4.1.1 Metodologias de abordagem

A **engenharia orientada a modelos** (MDE – *Model Driven Engineering*), amplamente considerada na generalidade das aplicações na área de *IT* é apresentada por Qamsane et al. (2018), Thramboulidis (2012), Thramboulidis e Frey (2011) e Vogel-Heuser, Witsch e Katzke (2005) como uma possível metodologia para a geração de código de PLCs através de linguagens de modelação. Por sua vez, Darvas, Viñuela e Majzik (2016) apontam como limitações à implementação de MDE o facto das linguagens de programação de PLCs e de os programadores dos mesmos terem geralmente conhecimentos insuficientes acerca de engenharia de software. No entanto, Schumacher e Fay (2014) e Schumacher, Schröck e Fay (2013) concluíram ser possível gerar código para PLCs utilizando linguagens como o UML, SysML ou GRAFCET.

Outros autores (Fernández Adiego et al. (2015), Frey e Litz (2000), (Hasdemir e Kurtulan 2006) e Thapa et al. (2009)) introduziram uma metodologia automatizada geral para a verificação formal de programas PLC. Os **métodos formais** (Figura 8 – percurso interior) permitem fazer a especificação formal do controlador a partir de uma descrição informal do processo não controlado e dos requisitos do sistema controlado, utilizando técnicas de verificação e validação (Darvas, Viñuela e Majzik 2016); (Frey e Litz 2000). A maioria dos estudos existentes sobre geração de código PLC baseada em métodos formais utilizam ferramentas como as Redes de Petri, teoria dos autómatos ou t-MPSG e explicam teoricamente a viabilidade de especificação, projeto e implementação de controladores lógicos. Contudo, são geralmente adequados a sistemas pequenos (Thapa et al. 2009).

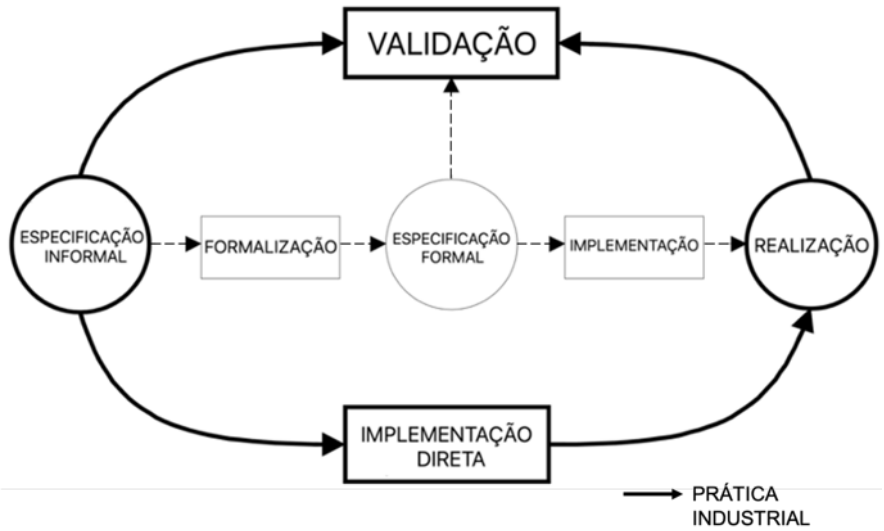


Figura 8 - Modelo do processo de controlo lógico. In Frey e Litz, Formal Methods in PLC programming, 2000.

Güttel, Weber e Fay (2008) procuraram um conceito para gerar automaticamente o código de controlo relativo às funções de segurança e sequências de inicialização e paragem. Isto porque, segundo os autores, na maioria dos sistemas de geração automática, o código gerado serve apenas para o controlo da sequência nominal do processo de produção.

Güttel, Weber e Fay (2008) defendem ainda que, de modo a implementar um mecanismo de geração de código, é necessária a existência de bibliotecas de *function blocks* padronizados e construidos de forma modular para os diferentes componentes da instalação. Além disso, recomendam que a estrutura da instalação, a experiência dos profissionais e a descrição do processo estejam disponíveis num formato legível por computador. Para esta última, sugerem a utilização da linguagem SFC (IEC 61131-3) representada no formato PLCOpen-XML. O código PLC resultante deste processo deverá ser transparente e extensível, isto é, com uma estrutura bem definida e facilmente escalável, capaz de evoluir conforme o processo e produto.

2.4.1.2 Casos de estudo

Recurso ao software Siemens Automation Designer®

De acordo com Andersson e Helander (2010), a utilização de simulações virtuais para examinar o comportamento dos sistemas constitui uma forma de reduzir o tempo de preparação de sistemas automáticos. Contudo, para a implementação da simulação, a totalidade lógica é requerida, não podendo esta ser transferida diretamente para o PLC, o que faz com que seja necessária uma transferência manual, resultando em trabalho em duplicado. Utilizando o software *Automation Designer*® da Siemens, foi possível reutilizar dados de projetos anteriores e aplicá-los nos mais recentes.

O *Automation Designer* é destinado a engenheiros elétricos e de automação do setor discreto e que, várias vezes, trabalham em equipa. Ao focar na reutilização, engenharia baseada em regras e padrões e integração de dados, o software aperfeiçoa a consistência dos projetos, o que resulta na redução de tempo e custo investidos neles e aumenta a produtividade de todo o processo de engenharia inerente. Este software, além de permitir a geração de programas para PLCs, permite ainda a geração de esquemas elétricos e HMIs, a partir da importação de ficheiros XML para os softwares WinCC® e EPLAN Electric P8®, respetivamente (Siemens 2016b).

Assim, no caso estudado, modelos reutilizáveis foram criados, com informação que se mantinha invariável ao longo do projeto, para o equipamento usado pela empresa, e armazenados numa base de dados. Com base nesses modelos, é gerado código para o software SIMATIC S7® da Siemens bem como o hardware e a *symbol table* necessários ao projeto (Andersson e Helander 2010). A interface criada é constituída por um projeto base, que atua como uma base de dados, na qual são inseridos objetos (por exemplo, um robôt), caracterizados por atributos (como informações de hardware e código PLC) (Falkman, Helander e Andersson 2011). Este projeto base pode ser considerado um modelo a partir do qual novos projetos serão construídos, pelo que é importante que a sua estrutura esteja bem definida. Uma variedade de funcionalidades é garantida pela inserção do mesmo objeto em diferentes *templates* mas configurando os seus atributos de formas distintas (Andersson e Helander 2010).

A sequência geral de coordenação do sistema não está incluída no código gerado, tendo que ser ainda realizada manualmente, o que dificulta a validação do código. Conclui-se que o projeto de PLC gerado pode funcionar como um projeto base melhor do que o usado pela empresa nos dias atuais, mas não é eficiente como projeto final (Falkman, Helander e Andersson 2011). Além disso, embora o projeto descrito tenha ido de encontro às necessidades da empresa, não foi possível obter o *software* em questão de forma a explorar as suas capacidades.

Recurso ao software Tia Portal Openness®

O TIA Portal Openness® é uma extensão do TIA Portal® que permite aceder, remotamente, às funções deste segundo, através de uma interface (Siemens 2016a).

De facto, a padronização de projetos segue uma sequência definida de forma a encontrar a solução ideal específica para cada cliente, dado que os fabricantes de equipamentos automáticos produzem sistemas com vários requisitos diferentes. Assim a Siemens (2020) sugere a utilização das bibliotecas do TIA Portal® como forma de criar um fluxo contínuo de trabalho (Figura 9).

O primeiro estágio para a geração de uma fração significante de código um programa de controlo de um sistema, passa pela da criação e armazenamento de *function blocks* padronizados, responsáveis pelo controlo de funções que se irão repetir dentro do mesmo equipamento, funcionando como "unidades reutilizáveis" (Siemens 2020). Assim, a Siemens afirma que, através de uma aplicação criada com o auxílio do TIA Portal Openness®, se recolhe a informação necessária à definição do sistema a controlar (dependendo das variantes a implementar). Com essa informação, podemos instanciar os *function blocks* pré-definidos, num projeto do TIA Portal®, tantas vezes quanto necessário, e reuni-los, resultando daí um projeto programado para controlar sistemas com configurações modulares (Siemens 2017).

Figura 9 - Fluxo de trabalho contínuo proposto pela Siemens. In Siemens, Standardization, 2020.

Uma vez que os blocos pré-definidos são previamente testados, este processo permite não só reduzir significativamente o tempo de programação dos equipamentos, mas também evitar erros humanos na criação de programas de controlo (Siemens 2017).

Apesar das vantagens apresentadas, este método implica que a lógica do equipamento resulte apenas da implementação dos mesmos blocos, várias vezes, reduzindo a sua utilidade a equipamentos em que o número de variações da configuração seja previsível e reduzido.

Geração de código a partir do Microsoft Excel e VBA

As folhas de cálculo do MS Excel fazem parte do passado e presente industrial sendo utilizadas maioritariamente por pessoal não programador. Por sua vez, o *Visual Basic for Applications* (VBA), é uma ferramenta de programação integrada no MS Excel, podendo ser utilizada para automatizar processos e funções, geralmente usada pelos programadores. O VBA pode ser considerado uma base do MS Excel, oculto para a maioria das pessoas. As macros são uma versão automatizada da programação do VBA, permitindo apenas programar funções simples, sendo assim significativamente menos poderosas que a programação direta no espaço do VBA. Nas abordagens que se seguem ambas as partes do MS Excel serão usadas: as folhas como uma interface do utilizador - utilizadas pelo engenheiro para inserir ou selecionar as informações necessárias para o software para criar o código PLC - e a parte VBA para programar as funções da aplicação software com a informação inserida na folha (Blagojevic et al. 2019; Mastilovich 2010).

Na Consoveyo, S.A. também já se adotou uma estratégia que pretendia diminuir a carga de trabalho manual dos programadores, usando o Microsoft Excel para a implementação dos comandos de geração automática de código para PLC. Nesse caso, a informação inserida no Excel provém de uma lista de definições, previamente preparada pelo engenheiro de sistema, que contém informação detalhada acerca dos mecanismos fabricados pela empresa e que serão controlados pelo PLC, como transportadores. Com estes dados, e com auxílio de uma macro criada no VBA, a ferramenta gera maioritariamente *data blocks* onde são guardados por exemplo, os estados do transportador, a informação lógica da palete que transportam, etc... Esta

abordagem permite também a geração de alguns *function blocks*, porém com pouca lógica associada, isto é, correspondentes a funcionalidades muito básicas, mas necessárias aos transportadores. A lógica necessária ao funcionamento do mecanismo é, portanto, inserida posteriormente de forma manual, mas com a informação contida nas DBs geradas.

Tal como se confirmou pela experiência da Consoveyo, S.A., essa abordagem servia apenas para gerar frações bastante simples de código, caindo em desuso por não ser considerada muito útil. Pesquisas académicas de autores como Blagojevic et al. (2019) e Mastilovich (2010), aprofundam esta abordagem, sendo importante referir que ambas foram realizadas como objetos de controlo de atuadores pneumáticos. Assim, os seus resultados favorecem aplicações em que o sistema tenha sempre os mesmos constituintes, mudando apenas o seu comportamento de um projeto para o outro. Para além disso, segundo Pluške (2017), o Excel apresenta um baixo nível de automação, o que resulta em dificuldades no caso de se pretender trabalhar com projetos complexos, como é o caso do tema da presente dissertação.

Geração de código a partir da manipulação de documentos XML no Microsoft Visual Studio®

Tal como referido no Capítulo 2.4.1, nesta secção será abordado um estudo recente realizado por Domingos (2018), desenvolvido igualmente em colaboração com a Consoveyo, S.A.

Domingos (2018) aborda a possibilidade de gerar automaticamente código para o programa PLC de controlo de um *shuttle car*, equipamento utilizado pela empresa nos seus armazéns automáticos. A presente dissertação apoiou-se na abordagem e rumo traçado neste estudo, dado tratar-se de um trabalho bastante completo e com objetivos próximos e em contexto semelhante.

Neste estudo, o tema da geração automática de código foi generalizado e exploraram-se as possibilidades no que diz respeito ao ramo da flexibilização da programação de PLCs. Para isso o autor efetuou uma pesquisa bibliográfica intensiva, explorando diversos casos de estudo, alguns mencionados na Secção 2.4.1.1. Desta pesquisa surgiram duas possíveis abordagens: a parametrização de código a partir da comunicação com o PLC e a geração de código.

A primeira alternativa facilita a troca de dados entre PLCs e computadores, sendo possível implementar funções de leitura e escrita de variáveis de e para o PLC. Neste contexto, a parametrização de código PLC poderia ser feita através de ferramentas como o Snap7 Library, recorrendo à utilização da sua interface com o Microsoft Visual Studio®. No entanto, esta abordagem envolveria o desenvolvimento de um programa global/completo de controlo do equipamento, de maneira a que, posteriormente, através do Snap7, fossem selecionados apenas trechos da lógica necessários de modo atingir o comportamento desejado para um cenário específico. Deste modo, independentemente da configuração final desejada, o código presente no programa permaneceria inalterado e incluiria todas as variações para o sistema, que viriam mais tarde a ser selecionadas de modo a utilizar apenas o código necessário para o controlo do cenário em causa. Isto resulta num programa mais pesado e de difícil compreensão. Por conseguinte, esta opção foi pouco explorada por Domingos (2018).

Na segunda abordagem, Domingos (2018) explorou diversas ferramentas para a problemática da geração de código, acabando por dar maior relevância ao TIA Portal Openness®. Isto porque, este software permite a comunicação direta com o TIA Portal® e a importação/exportação de documentos XML com informação relativa ao hardware e software presentes num programa. Após um estudo das funcionalidades do TIA Portal Openness®, nomeadamente através da análise dos exemplos fornecidos pela Siemens (também

mencionados ao pormenor na Secção 3.4.1.3 desta dissertação), Domingos (2018) concluiu ser possível alterar o código presente num programa do TIA Portal® através dos seus documentos XML.

A alteração do código via documentos XML foi conseguida através de uma aplicação externa, que mediante uma dada configuração gerava o programa TIA Portal® necessário ao controlo de um determinado cenário. A linguagem selecionada por Domingos (2018) para a aplicação foi #C, por se tratar de uma linguagem comum que permitia a utilização da interface do TIA Portal Openness®.

Para a criação da referida aplicação, em primeiro lugar, foi necessário desenvolver um programa base/geral de controlo no TIA Portal® para posterior manipulação/geração dos seus constituintes adaptá-lo a uma variação específica do sistema. Segundo Domingos (2018), este programa foi construído da forma mais modular possível, de modo a facilitar e minimizar a necessidade de alteração da lógica de controlo.

Em segundo lugar, foi necessário selecionar uma estratégia para a leitura e escrita de ficheiros XML. Dado que a *Siemens* fornecia os *Schemas* correspondentes à estrutura dos seus documentos XML e que o Microsoft Visual Studio® permitia a conversão automática dos mesmos em classes (facilitando o acesso aos campos dos elementos do XML), optou-se pela utilização da classe *XmlSerializer.* Esta estratégia possibilitou a criação e edição de documentos XML a partir da aplicação criada.

Fruto da modularidade do código desenvolvido e das funcionalidades do TIA Portal Openness®, concebeu-se uma aplicação capaz de gerar o programa necessário ao controlo do sistema, perante a seleção de um número relevante de configurações do mesmo.

A aplicação desenvolvida começa por ler um conjunto de documentos XML modelo, correspondentes aos diversos elementos de software a manipular. Posteriormente, através da interface gráfica, na qual o utilizador insere a configuração desejada, é recolhida a informação necessária à geração. Após despoletar este processo é gerado o software contido no programa TIA Portal®, quer ao nível de variáveis utilizadas e seus valores, identificações e endereçamento, quer ao nível do código incluído. Assim, geraram-se *organization blocks, function blocks, database blocks* e *tagtables* a partir da manipulação dos documentos XML correspondentes a cada um destes elementos de software. Uma vez produzidos os documentos XML necessários, a aplicação permitia a sua importação para um programa modelo previamente desenvolvido e contendo apenas elementos que não necessitavam de alterações. Obteve-se assim o programa necessário ao controlo de um *shuttle car*, num estado quase final, tendo-se concluído que era viável controlar sistemas com diferentes configurações a partir da aplicação desenvolvida (Domingos 2018).

2.4.2 Ferramentas existentes

Além da pesquisa académica, alguns fornecedores de sistemas de controlo também desenvolveram ferramentas automáticas de geração de código. Essas ferramentas visam gerar automaticamente o código do PLC com base num fluxo de trabalho predefinido que precisa de ser configurado pelo programador (Jbair et al. 2018). No entanto, tendo em conta que as ferramentas em questão são bastante recentes, não se encontrou nenhuma aplicação que refira a sua utilização, além de que a informação disponível acerca do seu funcionamento não está disponível em abundância. A sequência de passos bem como as funções das ferramentas em questão são falados nesta secção.

Siemens TIA Portal Openness®

Como já referido na secção anterior (Secção 2.4.1.2), a Siemens oferece uma API (*Application Programming Interface*) chamado TIA Portal Openness®, uma extensão incluída na instalação do software TIA Portal®, que permite a criação, gestão e/ou modificação de um projeto a partir de aplicações externas.

Esta interface permite que elementos de um projeto, como blocos e tabelas de variáveis sejam facilmente importados e exportados em formato elementos XML. Para além disso, para aplicações mais complexas, de forma a reduzir o trabalho redundante, é possível criar rotinas para criar projetos completos a partir de ferramentas como o Microsoft Visual Studio®, atuando como gerador de código (explicitado no capitulo "Recurso ao software Tia Portal Openness®" da Secção 2.4.2). No entanto, essa API não é fácil de utilizar e requer alto conhecimento de programação.

O TIA Portal® suporta também a importação direta de tabelas de variáveis, *functions*, *function blocks* e *organisation blocks* (OBs) dos arquivos de texto fornecido, quando o código está numa linguagem de texto especificado pela norma IEC 61131 (Garcia 2017; Siemens 2019).

CIF 3 PLC Code Generator®

A geração de código para PLC de acordo com a norma IEC 61131-3 também é possível com recurso ao gerador de código CIF 3 PLC Code Generator® (Beek et al. 2014). O CIF 3 é uma ferramenta académica para modelar sistemas híbridos que inclui geração de código PLC. Embora a ideia e a ferramenta disponível sejam promissoras, a linguagem de modelação de texto aplicada é demasiado complexa e matemática. O seu uso exigiria um treino extensivo, dificultando a sua aplicação (Darvas, Viñuela e Majzik 2016).

MATLAB Simulink PLC Coder®

O MATLAB Simulink PLC Coder parece ser uma das melhores e mais maduras ferramentas de geração de códigos de PLC disponíveis (Darvas, Viñuela e Majzik 2016). Esta aplicação permite gerar código de acordo com a norma IEC 61131-3, partindo de modelos *Simulink*, gráficos *Stateflow* e funções MATLAB. Assim, a aplicação pode gerar código para diferentes PLCs e importar o código gerado para a aplicação específica de programação de cada fornecedor (MathWorks 2020; Jbair et al. 2018).

No entanto, segundo Darvas, Viñuela e Majzik (2016), mesmo programadores com experiência podem ter problemas ao usar o *Stateflow* para modelar programas de PLC, uma vez que a utilização destes gráficos é muito suscetível a erros pois possui uma semântica que não se aplica ao domínio dos PLCs.

Ferramentas desenvolvidas por fornecedores de PLCs

Para além das aplicações referidas, outros fornecedores de PLCs também desenvolveram ferramentas automáticas de geração de código, como Mitsubishi Adroit Process Suite (MAPS), Schneider Unity Application Generator (UAG) ou interface de automação Beckhoff TwinCAT (Jbair et al. 2019).

Devido à escassez de informação acerca das ferramentas referidas, não foi possível aprofundar o conhecimento sobre cada uma delas. Conclui-se que todas visam gerar

automaticamente o código do PLC com base num fluxo de trabalho predefinido que precisa de ser configurado pelo programador do PLC; esse fluxo de trabalho geralmente é integrado à ferramenta.

3 Sistema, problema em causa e abordagem selecionada

A Secção 3.1 foca-se nos componentes e princípios de funcionamento dos armazéns automáticos da Consoveyo, S.A. bem como o funcionamento e o programa de controlo dos seus elevadores, que são o foco desta dissertação.

Na Secção 3.2 é descrito o problema em causa de forma mais concreta e enquadrada no contexto real. As abordagens referidas na secção 2.4 são estudadas em maior profundidade, sendo definida aquela que vai ser seguida na execução do presente projeto.

3.1 Descrição do sistema em causa

Nas Figura 10a) e b), encontra-se um exemplo da representação do *layout* de um armazém automático desenvolvido pela Consoveyo, S.A.. Este armazém é composto por 2 pisos, sendo a Figura 10a) o piso 1 do armazém e a Figura 10b) o segundo piso do mesmo armazém, onde os elevadores LV1 e LV2 são utilizados para fazer a transferência de paletes, que chegam dos *conveyors,* dos RGVs ou dos tapetes de correntes, de um dos dois pisos para o outro.

O sistema tem dois modos de funcionamento: automático, totalmente controlado pelo PLC, recolhendo e enviando as paletes para os seus destinos; manual, em caso de necessidade e com o auxílio de uma consola, para manutenção ou ajustes de posicionamento de cargas.

Enquanto a funcionar em modo automático, o elevador recebe as paletes que chegam dos *conveyors* localizados num dos lados da sua plataforma e transfere-as, ou para o outro lado da linha (não alterando a sua posição vertical) ou para outro piso. No caso da Figura 10, como se pode verificar pela observação dos sentidos das setas, os elevadores fazem a transferência da carga num sentido que é fixo e definido pelas restrições do mecanismo. Por outro lado, a Consoveyo, S.A., possui outros mecanismos, em que a plataforma do elevador pode efetuar movimentos em ambos os sentidos, sendo este apenas definido pela lista de ordens, dada pelo PLC, que o mesmo vai cumprir.

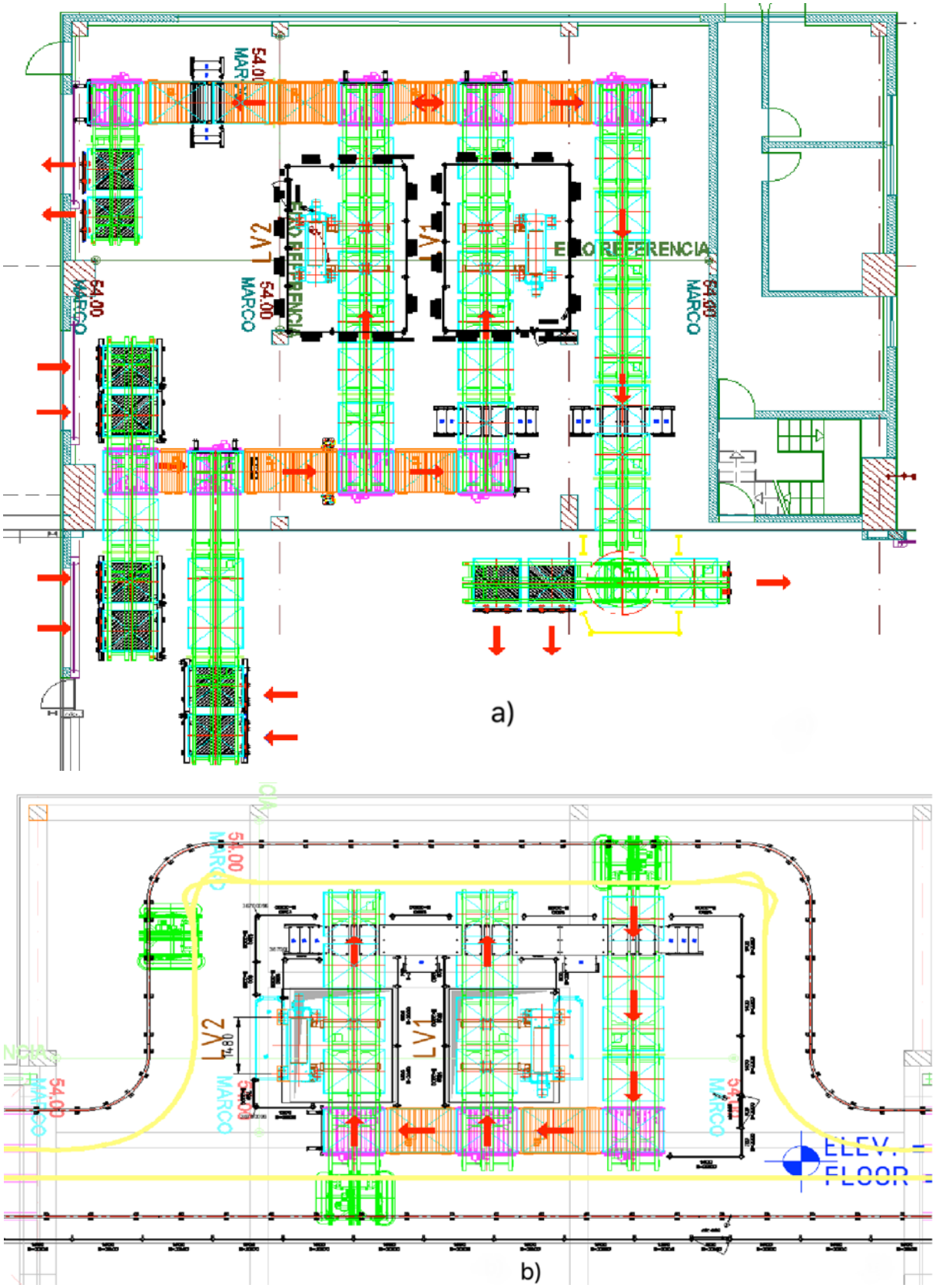


Figura 10 - Layout geral de um armazém automático. a) piso 1; b) piso 2

3.2 Princípio de funcionamento de um elevador Consoveyo, S.A.

Os elevadores, tal como descrito no Capítulo 2, têm como objetivo transferir cargas entre diferentes andares do sistema de transportadores. A carga é introduzida no sistema em paletes, onde circula através de *conveyors* e transportada pelo elevador até ao piso de destino pretendido.

3.2.1 Constituintes do elevador e hardware

O elevador é composto por variados elementos, que podem ser analisados na Figura 11 (Consoveyo 2020b).

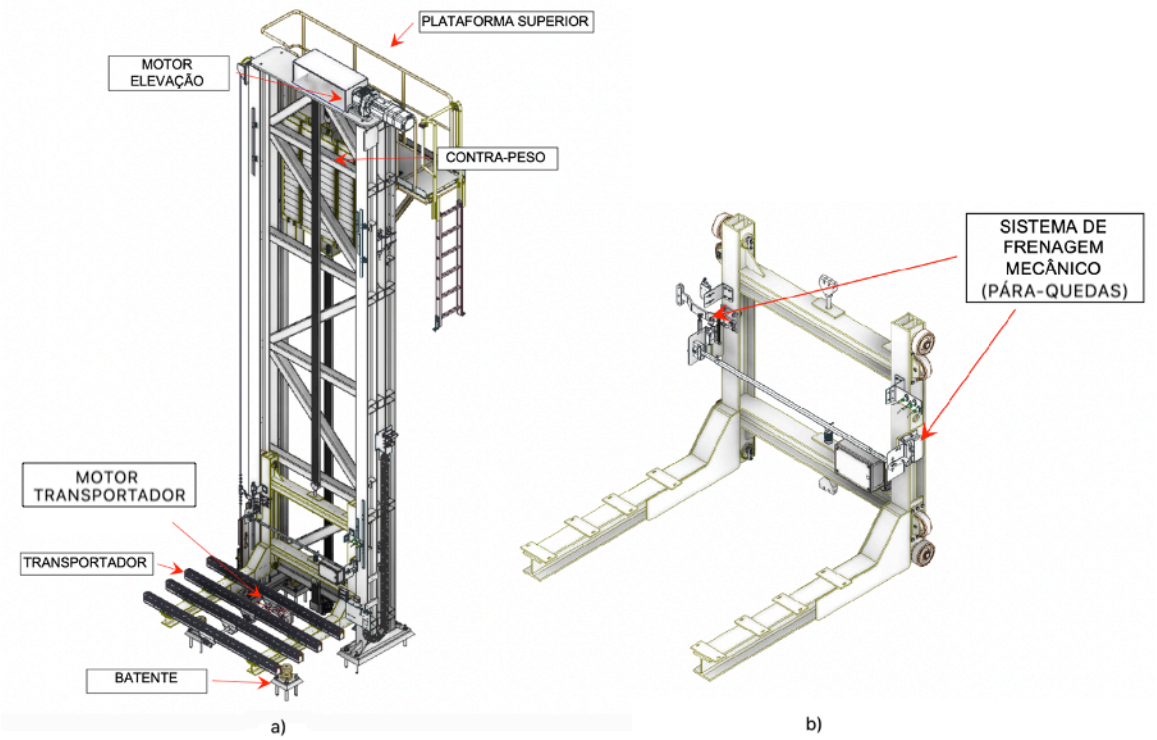


Figura 11 - Constituintes de um elevador produzido pela Consoveyo: a) elevador; b) plataforma. In Consoveyo, Manual_Operação_EL8x_PT_V003, 2020

O elevador é composto por um sistema de comunicação por laser, que se encontra posicionado na zona superior do mecanismo, que transmite os dados relativos à posição absoluta da plataforma ao variador de elevação, referidos na Secção 3.2.1, que por sua vez envia essa informação ao PLC.

3.2.1.1 Sistema de controlo - PLC

Os CPUs são escolhidos tendo em conta o preço, a velocidade de processamento, a memória, os periféricos e se têm ou não funções *safety*. Na Consoveyo, S.A. são utilizados equipamentos Siemens, marca com a qual estabelecem um acordo com preços especiais.

O módulo para controlo do elevador foi desenvolvido inicialmente para um autómato da gama 1512 F PN:

- (F) – tem um processador *safety;*
- (PN) – tem interface *profinet*, que foi a rede escolhida para comunicar com os restantes equipamentos do campo.

Para este autómato, adicionalmente, seria necessária uma ET200 (módulo de I/O remoto) para acomodar as cartas de sinais. Foi então que se optou por uma solução integrada CPU+ET200, cujo custo é menor do que ter os dois dispositivos separados, e está disponível no catálogo da Siemens. Assim, selecionou-se pelo que se chama uma ET200 com CPU.

O PLC é a unidade central de processamento do sistema, executando as tarefas mais importantes, sendo estas (Consoveyo 2020b):

- Supervisionar os reguladores de velocidade;
- Supervisionar o circuito da cadeia de emergência e deteção de falhas;
- Leitura de todos os sinais oriundos de fotocélulas e interruptores;
- Interface com o operador através da consola de operação e painel de controlo, botoneira e sinalizadores.

3.2.1.2 Variador de velocidade – SEW MOV-IC

Os variadores de velocidade usados para a elevação são da nova série MOVIDRIVE MDX91A, produzidos pela SEW – Eurodrive, e são implementados no quadro elétrico do sistema.

Ao contrário dos antigos mecanismos executados pela Consoveyo, em que a posição do elevador era medida através de um leitor e um código de barras acoplado ao longo do mastro e o variador recebia uma referência de velocidade, sendo o perfil de velocidades seguido definido pelo PLC, agora, com a nova geração de drives da SEW, o processo torna-se mais rápido e fácil: uma referência de posição é enviada para o variador, que controla o perfil de velocidades, com apoio de um laser instalado na parte superior da estrutura e um refletor na plataforma do elevador, consoante as referências que tem de velocidade máxima e as necessidades para alcançar o destino.

3.2.1.3 Dispositivo de medição da elevação - sensor laser

O dispositivo de medição laser instalado na zona superior do elevador, permite determinar a distância ao refletor do encoder (Figura 12), sem qualquer contacto físico e assim efetuar a avaliação do controlo do posicionamento do elevador (Consoveyo 2020b). O laser comunica com o variador através de uma comunicação série SSI (*Synchronous Serial Interface*).

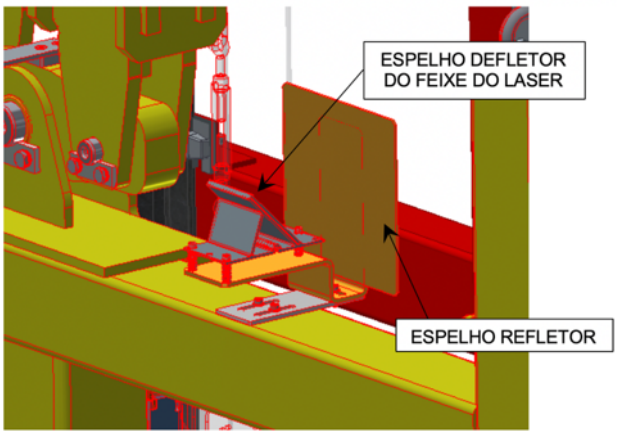


Figura 12 - Refletor do laser para posicionamento da plataforma

3.2.2 Modos de funcionamento

O sistema pode funcionar em dois modos distintos, selecionados a partir do seletor de chave no painel de acesso do elevador: modo **automático**, com as chaves na posição "remoto", modo normal de movimento das paletes no sistema; modo **manual**, com a chave na posição "local" usado caso seja necessário realizar manutenção, ou seja, necessário ajustar o posicionamento de alguma palete.

3.2.2.1 Modo automático

Quando selecionado o modo automático o elevador possui um gestor (implementado no PLC) que determina que tarefa irá realizar. Este gestor analisa a informação física (sensores) e lógica (ID palete, destino) e determina em cada momento o posto a ser atendido e o destino de cada palete que chega à plataforma.

No modo automático é possível verificar a seguinte alternância de opções:

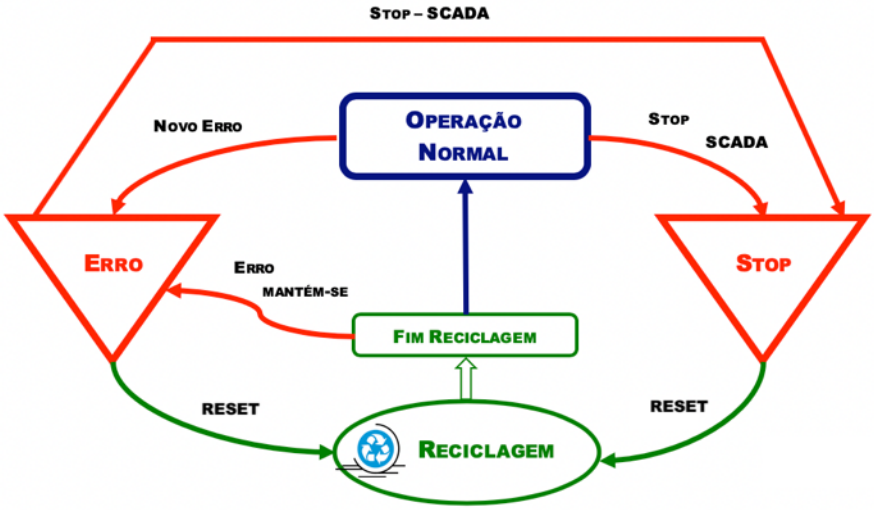


Figura 13 - Diagrama de estados possíveis em modo automático. In Consoveyo, Manual Operação Periféricos Paletes, 2020

Quando o sistema entra em erro, só a área afetada fica em erro, ficando as restantes em operação normal, permitindo 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. O estado de erro elimina-se pelo processo de reciclagem, que pretende apagar a informação de erro existente e verificar se o transportador se mantém em erro.

É possível colocar uma área em STOP, para paragem de sistema ou manutenção de informação lógica de paletes nos transportadores. O estado de STOP elimina-se premindo novamente no botão (Consoveyo 2020c).

A geração de uma ordem por parte do gestor é apenas possível se todas as condições, requeridas à carga e descarga de uma palete por parte do elevador, se reunirem, como a presença da carga no *conveyor* de entrada e caso o *conveyor* de destino esteja disponível para a receber. Dada a ordem, o elevador desloca-se para o piso correspondente à coordenada de carga disponibilizada pelo gestor, verifica o seu posicionamento e, caso se confirme o seu correto posicionamento, inicia-se a transferência da palete para a plataforma do elevador. Após o carregamento, o elevador desloca-se para o piso de descarga, repetindo o processo de verificação, após o qual se inicia a transferência da palete para o transportador de destino, até que esta seja entregue com sucesso.

3.2.2.2 Modo manual

Quando selecionado o modo manual o controlo é local, feito através da consola portátil de operação incorporada, sem necessidade de aceder à zona do elevador.

No modo manual, é ainda possível, através do seletor, ter dois modos de funcionamento distintos: com o seletor na posição "0", a atuação dos fins de curso ou a abertura de outros circuitos de emergência torna impossível movimentar os motores; com o seletor na posição "By-Pass", todas as emergências, como os interruptores de limite, deixam de ser consideradas na operação do equipamento, podendo-se, por exemplo, retirar o elevador dos limites da ala.

Por razões de segurança, a velocidade do elevador em modo manual é menor que a velocidade em modo automático.

3.2.3 Variantes possíveis

Como já referido no Capítulo 1, a Consoveyo, S.A. idealiza armazéns de acordo com os requisitos particulares de cada cliente, pelo que se torna difícil prever todas as variações possíveis no desenvolvimento dos mesmos. No que diz respeito aos elevadores, é possível enumerar um conjunto de variações mais comuns, descritas em seguida:

- Número de pisos;
- Número e posicionamento das entradas (linhas de carga) e saídas (linhas de descarga);
- Velocidades e acelerações;
- Tipo de *conveyor*: rolos/correntes;
- Número, tipo e posicionamento de sensores.

3.3 Definição do problema e abordagem

Tendo em conta o elevado número de possibilidades no desenvolvimento de um sistema com elevadores implementados, considera-se impossível reutilizar o mesmo programa de controlo para todos os projetos deste equipamento. Nesse contexto, a adaptação manual do programa à situação em causa seria necessária, o que seria uma tarefa não só trabalhosa e morosa como também suscetível a erros humanos, uma vez que nada indica que a pessoa que projetou o programa inicialmente seja a mesma que o vai alterar mais tarde em caso de necessidade.

Chegamos assim ao tema desta dissertação que, pela necessidade da adaptação automática do programa de controlo às variações impostas pelos clientes, propõe o desenvolvimento de uma aplicação software que permite gerar o programa de controlo do elevador e assim flexibilizar e agilizar a programação destes equipamentos.

Além de querer uma solução viável e com potencialidade, é importante ter em conta as restrições e o estilo de programação da Consoveyo, S.A.. A empresa, que emprega software da Siemens para a programação dos seus equipamentos, está de momento no processo de migração entre o STEP7 e o TIA Portal, utilizando, sempre que possível, a linguagem LD. Como tal, é importante que o programa de controlo gerado pela aplicação esteja de acordo com essas limitações.

3.4 Ferramentas exploradas

Depois de excluir algumas das opções pesquisadas no Capítulo 2.4 por não serem compatíveis com as ferramentas utilizadas na Consoveyo, S.A., nesta secção são exploradas em profundidade duas alterativas para criação da aplicação para geração de código, que se consideram mais viáveis. Para começar, será analisado o TIA Portal Openness®, que embora até ao momento seja uma ferramenta pouco explorada, é um software com grande potencialidade. Dentro dessa seleção, serão investigadas duas abordagens diferentes: a geração de código com recurso ao Microsoft Excel® e a manipulação de ficheiros XML.

3.4.1 TIA Portal Openness

O TIA Portal Openness® é uma interface de programação aberta (*Public* API - Figura 14) para o TIA Portal®, que permite interagir com este último usando uma aplicação customizada, a partir de ambientes de desenvolvimento externo, como o Microsoft Visual Studio® (Siemens 2018a).

Com o TIA Portal Openness® é possível automatizar tarefas de engenharia, como a programação, controlando-as com um programa desenvolvido pelo utilizador, permitindo não apenas simplificar significativamente o trabalho do programador, como também minimizar o número de erros inerentes à criação de um projeto (Siemens 2018b). Esta interface assenta na utilização de duas bibliotecas de vínculo dinâmico (*Dynamic-Link Library* - DLL) (Siemens.Engineering.dll e Siemens.Engineering.HMI.dll) que permitem a implementação de diversas funções de interface com o TIA Portal® (Siemens 2019).

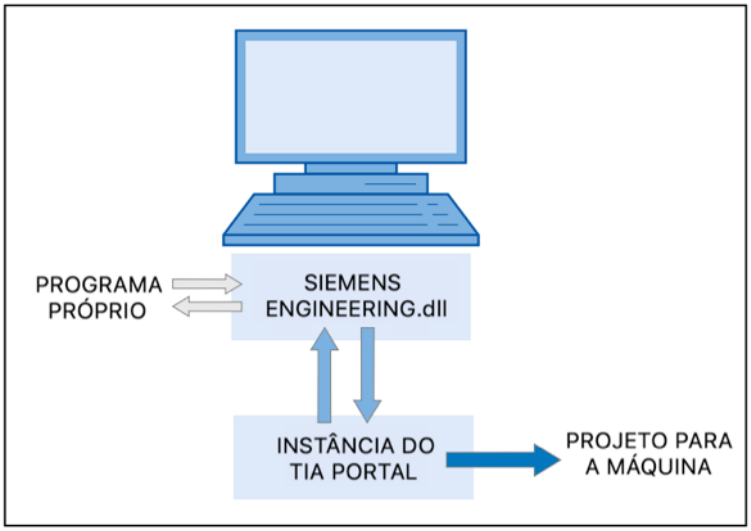


Figura 14 - Public API: Arquitetura. In Siemens, Openness: Automating creation of projects, 2018

Para a instalação do TIA Portal Openness®, bem como a inclusão dos ficheiros DLL na aplicação de desenvolvimento externa, recomenda-se a consulta dos procedimentos indicados no Capítulo 4 do manual "TIA Portal Openness: Introduction and Demo Application" (Siemens 2019).

Normalmente utilizado para criar um código modular, que pode ser usado por vários dispositivos, com mudanças específicas da aplicação feitas automaticamente, esta ferramenta oferece variadas vantagens que podem levar a uma forma eficiente de criação de código customizado. Definindo alguns templates base, é possível que o utilizador elabore projetos inteiros sem conhecimento de programação PLC - caso em que o código está escrito na totalidade, o utilizador apenas seleciona, da biblioteca, que frações do código vai querer utilizar;

não é gerado novo código, apenas organizado o que já havia sido criado por alguém antes (Smith 2019a).

3.4.1.1 Funcionalidades

O TIA Portal Openness® permite aceder a diversas áreas do TIA Portal® a partir da interface gráfica *Public* API, nomeadamente a dados de projeto, de PLCs e HMIs. Adicionalmente, esta ferramenta oferece a possibilidade de implementar funções para cumprir determinadas tarefas, designadamente (Siemens 2018a):

- Aceder ao TIA Portal® (quer por uma instância externa – com ou sem interface de utilizador – quer por acesso a processos a decorrer no momento);
- Fechar, abrir e guardar um projeto;
- Compilar alterações de hardware e/ou software;
- Conectar/desconectar o PLC;
- Importar/exportar informação, como dados do projeto, de HMIs (*tag tables, screens*), de PLCs (vários tipos de *program blocks, tag tables*) e de hardware, de projetos através de ficheiros XML;
- Criar, modificar, ler ou apagar informação de projetos.

Deste modo, segundo a Siemens (2018a), o TIA Portal Openness® permite a troca de informações e fornecimento de dados de projetos para aplicações externas, assim como o processamento, externo, de dados para novos projetos com base em configurações existentes e, posteriormente, a sua exportação.

Um projeto no Openness pode ser divido em duas partes (Smith 2019a):

- A aplicação em C# que dá a interface ao TIA Portal®;
- Os ficheiros XML que são usados para criar código no TIA Portal®.

A representação esquemática da Figura 15 mostra que componentes estão envolvidos na configuração máxima de uma solução exemplar (Siemens 2018b).

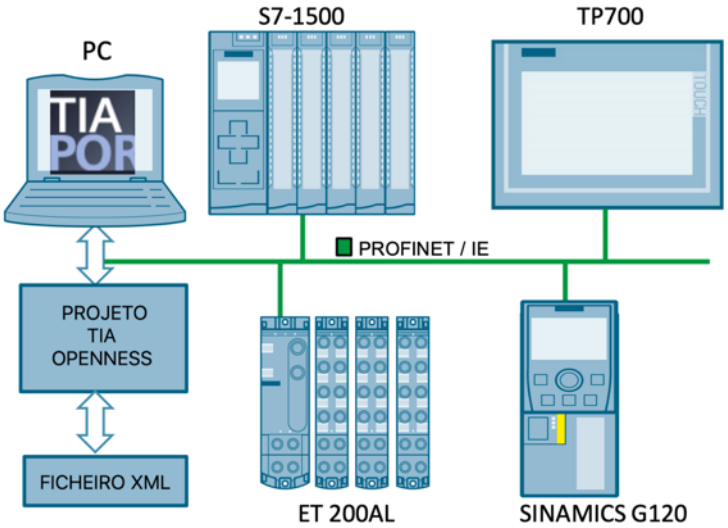


Figura 15 - Esquema de um exemplo de uma solução na máxima configuração

3.4.1.2 Variantes do TIA Portal Openness®

De acordo com a Siemens, o TIA Portal Openness® pode operar duas configurações distintas, ilustradas na Figura 16 (Siemens 2018a).

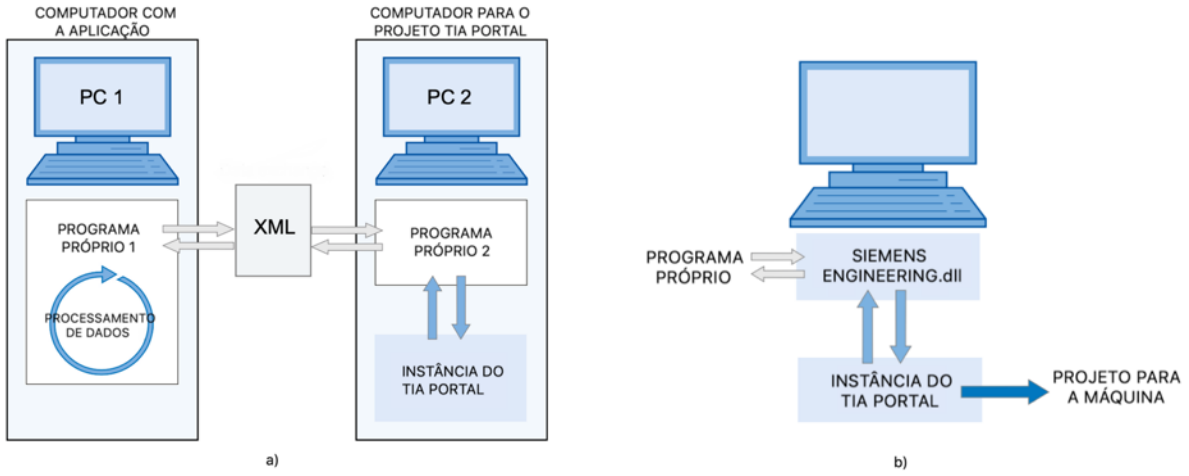


Figura 16 - Configurações TIA Portal: a) PCs distintos; b) PC único. In Siemens, Openness: Automating creation of projects, 2018

Aplicação e o TIA Portal® em computadores distintos

A aplicação e o software encontram-se em computadores distintos (Figura 16a)). A troca de dados é concebida a partir de ficheiros XML, que podem ser importados ou exportados pelas aplicações desenvolvidas para cada computador. A informação exportada do projeto de TIA Portal® do PC2 pode ser modificada no PC1 e reimportada de novo para o PC2 (Smith 2019a).

Aplicação e o TIA Portal® no mesmo computador

Nesta configuração (Figura 16b)), é utilizado apenas um computador no qual estão disponíveis tanto o TIA Portal® como a aplicação desenvolvida. Deste modo, o programa abre o TIA Portal® (com ou sem interface de utilizador), guarda e/ou fecha o projeto. É também possível fazer a conexão a um projeto já aberto. A partir daí podemos usar as funcionalidades do TIA Portal® para solicitar, gerar ou modificar informação do projeto ou para iniciar os processos de importação e exportação (Smith 2019a).

3.4.1.3 Exemplos de aplicação

No caso de uma situação representativa de engenharia mecânica modular, a Siemens apresenta uma outra configuração baseada na de apenas um computador, presumindo que o objetivo final é programar vários mecanismos semelhantes com variantes distintas. Para isto, deverá haver um projeto que contenha todos os componentes da máxima configuração do sistema, ou seja, todas as variantes possíveis da máquina. A aplicação desenvolvida deverá funcionar como gerador de código, ficando responsável por tarefas como obtenção dos parâmetros para uma máquina específica e controlo da criação do projeto a partir da separação e alteração dos elementos relevantes. De acordo com essas alterações, o ficheiro XML (os *program* blocks) criado previamente é alterado e importado no projeto (Siemens 2018b).

De facto, esta situação é bastante semelhante ao cenário em causa nesta dissertação. No entanto, a aplicação desta configuração implica a criação de um projeto global que inclua todo

o código que possa vir a ser utilizado. No caso da Consoveyo, S.A.., onde as soluções são personalizadas de acordo com o pedido do cliente, isto iria envolver o desenvolvimento de um projeto que abrangesse todas as variantes possíveis, o que, dado o elevado número das mesmas, seria uma tarefa bastante complexa e exigente.

Aplicações Demonstração

A Siemens disponibiliza duas aplicações de demonstração em C#, desenvolvidas no software Microsoft Visual Studio®: a *StartOpenness* e a *DemoOpenness*, interessantes de serem exploradas, uma vez que o objetivo da presente dissertação passa pelo desenvolvimento de uma aplicação que flexibilize a programação dos PLCs do elevador. Estas aplicações ilustram várias funções, já implementadas no programa, fornecendo uma visão geral das mesmas funções bem como ajuda detalhada, que guia o utilizador durante a programação da sua própria aplicação (Siemens 2019).

A partir da observação da interface fornecida pela aplicação *StartOpenness* (Figura 17) bem como do seu código, é possível identificar algumas funções úteis:

- Start TIA: inicia o TIA Portal® (opção com ou sem interface de utilizador);
- Dispose TIA: termina a conexão ao TIA Portal®
- Open Project: abre um projeto existente;
- Connect to open TIA Project: conecta-se a um projeto do TIA Portal® que esteja aberto no momento;
- Save/Close Project: guarda/fecha o projeto aberto;
- Add Device: adiciona um novo hardware;
- Compile: faz a compilação do dispositivo adicionado.

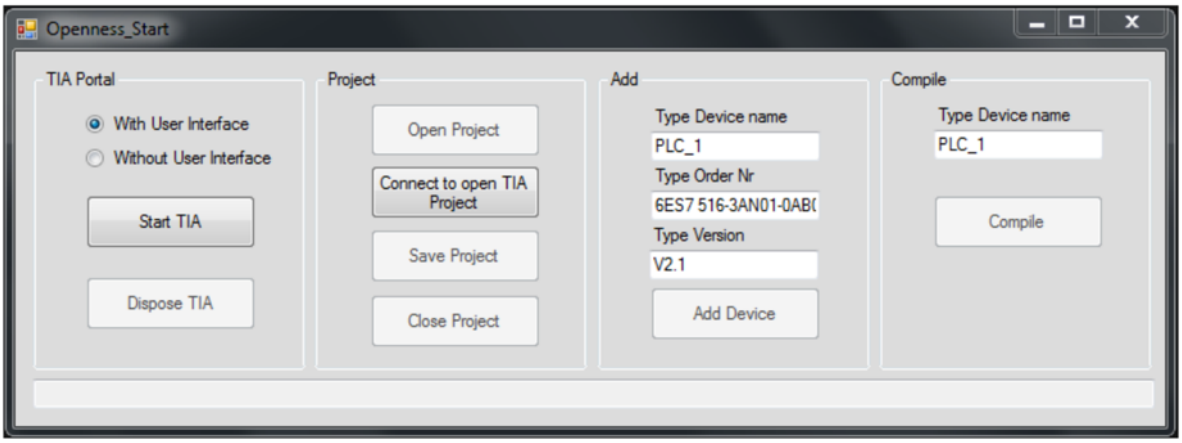


Figura 17 - Interface da aplicação StartOpenness. In Siemens, TIA Portal Openness: Introduction and Demo Application, 2019

A aplicação *DemoOpenness*, consiste em dois projetos: *TIAOpennessHelper*, com métodos recorrentes, prováveis de serem utilizados numa aplicação própria e o *TIAOpennessDemo*, que fornece uma visão geral das funções que podem ser implementadas com o TIA Portal Openness® (Siemens 2019).

3.4.2 Documentos XML e o TIA Portal®

Como visto anteriormente, o TIA Portal Openness® permite a importação e exportação de documentos XML com informação relativa a um projeto previamente criado do TIA Portal®. Significa que, no caso de abrirmos um ficheiro XML do TIA Portal Openness®, ele irá conter variadas tags definidas pela Siemens (Figura 18).

```
<Document> 1
  <SW.Blocks.FB ID="0"> 2
    <AttributeList> 3
      <AutoNumber>true</AutoNumber>
      <HeaderAuthor />
      <HeaderFamily />
      <HeaderName />
      <HeaderVersion>0.1</HeaderVersion>
      <Interface>
       4 <Sections xmlns="http://www.siemens.com/automation/Openness/SW/Interface/v2">
          <Section Name="Input">
           5 <Member Name="bInCondition" Datatype="Bool" Remanence="NonRetain" Accessibility="Public">
              <AttributeList>
                <BooleanAttribute Name="ExternalAccessible" SystemDefined="true">false</BooleanAttribute>
                <BooleanAttribute Name="ExternalVisible" SystemDefined="true">false</BooleanAttribute>
                <BooleanAttribute Name="ExternalWritable" SystemDefined="true">false</BooleanAttribute>
              </AttributeList>
              <Comment>
                <MultiLanguageText Lang="en-US">Input condition</MultiLanguageText>
              </Comment>
            </Member>
          </Section>
          <Section Name="Output">
            <Member Name="bOutDoSomething" Datatype="Bool" Remanence="NonRetain" Accessibility="Public">
              <AttributeList>
                <BooleanAttribute Name="ExternalAccessible" SystemDefined="true">false</BooleanAttribute>
                <BooleanAttribute Name="ExternalVisible" SystemDefined="true">false</BooleanAttribute>
                <BooleanAttribute Name="ExternalWritable" SystemDefined="true">false</BooleanAttribute>
              </AttributeList>
              <Comment>
                <MultiLanguageText Lang="en-US">Output action</MultiLanguageText>
              </Comment>
            </Member>
          </Section>
```

Figura 18 - Exemplo de um ficheiro XML de um *function block* do TIA Portal Openness

Segue-se a descrição de alguns desses elementos (Smith 2019b):

1. *Tag* "**Document**": Serve para delinear o início e o fim do documento XML. Todos os XML criados para o TIA Portal Openness® devem conter essa *tag*;
2. Tipo de **POU** (No código: SW.Blocks.FB ID="0"): Neste caso, "SW.Blocks.FB" diz ao programa que está a criar um *function block*. O ID="0" é um ID interno usado para identificar a informação dentro do XML. Se tivermos vários blocos dentro da *tag* "Document", cada bloco deverá ter um ID único. Em caso de erro, o TIA Portal Openness® irá usar esse número ID para especificar onde ocorreu o erro;
3. **Lista de atributos**: Cada bloco tem uma série de atributos, contidos dentro da *tag* "AttributeList", similares aos que vemos nas propriedades de um bloco no TIA Portal®; contém a informação geral do bloco (nome, número, linguagem de programação) e as variáveis declaradas no bloco – *Interface;*
4. **Secção**: Um dos diferentes tipos de informação que um *function block* pode conter. Por exemplo, podemos ter uma secção "Input" e outra "Output", que correspondem aos parâmetros de *input* e *output*. Para definir um parâmetro *input* temos que criar um "Member" na secção "Input".
5. ***Member***: Aqui podem-se observar os atributos: "Name", "Datatype", "Remanence" e "Accessibility". Por exemplo, os três atributos apresentados como "ExternalAccessible", "ExternalVisible" e "ExternalWritable" correspondem aos parâmetros que encontramos no TIA Portal® definidos como "Accessible from HMI", "Writable from HMI" e "Visible in HMI Engineering".

Como concluído no Capítulo 3.4.1.3, o desenvolvimento de um projeto que englobasse todas as variantes possíveis para um sistema da Consoveyo, S.A. resultaria num processo demasiado complexo e moroso. Como tal, avaliou-se a possibilidade de utilizar o TIA Portal Openness na geração de código para PLC, através da manipulação de um programa para PLCs a partir dos seus documentos XML.

De modo a proceder à avaliação referida, recorreu-se a um simples teste descrito nos pontos a seguir:

- Foi implementado, no TIA Portal®, um simples projeto, com apenas um *function block* que continha um pequeno código de controlo de um *conveyor*;
- Através da aplicação *OpennessDemo*, fornecida pela Siemens, um ficheiro XML relativo ao código desse FB foi exportado;
- Recorrendo ao Visual Studio®, alterou-se manualmente o nome de uma das variáveis, de maneira a que este apresentasse um código distinto, substituindo o ficheiro original;
- Este novo ficheiro XML manipulado foi então importado para o TIA Portal®, verificando-se que o código tinha sido alterado com sucesso, tendo o nome da variável alterada sido substituído por um distinto do original, provando-se assim a possibilidade de manipular projetos do TIA Portal® através da edição dos seus ficheiros XML.

3.4.2.1 Microsoft Excel e o TIA Portal®

O Microsoft Excel é uma ferramenta desenvolvida pela Microsoft que permite ler e possivelmente modificar partes do projeto, exportadas do TIA Portal®, e reimportá-las de volta. Além disso, o Excel possui a capacidade de programar macros (sequências automáticas de funções) usando a linguagem de programação VBA, automatizando procedimentos usados com frequência. Neste caso, a ideia seria desenvolver uma aplicação "escondida" no documento, como uma macro, e as folhas do Excel seriam consideradas a base de dados.

Como não é possível conectar diretamente o Microsoft Excel ao TIA Portal, uma vez que as tecnologias utilizadas não são compatíveis, será necessário recorrer ao TIA Portal Openness para efetuar essa ligação. Para tal, a Siemens desenvolveu duas bibliotecas DLL que permitem a geração de código no TIA Portal com base nos dados inseridos num documento Excel: a DLL "*TiaPortalOpennessXmlSupporter*", que cria os ficheiros XML do TIA Portal necessários para gerar *program blocks*, *data types*, *global data blocks* e *tag tables*; e a DLL "*TiaOpennessForExcel*", que compreende algumas funções do TIA Portal Openness e torna-as disponíveis para usar no Excel, sendo esta biblioteca responsável por aceder ao TIA Portal, assim como ao projeto, e por importar os ficheiros XML de volta ao software para gerar o código PLC (Figura 19).

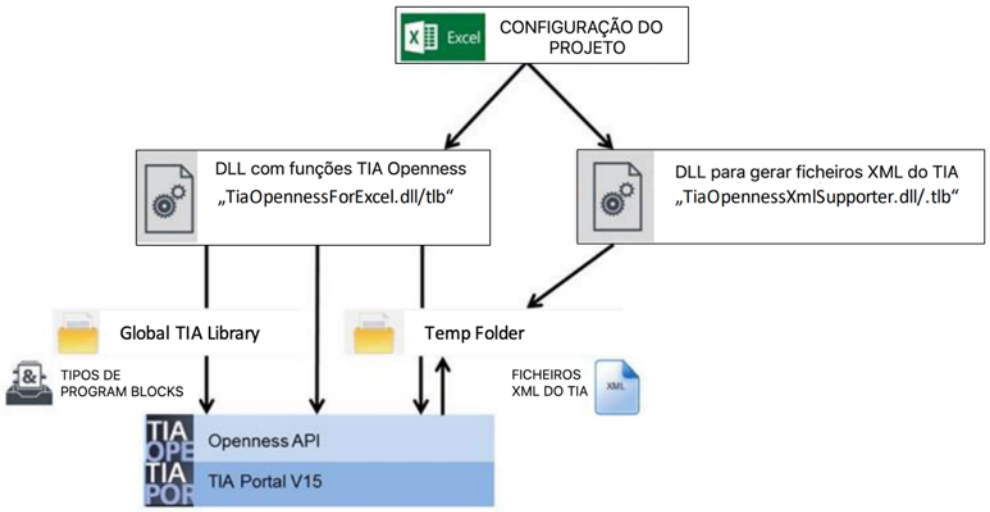


Figura 19 - Princípio de funcionamento de um gerador de código no Excel. In Siemens, Excel code generator for TIA Portal Openness, 2019.

O gerador de código é usado para criar automaticamente projetos que foram previamente configurados no Microsoft Excel. Um grande número de projetos semelhantes pode ser gerado, sendo útil, por exemplo, para máquinas em série com configurações diferentes.

Neste caso, a estrutura do projeto é criada na folha do Excel, e a lógica importada de uma galeria. Para isto, uma biblioteca com todos os *program blocks,* que contêm a lógica completa do programa, é necessária. Os blocos da biblioteca são interligados no gerador de código e posicionados na posição correta na hierarquia de chamadas.

Sendo esta uma estratégia bastante recente e pouco explorada, não foi possível testar a viabilidade do método num projeto com complexidade semelhante àquele que é tratado na presente dissertação. A partir da avaliação dos projetos, da bibliografia estudada, em que se utilizou o Microsoft Excel na geração de código para PLCs Siemens, conclui-se que esta é uma abordagem útil na importação de todo o tipo de variáveis, como *data blocks* e *tag tables*, sendo possível preencher estes campos do TIA Portal® através da importação de dados de uma tabela do Excel. Por outro lado, em relação à geração de FBs e OBs, a técnica já é mais limitadora. Este método requer que toda a lógica esteja previamente definida e permite apenas que esta seja importada e repetida ao longo do projeto, quantas vezes for necessária, tornando-se assim uma estratégia menos vantajosa que a utilização de uma interface no TIA Portal Openness, uma vez que esta segunda propõe o mesmo tipo de solução, sem a necessidade de conhecimento da linguagem VBA.

3.4.2.2 Manipulação de ficheiros XML a partir de uma aplicação em C# no Microsoft Visual Studio

Uma vez provada a possibilidade de manipular um programa para PLC a partir dos seus documentos XML, considera-se essencial proceder a um estudo acerca das formas de manipular documentos deste tipo atrás de uma aplicação de software em linguagem C#.

A leitura, escrita e edição de documentos XML a partir de uma aplicação nesta linguagem pode ser realizada de variadas formas. Apresentam-se em seguida algumas opções disponíveis para a manipulação de ficheiros deste tipo, ilustrando-se programas simples e o respetivo resultado, no que diz respeito aos ficheiros XML produzidos e, depois, editados.

XmlTextWriter (Figura 20): esta classe permite escrever ficheiros XML de acordo com as normas W3C; uma vez que o ficheiro não é retido em memória, é necessário escrever sempre o documento inteiro, do zero, e não é possível editar o mesmo.

```
XmlTextWriter xmlWriter = new XmlTextWriter                    //Cria o ficheiro XML
(@"C:\Users\crifer\Desktop\xmlWriter.xml", Encoding.UTF8);     //Caminho e nome do doc a criar

xmlWriter.Formatting = Formatting.Indented;                    //Formatação do ficheiro
xmlWriter.WriteStartElement("Modalidade");                     //Elemento <Modalidade>
xmlWriter.WriteStartElement("Formula1");                       //Elemento <Formula1>
xmlWriter.WriteAttributeString("Equipa", "Mercedes");          //Atributos <Equipa = Mercedes>
xmlWriter.WriteStartElement("Piloto");                         //Elemento <Piloto>
xmlWriter.WriteString("Hamilton");
xmlWriter.WriteEndElement();                                   //</Piloto>
xmlWriter.WriteEndElement();                                   //</Formula1>
xmlWriter.WriteEndElement();                                   //</Modalidade>
xmlWriter.Close();                                             //Fecha o Writer
```

"xmlWriter.xml"

```
<?xml version="1.0"?>
- <Modalidade>
  - <Formula1 Equipa="Mercedes">
      <Piloto>Hamilton</Piloto>
    </Formula1>
</Modalidade>
```

Figura 20 - Implementação e resultado do XmlTextWriter.

XmlTextReader (Figura 21): o XmlTextReader é uma alternativa, para a leitura de ficheiros XML, mais rápida e menos consumidora de memória, uma vez que não retém informação do ficheiro, não sendo, no entanto, possível fazer a sua edição.

```
XmlTextReader xmlReader = new XmlTextReader
(@"C:\Users\crifer\Desktop\xmlWriter.xml");          //Lê o ficheiro escrito antes
while (xmlReader.Read())                            //Lê enquanto houver elementos para ler
{
 if (xmlReader.Name == "Piloto")                    //Dentro do elemento Piloto
    {
        string Nome = xmlReader.ReadString();       //Lê o conteúdo
        Console.WriteLine(Nome);
        Console.ReadLine();                         //Escreve o que leu
    }
```

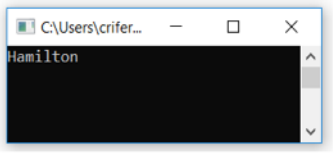


Figura 21 - Implementação e resultado do XmlTextReader.

XmlDocument (Figura 22): o XmlDocument é uma representação em memória de um documento XML, que permite ler, validar, manipular e posicionar porções de XML. Com recurso ao *XPath* é ainda possível navegar através dos elementos e atributos.

CONSTRUÇÃO DO DOCUMENTO

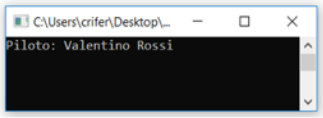
```
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.Load(@"C:\Users\crifer\Desktop\xmlWriter.xml");          //Leitura do documento "xmlWriter.xml"
XmlNode nodemotogp = xmlDoc.CreateElement("MotoGP");            //<MotoGP></MotoGP>
XmlNode nodepiloto = xmlDoc.CreateElement("Piloto");            //<Piloto></Piloto>
XmlAttribute atribequipa = xmlDoc.CreateAttribute("Equipa");    //Criação do atributo Equipa
atribequipa.Value = "Yamaha Factory Racing";                    //Atribuir um valor ao atributo
nodepiloto.InnerText = "Valentino Rossi";                       //Texto dentro do node Piloto
nodemotogp.Attributes.Append(atribequipa);                      //Atribuir atributo ao nó
nodemotogp.AppendChild(nodepiloto);                             //Nó piloto é filho do nó MotoGP
xmlDoc.DocumentElement.AppendChild(nodemotogp);                 //Nó MotoGP é filho do nó raiz

xmlDoc.Save(@"C:\Users\crifer\Desktop\xmlDocument.xml");        //Guardar ficheiro
```

"xmlDocument.xml"

```
<?xml version="1.0"?>
<Modalidade>
  - <Formula1 Equipa="Mercedes">
      <Piloto>Hamilton</Piloto>
    </Formula1>
  - <MotoGP Equipa="Yamaha Factory Racing">
      <Piloto>Valentino Rossi</Piloto>
    </MotoGP>
</Modalidade>
```

LEITURA DO DOCUMENTO

```
XmlDocument xmldoc2 = new XmlDocument();
xmldoc2.Load(@"C:\Users\crifer\Desktop\xmlDocument.xml");          //Lê o ficheiro
XmlNode node1 = xmldoc2.SelectSingleNode("Modalidade/MotoGP/Piloto");  //Seleção do nó
Console.WriteLine("Piloto: " + node1.InnerText);                   //Escrita
Console.ReadLine();
```



MANIPULAÇÃO DO DOCUMENTO

```
node1.InnerText = "Alteração: Maverick Vinales";                   //Alteração do node1
xmldoc2.Save(@"C:\Users\crifer\Desktop\xmlDocument_Alterado.xml");
```

"xmlDocument_Alterado"

```
<?xml version="1.0"?>
- <Modalidade>
  - <Formula1 Equipa="Mercedes">
      <Piloto>Hamilton</Piloto>
    </Formula1>
  - <MotoGP Equipa="Yamaha Factory Racing">
      <Piloto>Alteração: Maverick Vinales</Piloto>
    </MotoGP>
</Modalidade>
```

Figura 22 - Implementação e resultados do XmlDocument.

XmlSerialize: Serializa e desserializa objetos para e a partir de documentos XML, respetivamente, permitindo controlar a forma como os objetos são codificados no documento. Serialização é o processo de conversão de um objeto numa forma que pode ser facilmente transportada, sendo que, por outro lado, desserialização reconstrói o objeto. Os dados dos objetos são descritos usando construções de linguagem de programação como classes, campos e propriedades. Estas classes podem ser criadas pelo próprio programador (Figura 23) ou com recurso a uma ferramenta que gera classes baseadas num XML *Schema* previamente existente.

CONSTUÇÃO DO XML VIA XMLSERIALIZER

```
try
{
    Piloto piloto = new Piloto
    {
        Número = "33",
        NomePiloto = "Max Verstappen",
        NomeEquipa = "RedBull",
        preço = new Preço { Valor = 26, Unidade = "Milhões de Euros" },
        descrição = new Descrição { Títulos = "0", Vitorias = "9",
            Corridas = "105" }
    };
    //Serialização: info presente nas classes, preenche o XML
    XmlSerializer xmlSerializer = new XmlSerializer(typeof(Piloto));
    StreamWriter sw = new StreamWriter(@"C:\Users\crifer\Desktop
    \Tese_Cris\4. Geração de Código Automático\1. Microsoft Visual Studio\
    Serialize\Pilotos.xml");
    xmlSerializer.Serialize(sw, piloto);
    sw.Close();
    Console.WriteLine("Serialization's Successfull");
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

"Pilotos.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
- <piloto NomeEquipa="RedBull" Número="33"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <NomePiloto>Max Verstappen</NomePiloto>
    <Preço Unidade="Milhões de Euros">26</Preço>
  - <Descrição>
        <Títilos>0</Títilos>
        <Vitórias>9</Vitórias>
        <Corridas>105</Corridas>
    </Descrição>
</piloto>
```

```
//Lista de Pilotos
try
{
    List<Piloto> listPilotos = new List<Piloto>();
    listPilotos.Add(new Piloto{Número = "44",
        NomePiloto = "Lewis Hamilton",
        NomeEquipa = "Mercedes",
        preço = new Preço { Valor = 32, Unidade = "Milhões de Euros" },
        descrição = new Descrição
        {Títulos = "6", Vitorias = "88",Corridas = "255"} });
    listPilotos.Add(new Piloto{Número = "33",
        NomePiloto = "Valtteri Bottas",
        NomeEquipa = "Mercedes",
        preço = new Preço { Valor = 28, Unidade = "Milhões de Euros" },
        descrição = new Descrição
        { Títulos = "0", Vitorias = "8", Corridas = "142" } });
    XmlSerializer xmlSerializer = new XmlSerializer(typeof(List<Piloto>));
    StreamWriter sw = new StreamWriter(@"C:\Users\crifer\Desktop\Tese_Cris\
    4. Geração de Código Automático\1. Microsoft Visual Studio\Serialize\
    Lista_Pilotos.xml");
    xmlSerializer.Serialize(sw, listPilotos);
    sw.Close();
    Console.WriteLine("Serialization's Successfull");
```

"Lista_Pilotos.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
- <ArrayOfPiloto xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <Piloto NomeEquipa="Mercedes" Número="44">
        <NomePiloto>Lewis Hamilton</NomePiloto>
        <Preço Unidade="Milhões de Euros">32</Preço>
      - <Descrição>
            <Títilos>6</Títilos>
            <Vitórias>88</Vitórias>
            <Corridas>255</Corridas>
        </Descrição>
    </Piloto>
  - <Piloto NomeEquipa="Mercedes" Número="33">
        <NomePiloto>Valtteri Bottas</NomePiloto>
        <Preço Unidade="Milhões de Euros">28</Preço>
      - <Descrição>
            <Títilos>0</Títilos>
            <Vitórias>8</Vitórias>
            <Corridas>142</Corridas>
        </Descrição>
    </Piloto>
</ArrayOfPiloto>
```

LEITURA DO XML VIA XMLSERIALIZER

```
//Dessirialização: info do XML preenche as classes
try
{
    XmlSerializer xmlSerializer = new XmlSerializer(typeof(Piloto));
    StreamReader sr = new StreamReader(@"C:\Users\crifer\Desktop
    \Tese_Cris\4. Geração de Código Automático\1. Microsoft Visual Studio\
    Serialize\Pilotos.xml");
    Piloto piloto = (Piloto)xmlSerializer.Deserialize(sr);
    Console.WriteLine("INFORMAÇÃO DO PILOTO: ");
    Console.WriteLine("Nome: " + piloto.NomePiloto);
    Console.WriteLine("Nome Equipa: " + piloto.NomeEquipa);
    Console.WriteLine("Número: " + piloto.Número);
    Console.WriteLine("Preço: " + piloto.preço.Valor);
    Console.WriteLine("Unidade: " + piloto.preço.Unidade);
    Console.WriteLine("Títulos: " + piloto.descrição.Titulos);
    Console.WriteLine("Vitórias: " + piloto.descrição.Vitorias);
    Console.WriteLine("Corridas: " + piloto.descrição.Corridas);
}
```

```
C:\Users\crifer\source\rep...
Serialization's Successfull
INFORMAÇÃO DO PILOTO:
Nome: Max Verstappen
Nome Equipa: RedBull
Número: 33
Preço: 26
Unidade: Milhões de Euros
Títulos: 0
Vitórias: 9
Corridas: 105
```

EDIÇÃO DO XML VIA XMLSERIALIZER

```
piloto.NomePiloto = "Max Verstappen Alterado";
XmlSerializer xmlSerializer1 = new XmlSerializer(typeof(Piloto));
StreamWriter sw = new StreamWriter (@"C:\Users\crifer\Desktop\
Tese_Cris\4. Geração de Código Automático\1. Microsoft Visual Studio\
Serialize\Piloto_alterado.xml");
xmlSerializer1.Serialize(sw, piloto);
sw.Close();
```

"Piloto_alterado.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
- <piloto NomeEquipa="RedBull" Número="33"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <NomePiloto>Max Verstappen Alterado</NomePiloto>
    <Preço Unidade="Milhões de Euros">26</Preço>
  - <Descrição>
      <Títilos>0</Títilos>
      <Vitórias>9</Vitórias>
      <Corridas>105</Corridas>
    </Descrição>
</piloto>
```

Figura 23 - Implementação do XmlSerialize a parte das classes manuais.

Caso não se disponha do XML *Schema*, é também possível gerá-lo de forma automática através de um documento XML, utilizando o Visual Studio (Figura 24). Uma vez concebido o acesso ao XML *Schema*, a obtenção das classes automaticamente pode ser realizada através do *Developer Command Prompt*, ferramenta do Microsoft Visual Studio®, utilizando o comando ilustrado na Figura 25. Ao executar este comando uma classe é gerada (ficheiro .cs) com as classes necessárias para armazenar a informação dos objetos do documento XML.

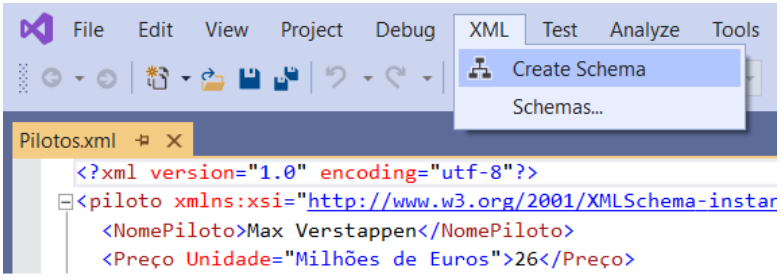


Figura 24 - Geração do XML *Schema* no Microsoft Visual Studio

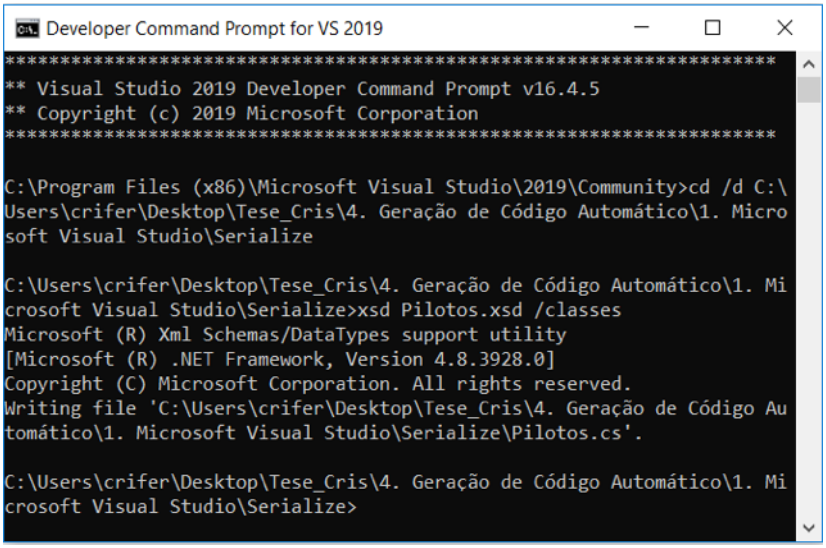


Figura 25 - Obtenção de classes através do *Developer Command Prompt*

Figura 26 - Classes obtidas: a) Automaticamente a partir do *Schema*; b) Manualmente pelo programador

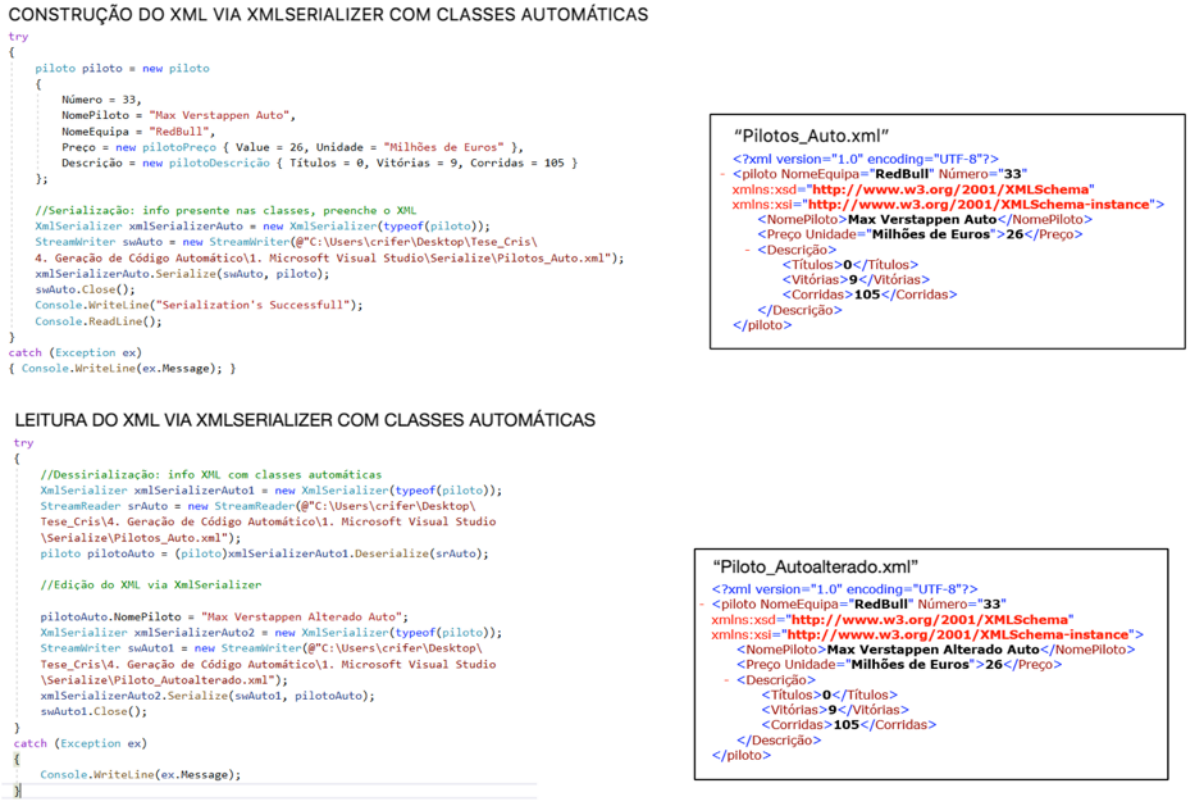


Figura 27 – Utilização do XmlSerialize a partir de classes automáticas

Como é possível constatar através da comparação entre as Figura 23 eFigura 27, os ficheiros XML gerados e manipulados a partir das classes criadas manualmente e das geradas automaticamente possuem uma estrutura bastante idêntica. Por outro lado, em alguns casos, as classes geradas automaticamente contêm erros, sendo esta uma desvantagem da sua utilização.

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

A dissertação mencionada na Secção 2.4.1.2, realizada em 2018, teve como objetivo o desenvolvimento de uma aplicação para a geração automática de código para o controlo de um *shuttle car*, que é outro dos componentes possíveis de equipamento igualmente produzido pela Consoveyo, S.A para os seus armazéns automáticos.

Um *shuttle car*, ou carro de transferência, é um veículo guiado por carris, utilizado para transferir carga entre as diferentes linhas de transportadores do sistema. Este trabalho distingue-se do propósito da presente dissertação uma vez que se pretendia controlar um equipamento cujo movimento é exclusivamente horizontal. Dada a relevância deste trabalho para a presente dissertação, será de seguida descrito detalhadamente os seus principais desenvolvimentos.

O trabalho centrou-se no desenvolvimento de uma aplicação em linguagem #C com a capacidade de gerar o código de controlo do *shuttle car*. Para tal, foi desenvolvido no TIA Portal® um programa de controlo para um cenário específico, dentro das possibilidades existentes. Na Figura 28 observam-se dois exemplos de *layouts* do Factory IO® para o equipamento em causa, concebidos por Domingos (2018) com o objetivo de definir o sistema a controlar e, posteriormente, testar o código desenvolvido no TIA Portal®.

Figura 28 - *Layouts* exemplares distintos no ambiente de simulação. In Henrique Domingos, Desenvolvimento de uma aplicação para geração automática de código para o PLC de controlo de um *shuttle car*, 2018.

3.5.1 Variantes do sistema a controlar

Tal como no caso dos elevadores, também os *shuttle cars* são projetados pela Consoveyo, S.A. de acordo com os requisitos do cliente. Deste modo, estes podem apresentar diversas variantes, sendo estas difíceis de prever e de juntar num mesmo programa. Como tal, expõem-se as variantes mais comuns e relevantes para o trabalho Domingos (2018):

- Número e posicionamento das linhas de carga/descarga;
- Número de *conveyors* por linha;

Face ao elevado número de variações passíveis de ser implementadas num equipamento deste tipo, torna-se impraticável utilizar o mesmo programa de controlo para os diferentes projetos. Deste modo, tal como nos elevadores, torna-se importante agilizar a programação destes equipamentos.

Uma vez exploradas ferramentas como o TIA Portal Openness® e o Sharp7, Domingos (2018) optou pela primeira para o desenvolvimento da aplicação de geração de código. O TIA

Portal Openness®, para além de possibilitar a comunicação com o TIA Portal®, através de uma interface, permite também importar e exportar documentos XML com informações relativas ao programa concebido no TIA Portal®.

3.5.2 Definição do sistema a controlar

O *layout* definido (Figura 29) por Domingos (2018) tem como componente principal o *shuttle car* propriamente dito, que se movimenta ao longo de um eixo horizontal ao nível do chão podendo parar nas posições nas quais se encontram as linhas de carga/descarga para recolher ou depositar carga.

As linhas de carga e descarga referidas anteriormente são compostas por diferentes números de *conveyors,* encontrando-se dispostas ao longo do curso do *shuttle car.* A carga, transportada em paletes, é inserida no sistema por *emitters*, alocados no início das linhas de carga, responsáveis pela admissão e orientação das paletes desde esse ponto inicial até ao local de carga. Já no final das linhas de descarga, encarregues de receber as paletes descarregadas pelo *shuttle car* e as encaminhar para o seu destino, estão colocados *removers*, que se responsabilizam da extração da carga do sistema. Ao longo das linhas de *conveyors* estão instalados diversos elementos sensores, que auxiliam no controlo dos mesmos (Domingos 2018).

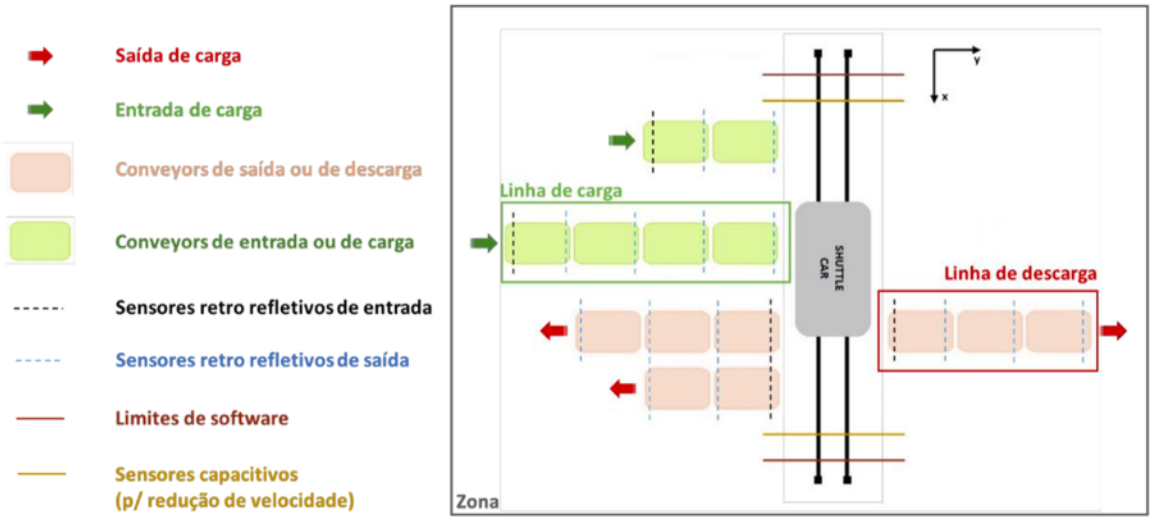


Figura 29 - Esquematização do layout desenvolvido. In Henrique Domingos, Desenvolvimento de uma aplicação para geração automática de código para o PLC de controlo de um *shuttle car*, 2018.

3.5.3 Programa de controlo do *shuttle car*

Com o intuito de obter um código que simplifique a complexidade do sistema e que seja passível de ser reutilizado, Domingos (2018) optou por desenvolver o programa no TIA Portal® de forma modular. Isto é, tipificando o problema, dividindo o sistema em vários subsistemas distintos e criando assim vários módulos capazes de os controlar. Neste caso, o programa foi concebido tendo em conta os seguintes pontos:

- Uma vez que o *shuttle car* tem dois modos de funcionamento distintos (automático e manual), o programa deverá conter dois módulos de código, correspondentes a cada um desses modos de funcionamento;

- Foram criados três módulos distintos para o controlo dos três tipos de *conveyors* (os primeiros, os intermédios e os últimos de cada linha), que podem ser reutilizados cada vez que se pretende adicionar uma nova linha de *conveyors*;
- Um módulo foi criado exclusivamente para a leitura organizada das ordens a executar pelo *shuttle car*;
- Existem também módulos para avaliar a presença de carga no local de origem e a disponibilidade do local de destino.

No caso destes dois últimos pontos, a informação obtida é transmitida ao módulo de funcionamento automático do *shuttle car.*

3.5.4 Implementação dos módulos no TIA Portal®

Os módulos definidos na Secção 3.5.3 foram implementados no TIA Portal® recorrendo à utilização de *function blocks.* Estes, como referido na Secção 2.2.1.1, possibilitam a reutilização do mesmo bloco de lógica em sistemas distintos, através de diferentes instanciações. Assim, Domingos (2018) incorporou a lógica necessária ao controlo de cada um dos módulos definidos previamente através dos seguintes elementos de software (Figura 30):

- FB_Shuttle_Manual e FB_Shuttle_Load_Unload: responsáveis pelo controlo do *shuttle car* nos modos de funcionamento manual e automático, respetivamente;
- FB_Conv_1, FB_Conv_i e FB_Conv_n: para cada um destes *function blocks*, cada instância permite controlar o primeiro *conveyor* de cada linha, um dos *conveyors* intermédios e o último *conveyor* da linha, respetivamente;
- FB_Reading_List e FB_Reading_Array: garantem a leitura e o cumprimento da lista de ordens de forma sequencial, a partir de uma lista de ordens, passando essa informação ao FB_Shuttle_Load_Unload;
- FB_Dest_Sensor: verifica o estado do sensor localizado no início da linha de descarga, de acordo com a coordenada da ordem a executar e transmite-o ao FB_Shuttle_Load_Unload;
- FB_Order_Sensor: verifica o estado do sensor localizado no início da linha de carga, de acordo com a coordenada da ordem a executar e transmite-o ao FB_Shuttle_Load_Unload;
- FB_Dest_Cnt: avalia o estado do contador do primeiro conveyor da linha de descarga, permitindo saber se o local de descarga está ou não livre para receber carga, e transmite-o ao FB_Shuttle_Load_Unload.

À parte destes elementos, foi ainda adicionado um outro *function block* para o controlo da simulação no Factory IO®, designado por FB_#Simulation. Este controla a colocação de paletes nas linhas de carga no ambiente de simulação, através da ativação dos *emitters* existentes, de acordo com as coordenadas lidas na ordem a executar (Domingos 2018).

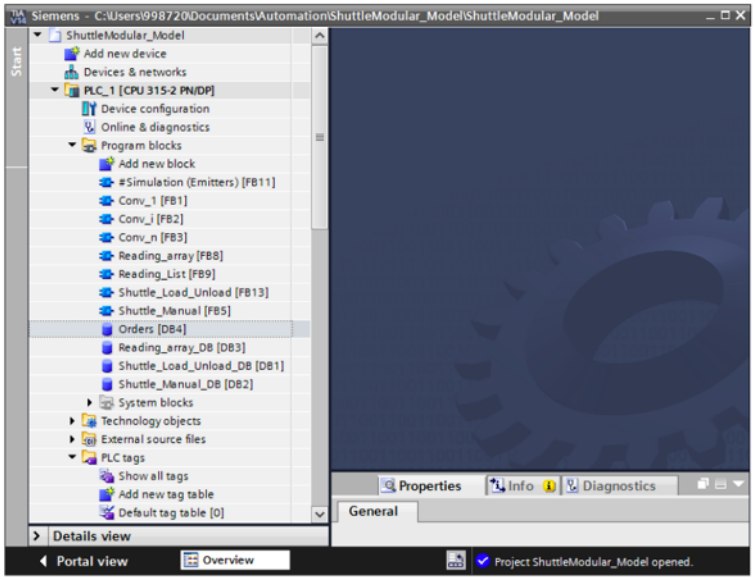


Figura 30 - Projeto modelo no TIA Portal. In Henrique Domingos, Desenvolvimento de uma aplicação para geração automática de código para o PLC de controlo de um *shuttle car*, 2018.

Para além dos *function blocks*, outros elementos de software foram empregues no programa de controlo do sistema (Domingos 2018):

- *Database Blocks* (DBs), utilizadas de modo a acomodar variáveis necessárias ao controlo do sistema; as DB_Global_Data e DB_Ordens, dois exemplos desse tipo de elemento, contém diversos parâmetros essenciais ao controlo do *shuttle car* (como tempos de carga/descarga e transição entre *conveyors*) e a lista de ordens a ser executadas, respetivamente;
- *TagTables,* criadas para acomodar os *inputs* e *outputs*;
- *Organization Blocks* (OBs), de maneira a instanciar os *function blocks* necessários ao controlo do sistema; de notar que, para cada instância são criadas *instance data blocks* contendo os valores das variáveis declaradas no FB instanciado. Além disso, o número de function blocks a instanciar no OB depende da configuração escolhida para o sistema a controlar.

3.5.5 Aplicação desenvolvida

Conforme referido anteriormente na Secção 2.4.1.2, a aplicação desenvolvida manipula os ficheiros XML do programa base/geral mediante as especificações do cenário e reimporta-os para o TIA Portal®, para um programa simples apenas com código estático. Deste modo, obtém-se um programa mais complexo, com a informação necessária ao controlo de uma variação do sistema (Domingos 2018).

A aplicação desenvolvida para a geração de código de controlo do PLC de um *shuttle car* segue o fluxo exposto em seguida (Figura 31):

1. A aplicação lê um conjunto de documentos XML modelo, respeitantes aos elementos de software que se pretende manipular;
2. O preenchimento de todos os parâmetros da interface gráfica, de acordo com o cenário escolhido, permite recolher toda a informação necessária ao preenchimento dos ficheiros XML modelo, passando estes a conter o código necessário ao controlo do sistema;

3. Assim que esses documentos XML preenchidos são produzidos, a sua importação é feita para o projeto modelo desenvolvido no TIA Portal®.
4. Após a importação dos ficheiros XML, obtém um programa de controlo do sistema, num estado quase final, necessitando apenas de algumas alterações ao nível das cartas I/O incorporadas no projeto.

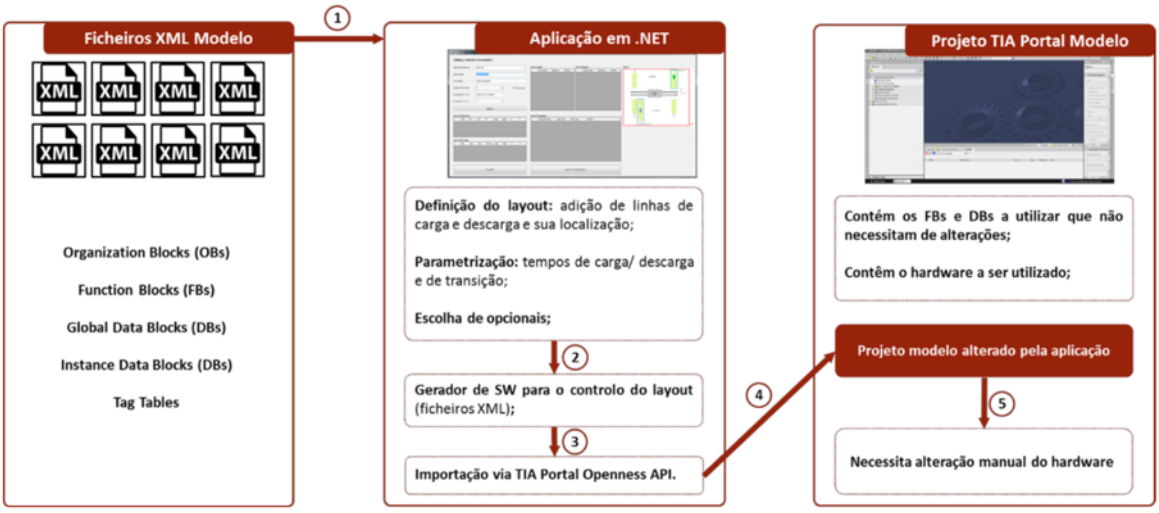


Figura 31 - Arquitetura da aplicação desenvolvida. In Henrique Domingos, Desenvolvimento de uma aplicação para geração automática de código para o PLC de controlo de um *shuttle car*, 2018.

3.5.5.1 Descrição da interface definida

A interface construída para a aplicação desenvolvida por Domingos (2018) permite definir a configuração do sistema pretendido através da introdução de informação relativamente a novas linhas de carga/descarga e adicionando essa linha ao sistema. Para adicionar uma nova linha é necessário definir:

- O tipo de linha (carga ou descarga);
- Identificação da nova linha (número da zona e da linha);
- Quantidade de *conveyors* que a linha deverá possuir;
- Posicionamento da linha (coordenadas X e Y).

3.5.5.2 Funcionamento da aplicação

Após a introdução de todos os parâmetros a preencher na interface, ao carregar no botão com a função de adição de uma nova linha, a informação relativa a essa linha é adicionada à aplicação, possibilitante a manipulação dos ficheiros XML modelo e, consequentemente, a geração dos ficheiros respeitantes ao controlo dessa linha.

Este processo de manipulação de documentos XML modelo resulta na alteração do conteúdo de diversos elementos de software do projeto desenvolvido no TIA Portal® de maneira a que este seja adaptado ao sistema a controlar. Na Figura 32 encontra-se um esquema que esclarece quais os elementos manipulados pela aplicação e quais aqueles que não necessitam de alterações (Domingos 2018).

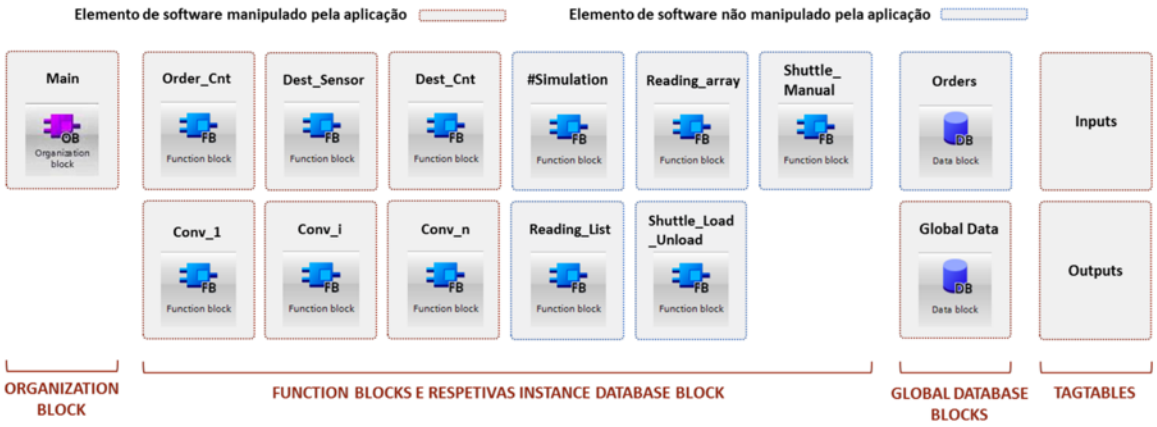


Figura 32 - Elementos de software do projeto desenvolvido manipulados pela aplicação. In Henrique Domingos, Desenvolvimento de uma aplicação para geração automática de código para o PLC de controlo de um *shuttle car*, 2018.

Para facilitar a compreensão dessas manipulações, explicam-se as implicações da adição de novas linhas de carga e descarga ao programa de controlo. Neste caso, a adição de uma nova linha de carga ao sistema implica:

- Alteração dos FBs Dest_Sensor e Dest_Cnt, uma vez que a adição de uma nova linha pressupõe o aparecimento de uma nova coordenada de destino e como tal, novos elementos cujos estados devem ser avaliados;
- Alteração das DBs correspondentes aos dois FBs alterados e instanciação de ambos na OB;
- Alteração das DBs correspondentes aos FBs que controlam os *conveyors* da nova linha (Conv_1, Conv_i e Conv_n) e instanciação dos seus FBs no OB;
- Adição de novos *inputs* e *outputs* que provêm da adição de novos *conveyors*.

A título de exemplo, passa-se a analisar com maior cuidado o FB_Dest_Sensor.

A Figura 33 representa o FB_Dest_Sensor, presente num programa de controlo de um cenário com três linhas de descarga posicionadas nas coordenadas (7,381;1), (7,381;-1) e (9,405;-1). Assim, como se pode observar, o *function block* é composto por três linhas, cada uma respeitante a uma linha de descarga distinta.

Compreende-se a necessidade de tal estrutura uma vez que, devido ao elevado número de possibilidades de colocação de novas linhas, não é possível prever e englobar todas as alternativas, à partida, num FB. Como tal, conclui-se que este é um exemplo de um elemento de software cujo estado inicial (no projeto modelo) se encontra vazio e será preenchido/alterado ao longo do processo de adição de novas linhas.

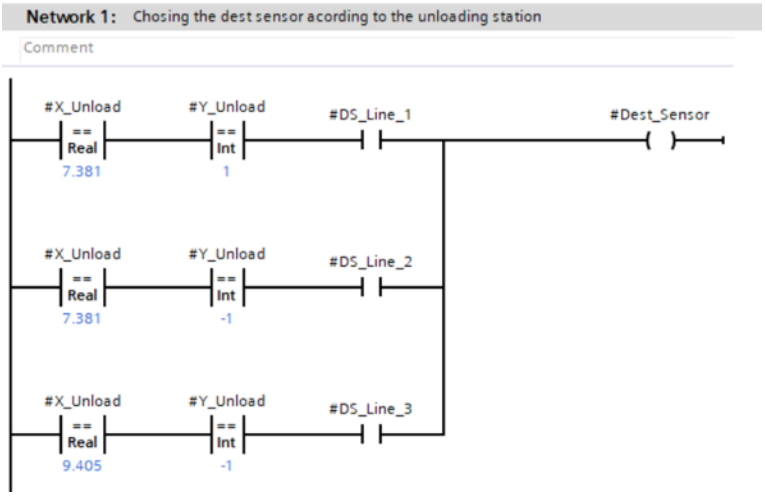


Figura 33 - FB_Dest_Sensor num programa de controlo de um cenário com três linhas de descarga.

A aplicação permite, a partir dos documentos XML produzidos, gerar o código contido no bloco FB_Dest_Sensor, permitindo que este seja adaptado a vários sistemas com configurações distintas.

Após a adição de todas as linhas pretendidas, as listas de informação necessária à manipulação dos ficheiros XML modelo estão completas. Como tal, o próximo passo será guardar esses novos ficheiros e importá-los para o projeto modelo, de forma a obter um projeto completo, capaz de controlar o cenário escolhido. Estas operações são realizadas através de métodos explorados na Secção 3.4.1.3.

3.6 Abordagem selecionada

Uma vez exploradas todas as ferramentas e abordagens mencionadas, confirmou-se que a utilização do TIA Portal Openness® permite implementar um leque de funcionalidades, incluindo a interface com o TIA Portal® (software de programação imposto pela empresa) e a importação/exportação de documentos XML. Como estudado na Secção 3.5 e com a execução de um simples teste, foi possível comprovar a possibilidade de alterar o código presente num programa TIA Portal®, através dos seus documentos XML. Como tal, o presente trabalho tem como objetivo a aplicação de métodos que possibilitam a manipulação deste tipo de documentos, a partir de uma aplicação externa, de acordo com uma dada configuração pretendida para um sistema a controlar.

Perante as vantagens inerentes à geração de código, optou-se por direcionar a presente dissertação para a abordagem da utilização do TIA Portal Openness®. O caso estudado na Secção 3.5 servirá de base ao projeto desenvolvido, sendo o presente trabalho uma expansão do trabalho realizado por Domingos (2018), com o foco na programação de um elevador.

Além disso, o tema deste documento tem por base o desenvolvimento do programa de controlo de um elevador protótipo para testes da Consoveyo, S.A., que estaria a ser desenvolvido em paralelo com a presente dissertação, não havendo ainda um programa de controlo finalizado para o mesmo. Além disso, tal como exposto na Secção 2.4.1.1, o desenvolvimento de uma ferramenta com capacidade de gerar e manipular código para PLC requer total entendimento do programa e do sistema a controlar, o que, partindo de um programa desenvolvido por terceiros, seria uma tarefa demasiado complexa e morosa perante o curto prazo da dissertação. Por estas duas razões, optou-se por desenvolver um novo programa de controlo em diagrama de contactos, utilizando o TIA Portal®. Por sua vez, para a simulação

foi utilizado o FactoryIO® de modo a testar o comportamento do sistema de acordo com o programa desenvolvido.

O novo programa de controlo desenvolvido tenta ao máximo replicar todos os princípios de funcionamento e variações de um verdadeiro elevador da Consoveyo, S.A., referidos no Capítulo 3.2.

Porém, tendo em conta as limitações do simulador em questão, o programa desenvolvido e, consequentemente, as variações a considerar na implementação do elevador ficaram condicionadas, sendo necessário redefinir ligeiramente o sistema a controlar. Por exemplo, os sensores implementados no protótipo do elevador, como referido na Secção 3.2.1.3, serão sensores laser, sendo a posição absoluta do elevador conhecida a qualquer instante. Já no simulador, devido à impossibilidade de instalação de sensores desse tipo, a posição do elevador não é continuamente conhecida e a paragem do mesmo nos pisos é feita através de vários sensores indutivos localizados estrategicamente ao longo da haste.

Assim, tiveram-se em conta algumas das variações implementadas no elevador da Consoveyo, S.A. referidas na Secção 3.2.3, nomeadamente:

- Número de pisos;
- Número e posicionamento das entradas (linhas de carga) e das saídas (linhas de descarga);
- Velocidades e acelerações das subidas e descidas do elevador.

Além disso, algumas variações opcionais foram acrescentadas, tais como:

- Indicação de sinalizadores indicativos da disponibilidade das linhas e do movimento de elevação.

4 Programa de controlo e ambiente de simulação desenvolvidos

Este capítulo foca-se na definição do sistema a controlar, desenvolvido no software de simulação Factory IO®, bem como a descrição do seu comportamento baseado no programa de controlo desenvolvido em TIA Portal®, que será mais a frente apresentado e analisado.

4.1 Ambiente de simulação

Como referido na Secção 3.6, a presente dissertação foi desenvolvida em paralelo com a elaboração do sistema de elevação em questão. Devido ao atraso no projeto deste sistema, optou-se por desenvolver um cenário em FactoryIO® que servisse simultaneamente de definição do sistema a controlar e ambiente de simulação para teste do programa de controlo do TIA Portal®. A construção deste ambiente de simulação permite não só a visualização em tempo real do seu comportamento, mas também a possibilidade de criar diferentes cenários tendo em conta as variações implementadas no sistema real, o que facilita o desenvolvimento da aplicação de geração de código.

4.1.1 Sistema a controlar

Para a definição do sistema a controlar foi desenvolvido um *layout*-tipo no Factory IO® à semelhança dos sistemas produzidos pela Consoveyo, S.A., que pode ser observado na Figura 34, descrito seguidamente.

O *layout*-tipo é constituído por um elevador de cargas, previamente agrupado, disponibilizado pelo programa, composto por uma plataforma, diversos sensores e um sinalizador luminoso, com possibilidade de efetuar paragem em três pisos distintos (piso do chão incluído).

Além disso, inclui-se também no *layout* linhas de carga e descarga, composta por dois *conveyors*, dispostas de ambos os lados ao longo dos pisos do elevador, necessários à simulação da chegada e saída de carga da plataforma, de maneira a imitar o funcionamento real do sistema. A carga, transportada em paletes, é introduzida no sistema a partir de *emitters* (identificados na Figura 35 por setas verdes) e extraída por *removers* (identificados na Figura 35 por setas vermelhas). As linhas de carga são responsáveis pela receção e encaminhamento das paletes desde a entrada da linha até ao ponto de carga. Por sua vez, as linhas de descarga recebem as paletes deixadas pelo elevador e encaminham-nas até ao seu destino final. É importante notar que, apesar de neste cenário se ter definido as entradas de carga como as linhas da esquerda e as saídas como sendo as linhas da direita, numa situação real esta escolha é apresentada como uma das variantes do sistema, não sendo obrigatória que as entradas e saídas de carga sejam organizadas desta forma. No início e no fim de cada *conveyor* foram instalados sensores

refletivos (Figura 35), que detetam a presença, ou não, de carga no tapete, e controlam o funcionamento dos mesmos.

Figura 34 - *Layout* desenvolvido no Factory IO

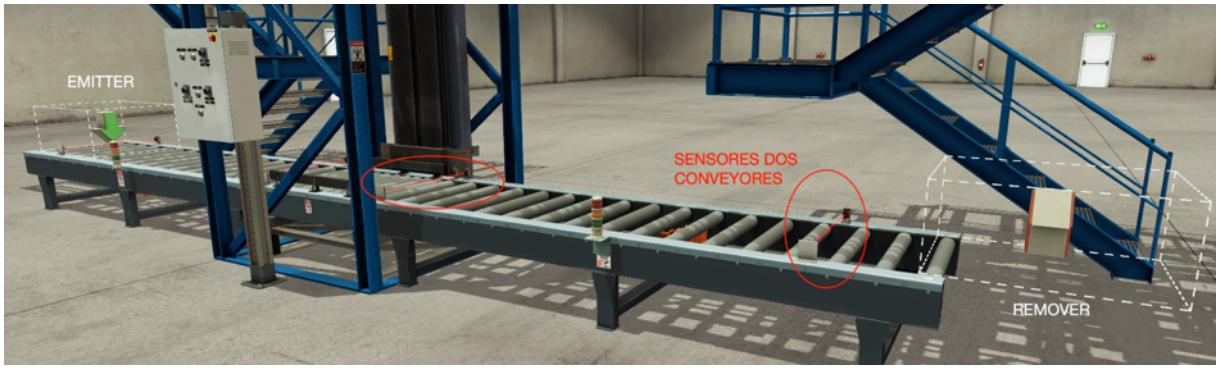


Figura 35 - Pormenor dos sensores dos *conveyors*, *emitters* e *removers*

Na Figura 36 é possível ver ao pormenor os sensores que controlam a elevação da plataforma de carga do elevador. Estes sensores estão localizados de maneira a controlar não só o seu movimento (é através da ativação destes sensores que se passa ao elevador a informação sobre em que piso deverá parar para receber ou deixar carga) mas também a velocidade da subida/descida. O elevador deverá abrandar para uma velocidade menor assim que o primeiro sensor seja intercetado (o de cimo no caso de descida ou o de baixo no caso da subida) e a paragem total deverá ser feita assim que ambos os sensores do piso de destino estejam ativos.

Para além da sinalização luminosa incluída no elevador, que anuncia o movimento de elevação da plataforma, outros sinalizadores foram adicionados, como se pode observar nas Figura 34 eFigura 35, nos *conveyors*, com o objetivo de indicar a disponibilidade de cada linha para receber carga.

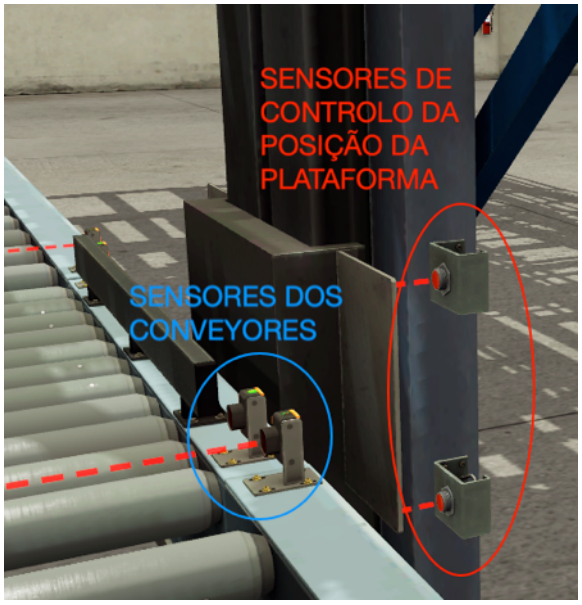


Figura 36 – Pormenor elementos sensores

4.1.2 Funcionamento do sistema a controlar

Assim como no sistema implementado na Consoveyo, S.A., referido no Capítulo 3.2, também o sistema desenvolvido no simulador possui dois modos distintos de funcionamento: modo automático e modo manual.

Modo Automático

No modo automático as deslocações dos *conveyors* e da plataforma de elevação do elevador são totalmente controlados pelo programa de controlo.

O elevador, sempre que se encontra pronto, recebe uma nova ordem de transporte. Isto implica que o elevador esteja parado, sem nenhuma operação de carga ou descarga em curso e se encontra dentro dos limites de segurança implementados. As ordens são cumpridas por ordem de chegada, não tendo sido implementada qualquer otimização que reduzisse as distâncias percorridas pelo elevador.

Após o recebimento de uma ordem e antes da sua execução, algumas condições são verificadas:

- Existência de uma carga a transportar no local de origem, a partir da análise do estado do sensor localizado no final da linha correspondente à coordenada do piso de entrada, imediatamente antes da plataforma;
- Disponibilidade do destino, a partir do estado do sensor colocado no início da linha correspondente à coordenada do piso de destino da ordem recebida;

Enquanto estas condições não se reunirem, a ordem recebida não será executada e o elevador posicionar-se-á na posição de *standby* após alguns segundos.

Modo manual

Quando colocado em modo manual, o elevador é controlado através dos comandos colocados no quadro elétrico (que representa o que na realidade seria a consola HMI) (Figura

34 e Figura 37). Este modo é ativado premindo a botoneira "*Reset*", seguindo-se a seleção do modo "Manual" a partir do seletor "Auto-Manual".

A velocidade é escolhida de acordo com a posição da botoneira "Velocidade Pequena-Grande", estando duas velocidades distintas disponíveis. Assim, se o seletor estiver posicionado na posição "Velocidade Pequena", o movimento é efetuado a partir das botoneiras "Descida" e "Subida", movendo-se para baixo e para cima, respetivamente, a uma velocidade baixa. O mesmo acontece no caso da posição "Velocidade Grande" estar selecionada, movimentando-se o elevador para baixo ou para cima a uma velocidade superior. Se nenhuma das botoneiras "Subida"/"Descida" estiver premida, o elevador permanece parado.

Neste modo existe ainda a possibilidade de o elevador ter dois comportamentos distintos: com o seletor "0-ByPass" na posição "0" é apenas permitido movimentar o elevador dentro dos limites de software; com o seletor na posição "By-Pass" é possível ultrapassar esses limites.

Figura 37 - Quadro elétrico do ambiente de simulação

4.2 Programa de controlo do elevador

Como referido na Secção 3.6, decidiu-se desenvolver um novo programa de controlo do elevador em TIA Portal®, escrito totalmente em linguagem LD. Este novo programa foi desenvolvido tendo em conta os requisitos de modularidade necessária à posterior geração de código e abrangendo as variantes referidas na Secção 3.6, de modo a que este possa ser aplicado em múltiplas configurações do mesmo sistema.

Esta alternativa passa por dividir o sistema nos seus diversos componentes, em que cada componente é programado separadamente e, depois de juntos, resulta uma sequência principal que controla todos os componentes juntos.

4.2.1 Modularidade do programa

De maneira a resolver o problema tratado na presente dissertação, considera-se muito importante que o programa desenvolvido para o controlo do sistema seja um programa modular, uma vez que se pretende que este possa ser utilizado de forma a controlar múltiplas variações de um tipo de sistema, e não um sistema em particular. Como tal, a modularidade do programa permite não só simplificar a estrutura de programação em variados subsistemas mais simples, mas também a criação de módulos que podem ser reutilizados ao longo do código.

Assim, de maneira a criar um código "tipo" que possa responder às distintas variantes do sistema referidas na Secção 3.6, dividiu-se o sistema exemplar apresentado na Secção 4.1.1 em vários subsistemas distintos. O raciocínio envolvido na partição dos vários módulos que constituem o programa é explicitado a seguir (Figura 38):

- O programa deve conter dois módulos de código, que dizem respeito ao modo de funcionamento do elevador: automático ou manual;
- Devem existir blocos de código que controlam os *conveyors* que levam a carga até à plataforma do elevador, de acordo com o seu tipo (*conveyor* de entrada ou saída);
- Módulos de código devem também ser criados de maneira a avaliar o estado dos pisos de carga e descarga (se o piso está disponível e se existe carga a ser transportada, respetivamente) da ordem a ser executada. Esta informação deve ser transmitida ao módulo de funcionamento automático do elevador;
- O programa deve incluir um bloco que permita a leitura das ordens a executar pelo elevador, a partir de uma listagem. A ordem lida com base nessa listagem, deverá ser transmitida ao módulo de funcionamento automático do elevador;
- De acordo com o número de pisos do elevador, blocos que controlam a elevação da plataforma do elevador deverão ser incluídos, de maneira a decidir, partindo da sua posição atual, se o elevador deverá subir ou descer, para cumprir a ordem recebida. Esta informação deverá também ser passada ao módulo de funcionamento automático do sistema.

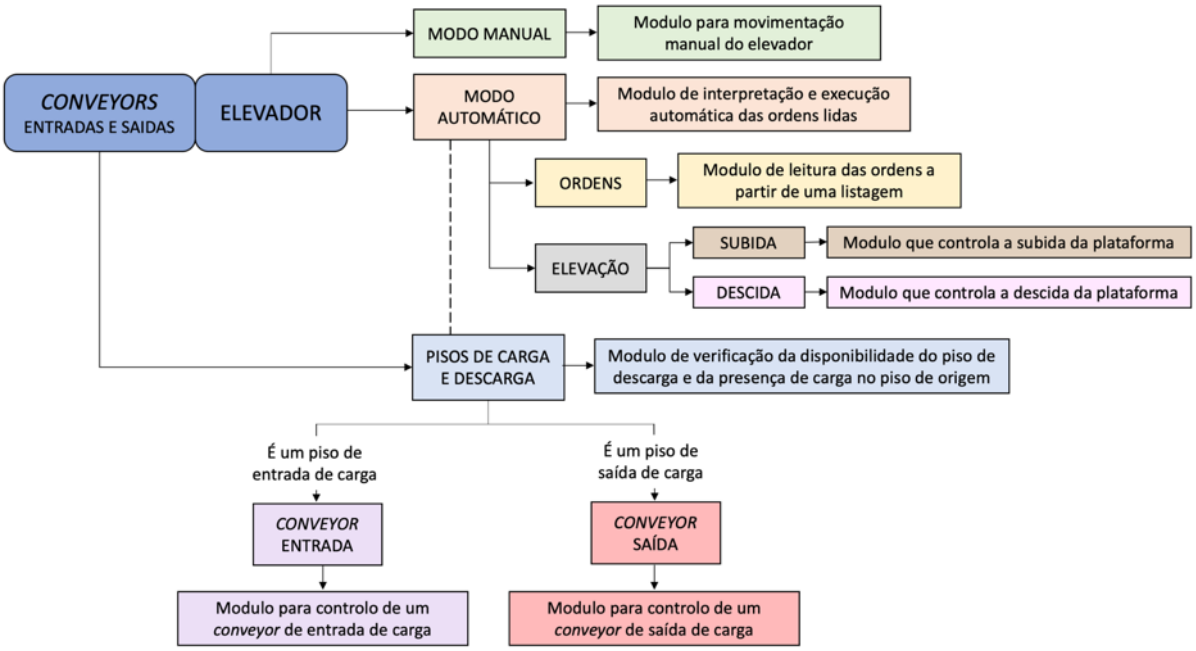


Figura 38 - Esquema da divisão modular do sistema em causa

4.2.2 Programa desenvolvido no TIA Portal®

Uma vez definidos os módulos necessários ao controlo do sistema (Secção 4.2.1), é necessária a sua implementação no TIA Portal®. Esse procedimento é realizado através da utilização de *function blocks* (FBs), dado que, como referida na secção 2.2.1.1, a utilização deste tipo de POU permite a reutilização da mesma lógica em sistemas distintos, através de diferentes instanciações.

Para além disso, ao longo do programa foram utilizados outros elementos de software, entre os quais *organization blocks* (OBs), *database blocks* (DBs) e *tagtables*. Uma descrição breve de todos esses elementos de software, integrados no programa desenvolvido no TIA Portal®, é efetuada a seguir.

Function Blocks

Os *function blocks* desenvolvidos no programa do TIA Portal® correspondem aos módulos referidos na Secção 4.2.1. São então aqui exploradas as funções atribuídas a cada um deles, ilustrando-se um GRAFCET concebidos com vista à sua programação. Os GRAFCETS não apresentados nesta secção, poderão ser encontrados no Anexo A.

FB_Elevador_Manual: Controla o funcionamento do elevador, quando colocado em modo manual;

FB_Elevação_Auto: Controla o funcionamento do elevador, em modo automático. Este *function block* controla a execução das ordens de transporte. Como tal, assim que uma nova ordem é recebida (fornecida pelo FB_Reading_Array, descrito posteriormente), é verificada:

- A presença de carga a ser transportada, a partir do estado do sensor existente no final da linha correspondente ao piso de origem, presente na ordem recebida (fornecido pelo bloco Sensor_Origem);

- A disponibilidade do *conveyor* de destino, a partir do sensor colocado no início da linha correspondente ao piso de destino presente na ordem recebida (fornecido pelo bloco Sensor_Destino).

Caso ambas as condições se verifiquem e o elevador esteja pronto, a ordem é executada. O GRAFCET deste *function block* é apresentado na Figura 39.

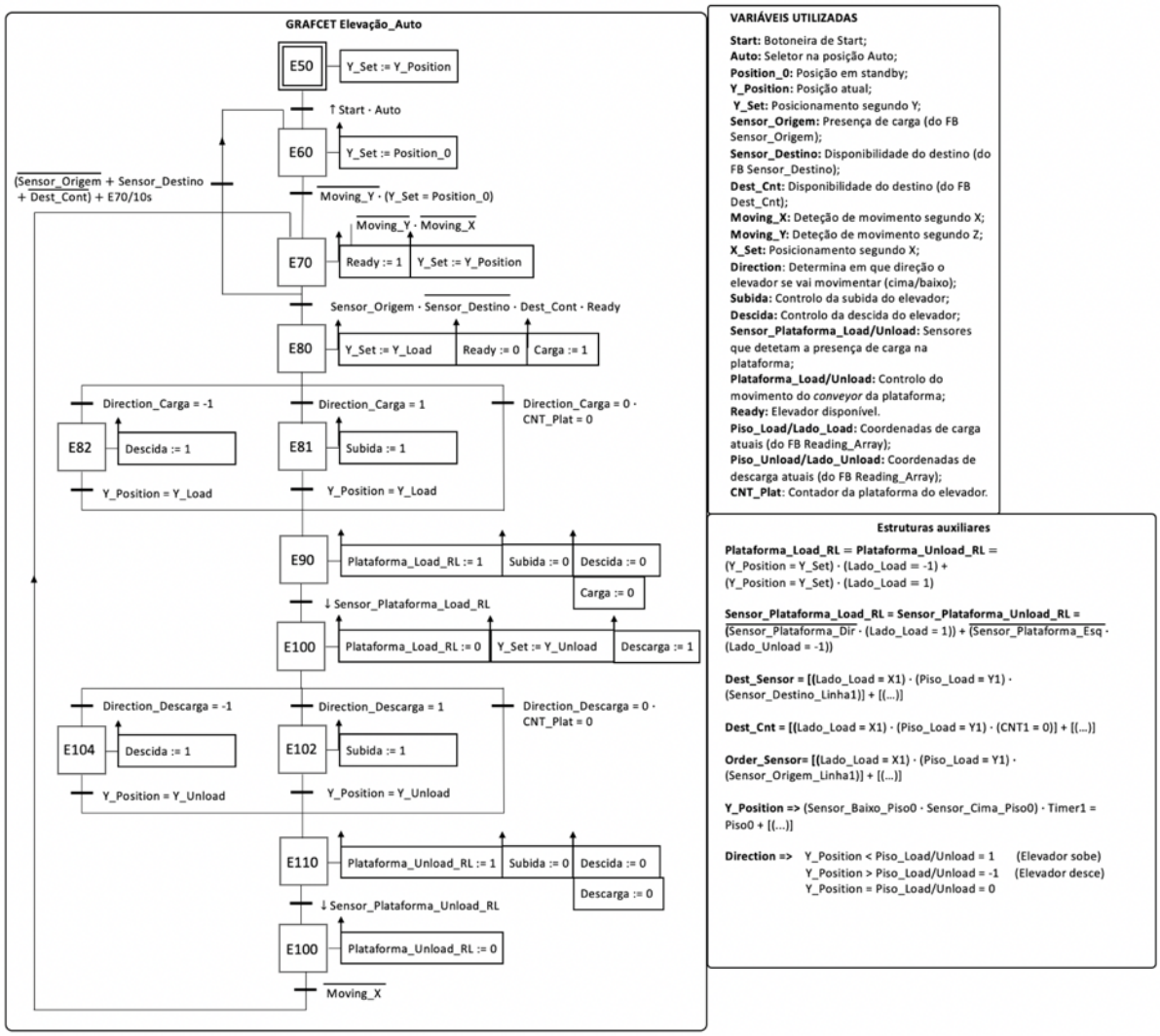


Figura 39 - GRAFCET de controlo do FB_Elevação_Auto.

FB_Reading_List: Responsável pela leitura das ordens. Lê a ordem de índice i, a partir da lista de ordens (fornecida pela DB_Ordens, mencionada posteriormente na Secção Database), extraindo as coordenadas de carga (Piso_Load e Lado_Load) e de descarga (Piso_Unload e Lado_Unload). Este FB é o único programado em texto estruturado, uma vez que, como se pode observar pela Figura 40, em linguagem LD não é possível fazer a indexação de um vetor através de uma variável interna.

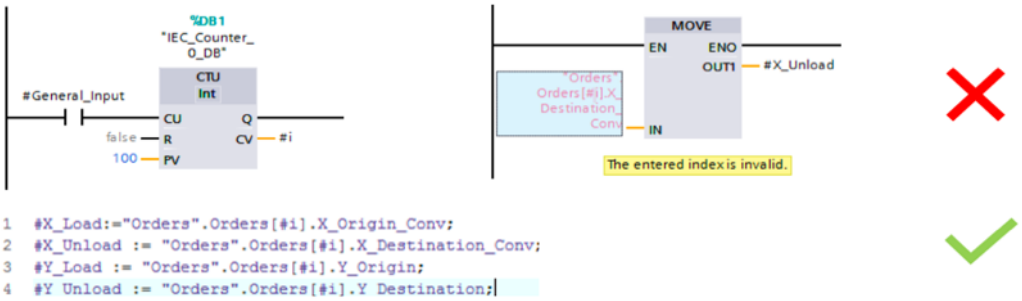


Figura 40 - Indexação de um vetor através de uma variável interna. In Henrique Domingos, Desenvolvimento de uma aplicação para geração automática de código para o PLC de controlo de um *shuttle car*, 2018.

FB_Reading_Array: Assim que o elevador estiver pronto, ou seja, dentro dos limites de software e sem qualquer operação em curso, este *function block* é o responsável por garantir o cumprimento das ordens (lidas a partir do FB_Reading_List) de forma sequencial. As coordenadas de carga e descarga lidas na ordem por este bloco são passadas ao *function block* que controla o funcionamento automático do elevador – FB_Elevação_Auto.

FB_Sensor_Origem: Avalia o estado do sensor colocado no final da linha de carga (imediatamente antes da plataforma do elevador). O estado deste sensor (inativo – sem carga; ativo – com carga pronta a ser transportada) é transmitido ao *function block* de controlo do elevador em modo automático, que avalia se a ordem pode ou não ser executada.

FB_Sensor_Destino: Avalia o estado do sensor colocado no início da linha correspondente à coordenada de descarga da ordem atual. O estado deste sensor (inativo – sem carga; ativo – carga presente no local de destino) é transmitido ao *function block* de controlo do elevador em modo automático, que avalia se a ordem pode ou não ser executada.

FB_Contador_Destino: Avalia o valor do contador do *conveyor* da linha correspondente às coordenadas de descarga presentes na ordem atual. O valor deste contador (1 – disponível; 0 – não disponível) é passado ao *function block* que controla o funcionamento automático do elevador, que avalia se a ordem pode ser executada.

Ao contrário do caso do *shuttle car* apresentado na Secção 3.5, o cenário utilizado para o controlo do elevador não permite conhecer a sua posição absoluta a qualquer momento, sendo essencial incluir um conjunto de FBs, cujas funções são exclusivamente a de determinação da piso atual do elevador (FB_Y_Position) e de comparação dessa piso com a pretendida na ordem a executar, de maneira definir se o elevador terá de descer ou subir (FB_Compare_Direction) para cumprir a ordem. Também o movimento vertical do elevador não pode ser comparado com o movimento horizontal realizado pelo *shuttle car*. Como tal, dois FBs foram criados apenas com o objetivo de controlar esse movimento e as variações da aceleração do elevador aquando da necessidade de paragem num determinado piso.

FB_Compare_Direction: Compara a posição/piso atual do elevador com o piso presente na ordem a executar. Caso:

- Piso_Atual>Piso_Ordem – Direction=-1 e elevador desce;
- Piso_Atual<Piso_Ordem – Direction=1 e elevador sobe;
- Piso_Atual=Piso_Ordem – Direction = 0 e a plataforma não se move na vertical.

FB_Y_Position: Perante a impossibilidade de conhecer a posição absoluta do elevador, este FB, através da avaliação dos sensores do elevador, retorna à posição/piso em que este se encontra parado.

FB_Plat_Elevador_Descida: Responsável pela descida da plataforma do elevador. Este FB é iniciado a partir da informação proveniente do FB_Elevação_Auto. O seu número de networks é variável consoante o número de pisos, não sendo este instanciado para o piso 0.

FB_Plat_Elevador_Subida: Responsável pela subida da plataforma do elevador. Este FB é iniciado a partir da informação proveniente do FB_Elevação_Auto. O seu número de networks é variável consoante o número de pisos, não sendo este instanciado para o piso mais elevado do elevador.

FB_Conv_Entrada: Este *function block* permite controlar os *conveyors* que funcionam como linhas de entrada de carga na plataforma do elevador.

FB_Conv_Saida: Este *function block* permite controlar os *conveyors* que funcionam como linhas de descarga da plataforma do elevador.

De maneira a que a simulação funcione de forma o mais semelhante possível à realidade, foi necessário adicionar um outro *function block* que controle a simulação no FactoryIO®, descrito a seguir:

FB_#Simulation: Controla o aparecimento de paletes nas linhas de carga do ambiente de simulação desenvolvido no FactoryIO®, através da ativação dos *emitters* existentes nas linhas de carga, consoante a coordenada lida na ordem a executar (a partir do FB_Reading_List). Perante a leitura de uma nova ordem, é verificada a disponibilidade do *conveyor* correspondente para receber, ou não, a carga (de acordo com o estado do sensor do início da linha de entrada). Caso esteja disponível, a palete é emitida; caso contrário, o *emitter* aguarda até que esta condição se verifique para emitir a palete.

Organization Blocks

De forma a instanciar os *function blocks* necessários ao controlo do sistema é necessário criar uma *organization block* Main, sendo os FBs instanciados de acordo com as variações na configuração do sistema a controlar. O OB foi também utilizado para fazer a passagem direta das variáveis e *inputs*/*outputs* necessários a cada FB.

Para cada *function block* instanciado, uma *instance data block* é criada (DBs) e atribuída a um bloco específico, contendo os dados das variáveis declaradas nesse FB.

Global Database Blocks

No caso de uma *global database blocks,* todos os blocos (FBs, FCs e OBs) podem ler os dados contidos na DB. As DBs descritas em seguida foram utilizadas de modo a acomodar algumas variáveis necessárias ao controlo do sistema.

DB_Global_Data: Contém diversas variáveis gerais, necessárias ao controlo do cenário entre os quais os limites de *software*, tempos de carga e descarga dos *conveyors*,

DB_Ordens: Contém as indicações das ordens a serem cumpridas pelo elevador. Cada ordem é composta por dois pares de coordenadas: as coordenadas de origem, correspondentes ao piso

e ao lado onde o elevador deverá recolher a carga (Piso_Load e Lado_Load) e as coordenadas de destino, compostas pelo piso e o lado em que o elevador deverá descarregar a carga que recolher (Piso_Unload e Lado_Unload);

Tagtables

Foram também criadas duas *tagtables* que acomodam os inputs e outputs necessários ao controlo do sistema.

Inputs: Contém os inputs físicos (sensores, botoneiras, etc.) e respetivo endereçamento no PLC.

Outputs: Contém os outputs físicos (motores do elevador e dos *conveyors*, etc.) e respetivo endereçamento no PLC.

5 Manipulação de ficheiros XML no Microsoft Visual Studio®

Neste capítulo pretende-se descrever a aplicação que deverá ser desenvolvida no Microsoft Visual Studio, de forma a que seja possível gerar código PLC através da manipulação de ficheiros XML modelo.

Como tal, começa-se por descrever, de forma geral, o objetivo pretendido para a aplicação e expõe-se a interface da aplicação, de modo a que se possa compreender as funções que deverão ser implementadas. De seguida, explora-se aprofundadamente o método pretendido a utilizar para implementar a geração de código - a manipulação de documentos XML.

5.1 Objetivo da aplicação

Como referido anteriormente, o objetivo da aplicação desenvolvida é gerar o código do programa de controlo do elevador, de modo a flexibilizar a programação destes sistemas, minimizando o seu tempo de desenvolvimento.

A aplicação em questão tem como objetivo gerar o código de controlo do elevador tendo em conta as variantes expostas na Secção 3.6. Assim sendo, será possível controlar sistemas semelhantes com *layouts* distintos, como os ilustrados nas Figura 41 e Figura 42.

Figura 41 - Sistemas com layouts distintos em ambiente de simulação: a) Sistema A; b) Sistema B.

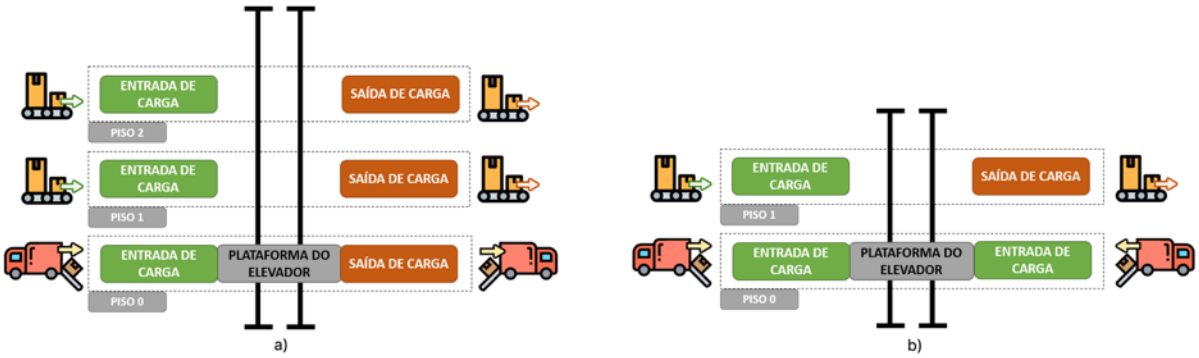


Figura 42 - Sistemas com layouts distintos: a) Sistema A; b) Sistema B.

5.2 Interface proposta da aplicação

A Figura 43 ilustra uma possível interface gráfica para a aplicação em causa, contendo uma descrição dos seus constituintes (imagem adaptada, da dissertação de Domingos (2018), aos requisitos do sistema em questão).

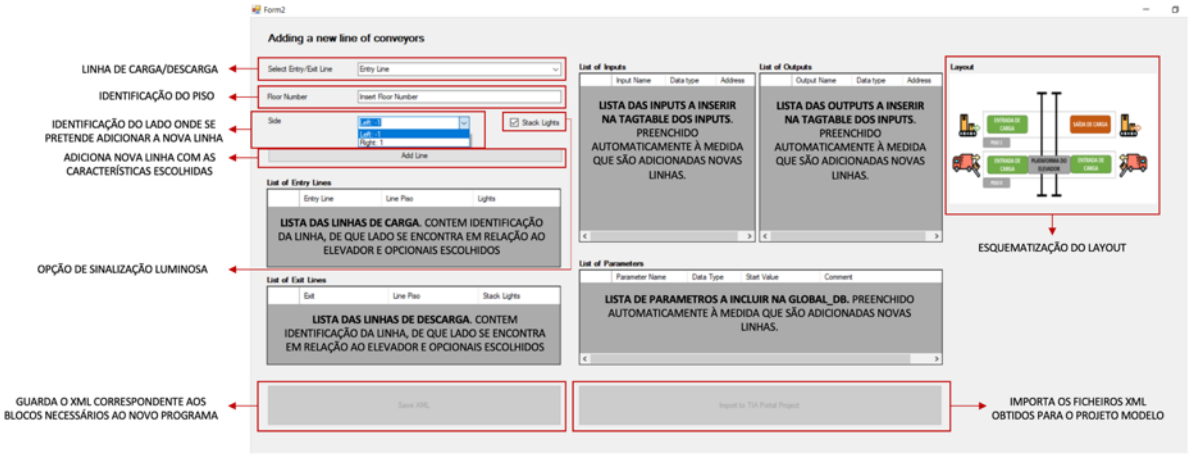


Figura 43 – Interface gráfica proposta da aplicação.

Neste caso, a interface gráfica permite definir a configuração do sistema a controlar a partir da introdução de linhas de carga/descarga no *layout*. Ao adicionar cada nova linha é necessário indicar um conjunto de informações, entre as quais:

- Tipo de linha: carga ou descarga;
- Posicionamento da linha: em que piso e de que lado da estrutura se encontra;
- Pretensão de incluir opcionais - sinalizadores em coluna (*stacklights*).

Além disso, a interface permite também visualizar as linhas introduzidas e, caso necessário, editá-las ou apagá-las a partir das listas de linhas.

Adicionalmente, de cada vez que é introduzida uma nova linha, é possível acompanhar a inserção de variáveis nas listas de *Inputs, Outputs* e parâmetros, sendo possível editá-las.

Por último, a interface gráfica permite fazer a gravação dos documentos XML modificados e a importação dos mesmos para o projeto modelo a partir dos botões "*SaveXML"* e "*Import to TIA Portal Project"*, respetivamente.

5.3 Funcionamento pretendido da aplicação

Nesta secção é analisada a estrutura pretendida para a aplicação, o comportamento desejado aquando da inserção de uma nova linha no sistema assim como no cumprimento das funções de gravação dos ficheiros XML e importação dos mesmos para o projeto modelo do TIA Portal®, que contém apenas o hardware a utilizar (PLC e cartas I/O) e os elementos de software que não necessitam de alterações.

5.3.1 Estrutura da aplicação

Inicialmente, a aplicação deverá ler um conjunto de ficheiros XML modelo, correspondentes aos elementos de software que se pretende manipular (OBs, FBs, *Global* e *Instance* DBs e *tagtables*).

Após definir na interface todos os parâmetros desejados em relação ao *layout* definido, a inserção destes dados deverá permitir que a aplicação defina a informação a introduzir nos ficheiros XML modelo, de maneira a que estes, após manipulados, contenham o código integral a implementar, necessário ao controlo do sistema. Uma vez tendo os documentos XML necessários, a sua importação é feita para o projeto modelo desenvolvido no TIA Portal® (Figura 44).

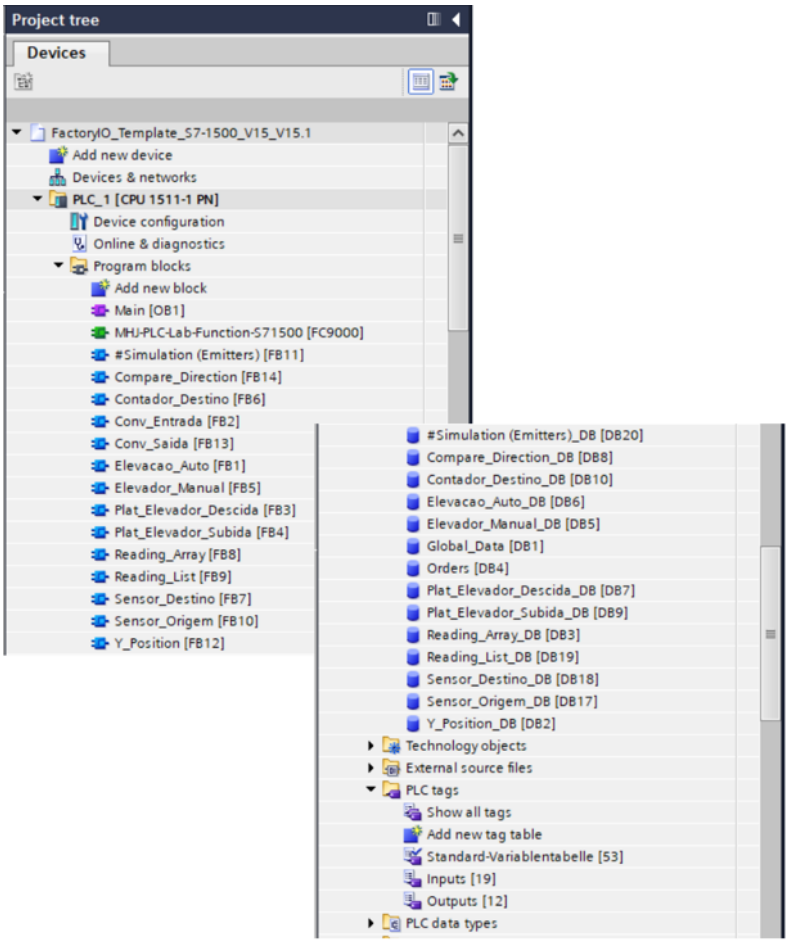


Figura 44 - Elementos de software contidos no programa modelo.

5.3.2 Adição de uma nova linha

Após o preenchimento da informação pedida relativamente à linha (Secção 5.2), a adição de uma nova linha deverá ser conseguida através do uso do botão "*Add Line*" da

interface. A utilização deste botão deverá ser responsável unicamente pelo preenchimento das tabelas de *inputs, outputs* e parâmetros incluídas na interface, passando a conter a informação necessária na qual a aplicação se baseia para, posteriormente, efetuar a manipulação dos documentos XML modelo, não produzindo qualquer alteração nos documentos XML modelo.

A adição de uma nova linha de carga implica o cumprimento dos seguintes passos:

1. Adição de *inputs* (sensores de cima e baixo do piso em questão – caso seja a primeira linha adicionada ao piso - e sensores de entrada e saída do *conveyor* adicionado) e *outputs* (motores do *conveyor* e sinalização opcional) às respetivas *tagtables* e de novos parâmetros à *database* Global_Data, como consequência da inclusão de novos *conveyors*.
2. Alteração do FB_Sensor_Origem, dado que passa a existir uma nova coordenada de origem e, consequentemente, um novo sensor cujo estado tem de ser verificado, e da respetiva DB (FB_Sensor_Origem);
3. Instanciação do FB_Order_Sensor e identificação da respetiva DB no OB_Main;
4. Alteração da *instance* DB correspondente ao FB que controla o *conveyor* da nova linha;
5. Instanciação do FB que controla o *conveyor* adicionado no OB_Main;
6. Preenchimento dos *inputs* que dizem respeito a informações provenientes da DB do *conveyor* da nova linha nos FBs Y_Position e Plat_Elevador_Descida/Subida, uma vez que estes dependem do número de pisos do elevador.

Por sua vez, a adição de uma nova linha de descarga implica:

1. Alteração dos FBs Sensor_Destino e Contador_Destino, dado que passa a existir uma nova coordenada de destino e, consequentemente, um novo sensor cujo estado tem de ser verificado e um novo contador cujo valor deverá ser avaliado, respetivamente;
2. Alteração das *instance* DBs correspondentes aos FBs Sensor_Destino e Contador_Destino;
3. Instanciação dos FBs Sensor_Destino e Contador_Destino e identificação das respetivas DBs no OB_Main;
4. Aplicação dos passos 1, 4, 5 e 6 descritos para a linha de carga.

5.3.2.1 Maximização da modularidade

Na Secção 5.3.2 é descrito o procedimento ideal que a aplicação deve seguir quando se adiciona uma nova linha ao cenário. Por outro lado, tendo em conta os armazéns produzidos pela Consoveyo, S.A., é possível assumir que, num cenário real, não é normal que um elevador estabeleça a ligação entre mais do que três pisos distintos (piso 0, 1 e 2, no máximo).

Posto isto, e de forma a maximizar a modularidade do programa modelo, este foi modificado adotando uma estratégia em que se considera a possibilidade do elevador ter o número máximo de pisos: três. Desta forma, caso seja adicionada uma nova linha, as alterações requeridas pela aplicação são reduzidas, em comparação com a proposta mencionada no início da presente secção. Com o âmbito de entender melhor esta adaptação do programa, exploram-se as manipulações necessárias aquando da adição de novas linhas.

A adição de uma nova linha de carga é resumida aos seguintes passos:

1. Adição de *inputs* (sensores de entrada e saída do *conveyor* adicionado) e *outputs* (motores do *conveyor* e sinalização opcional) às respetivas *tagtables* e de novos parâmetros à *database* Global_Data, como consequência da inclusão de novos *conveyors*.
2. Alteração da *instance* DB correspondente ao FB que controla o *conveyor* da nova linha;
3. Instanciação do FB que controla o *conveyor* adicionado no OB_Main;

Da mesma forma, a adição de uma nova linha de descarga implica apenas:

1. Aplicação dos passos 1, 2 e 3 descritos para a linha de carga.

Assim, a necessidade da alteração dos FBs Sensor_Destino (apresentado na Figura 45 a título de exemplo), Sensor_Origem e Contador_Destino é eliminada, uma vez que todas as possibilidades estão à partida englobadas no programa modelo. Além disso, os FBs Y_Position e Plat_Elevador_Descida/Subida estão também desenvolvidos de maneira a abranger a possibilidade da existência de três pisos, funcionando, sem qualquer problema, no caso de o sistema ter apenas dois pisos, não havendo necessidade de qualquer modificação destes FBs.

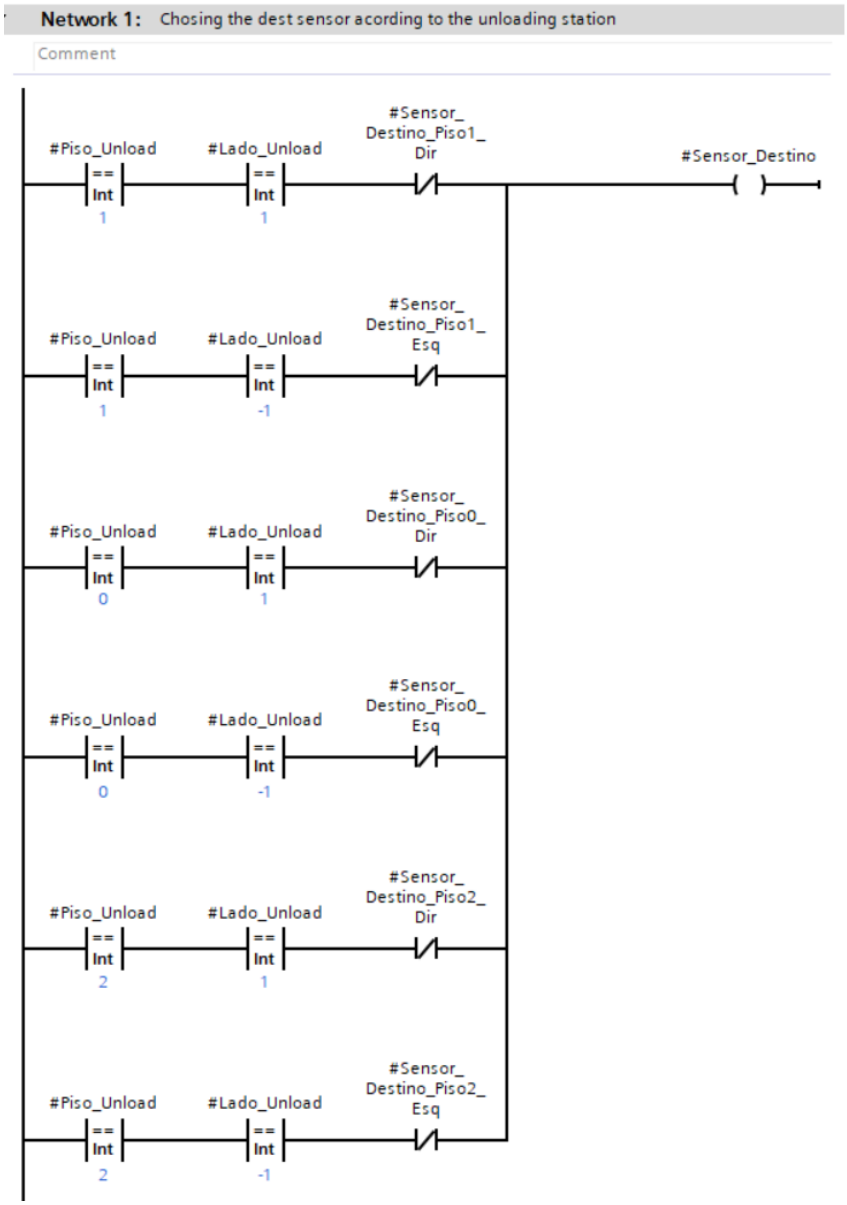


Figura 45 - FB Sensor_Destino no caso da maximização da modularidade do programa.

5.3.2.2 Elementos sujeitos a manipulações

Feitas as alterações que maximizam a modularidade do programa, a quantidade de elementos a manipular pela aplicação é reduzida. Fazendo um paralelismo entre a Figura 46 e a Figura 32, apresentada na Secção 3.5.5.2, que descreve o trabalho realizado previamente por Henrique Domingos, é possível verificar que também em comparação com este projeto, o número de blocos alterados pela aplicação é menor.

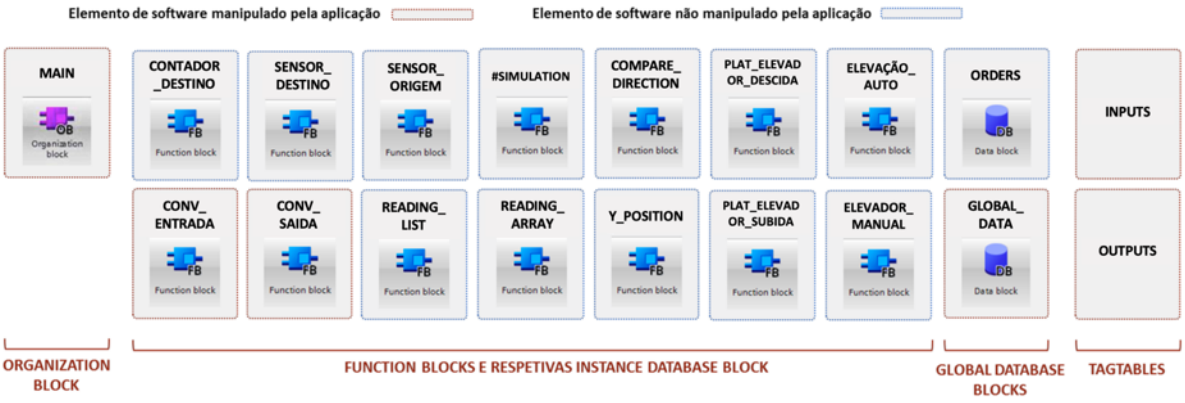


Figura 46 - Blocos que necessitam, ou não, de ser manipulados pela aplicação. Adaptado de Henrique Domingos, Desenvolvimento de uma aplicação para geração automática de código para o PLC de controlo de um *shuttle car*, 2018.

5.3.3 Gravação dos ficheiros XML

É através da função de gravação, acessível a partir do botão "*Save XML*", que se executam as alterações nos documentos XML, considerando as informações contidas nas listas de *inputs, outputs* e parâmetros.

5.3.4 Importação dos ficheiros XML para o projeto modelo

O botão "*Import to TIA Portal Project*", utilizado para executar a função de importação, deverá ser responsável pela execução de uma série de ações: iniciação de uma nova instância do TIA Portal®, abertura do projeto modelo e importação dos documentos XML manipulados. Exibem-se, em seguida, os métodos utilizados para cada uma das ações referidas, implementadas a partir da interface do TIA Portal Openness:

1. StartTIA(): Iniciação de uma nova instância do TIA Portal® (Figura 47);

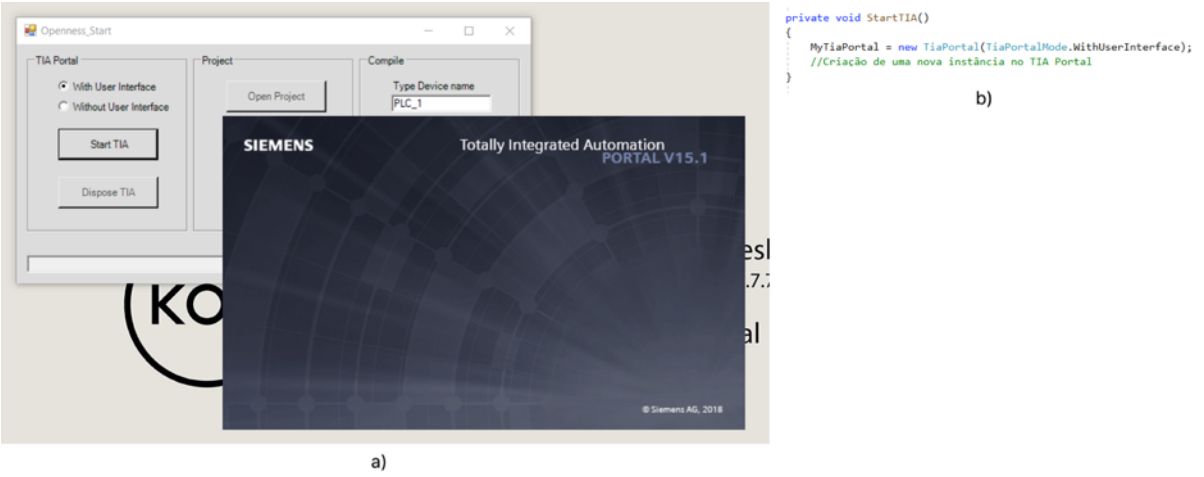


Figura 47 - Chamada de uma nova instância do TIA Portal: a) Interface; b) Método utilizado.

2. OpenProject(): Abertura do projeto modelo previamente definido (Figura 48);

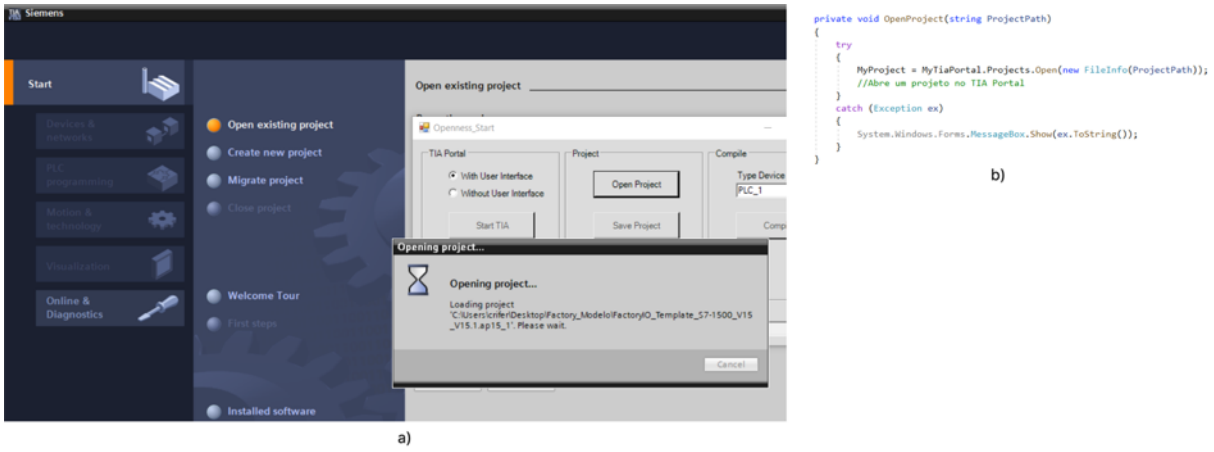


Figura 48 - Abertura do projeto modelo: a) TIA Portal; b) Método utilizado.

3. ImportBlocks(): Responsável pela importação dos OBs, FBs, *global* e *instance* DBs;

Uma vez que se pretende adicionar elementos de *software* ao programa modelo, a implementação deste método exige o acesso à área *Program Blocks* do TIA Portal®, que corresponde à área de destino deste tipo de elementos. Como se pode constatar a partir da observação da Figura 49, em primeiro lugar acedeu-se ao *Device* (PLC_1), seguido do acesso ao seu software (*plcSoftware*), que contém a área *ProgramBlocks*.

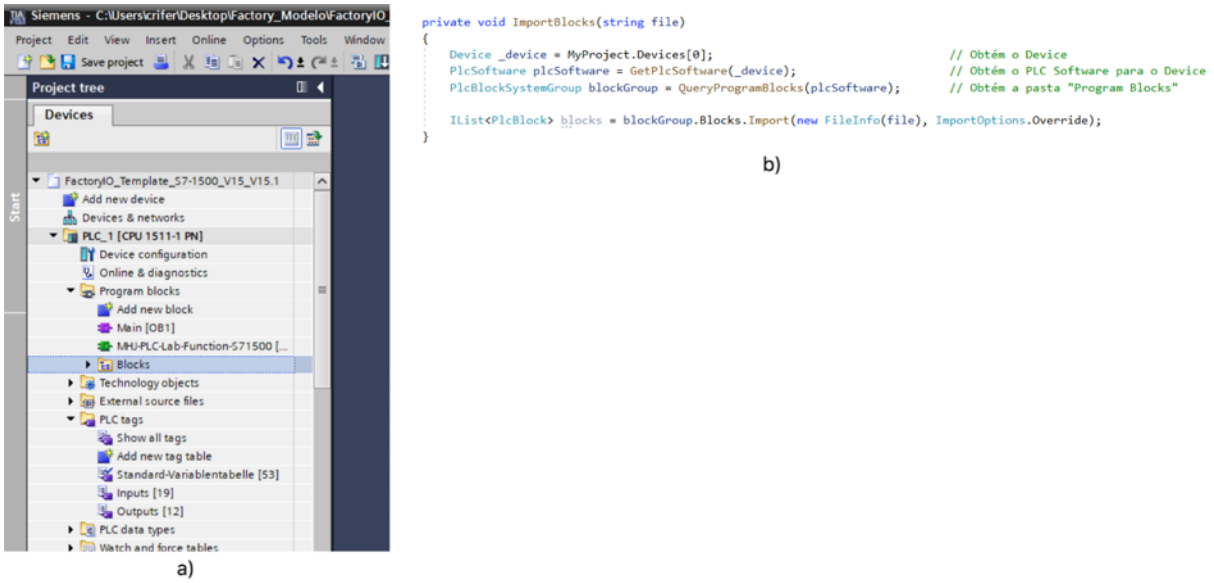


Figura 49 - Acesso e importação para a área *Program Blocks*: a) Árvore do TIA Portal; b) Método utilizado.

4. ImportTagTable(): Importação das *tagtables.*

Para além do acesso à área dos *ProgramBlocks*, de maneira a implementar este método, é necessário também o acesso à área de destino das *Tags*, visível na árvore do TIA Portal da Figura 49. O método utilizado para aceder a esta área, explicitado na Figura 50 deverá ser semelhante ao utilizado para aceder à área *ProgramBlocks*.

```
private void ImportTagTable(string file)
{
    Device _device = MyProject.Devices[0];                                          // Obtém a Station
    PlcSoftware plcSoftware = GetPlcSoftware(_device);                              // Obtém o PLC Software para a Station
    PlcTagTableSystemGroup plcTagTableSystemGroup = GetControllerTagfolder(plcSoftware);   // Obtém o grupo PLCTagTables
    PlcTagTableComposition tagTables = plcTagTableSystemGroup.TagTables;            // Obtém a past "PLCTags"
    tagTables.Import(new FileInfo(file), ImportOptions.Override);
}
```

Figura 50 - Acesso e importação para a área de *Tags*.

5.4 Geração de código a partir da manipulação de documentos XML

Nesta secção aborda-se a forma a partir da qual se pretende gerar o programa de controlo do elevador, com uma configuração predefinida, a partir da manipulação de ficheiros XML.

Passa-se então por, em primeiro lugar, descrever a estrutura geral dos documentos XML exportados do TIA Portal®, bem como o processo de criação dos documentos XML modelos a serem utilizados pela aplicação. É neste capítulo apresentado o método para alteração desses mesmos documentos.

5.4.1 Estrutura dos documentos XML do TIA Portal®

Uma vez que se optou por desenvolver uma aplicação que gerasse código a partir da manipulação dos documentos XML do TIA Portal®, é necessário analisar a sua estrutura.

Assim, analisa-se em seguida a estrutura geral dos documentos XML do TIA Portal®, bem como a estrutura de um *organization block* em particular.

Estrutura Geral

A estrutura de um documento XML de um elemento de software extraído do TIA Portal® pode variar consoante o tipo de exportação selecionado, o tipo de elemento exportado (FB, OB, DB, etc.) e da linguagem de programação do elemento.

Os ficheiros XML do TIA Portal® são compostos por duas áreas (Figura 51): *DocumentInfo*, que contém a informação relativa à criação do documento e aos produtos instalados no TIA Portal®; e *SW.CodeBlock*, que descreve o elemento de software exportado.

```
<?xml version="1.0" encoding="utf-8"?>
<Document>
  <DocumentInfo>
  <SW.CodeBlock ID="0">
</Document>
```

Figura 51 - Estrutura de um ficheiro XML do TIA Portal.

A área *SW.CodeBlock* descreve o bloco, podendo variar na sua designação e conteúdo conforme o elemento de software que descreve (Figura 52), e inclui também duas outras áreas: *AttributeList*, que contém informação geral sobre o elemento exportado e descreve a sua interface, caso esta exista (inputs, outputs, etc.) e *ObjectList*, que contém diversos objetos identificados por IDs (comentários, títulos, *networks* - *SW.CompileUnits* - , tags, etc.) O XML atribui um ID com valor distinto, entre 0 e 2147483647, que identifica cada objeto usado.

```
<?xml version="1.0" encoding="utf-8"?>
<Document>
  <DocumentInfo>
  <SW.Tags.PlcTagTable ID="0">
</Document>
```

a)

```
<?xml version="1.0" encoding="utf-8"?>
<Document>
  <DocumentInfo>
  <SW.Blocks.GlobalDB ID="0">
</Document>
```

b)

```
<?xml version="1.0" encoding="utf-8"?>
<Document>
  <DocumentInfo>
  <SW.Blocks.FB ID="0">
</Document>
```

c)

Figura 52 - Designações da área SW.CodeBlock para os componentes: a) *TagTable*; b) Global DB; c) FB.

A Figura 53 mostra a relação entre estes elementos e a estrutura do ficheiro XML.

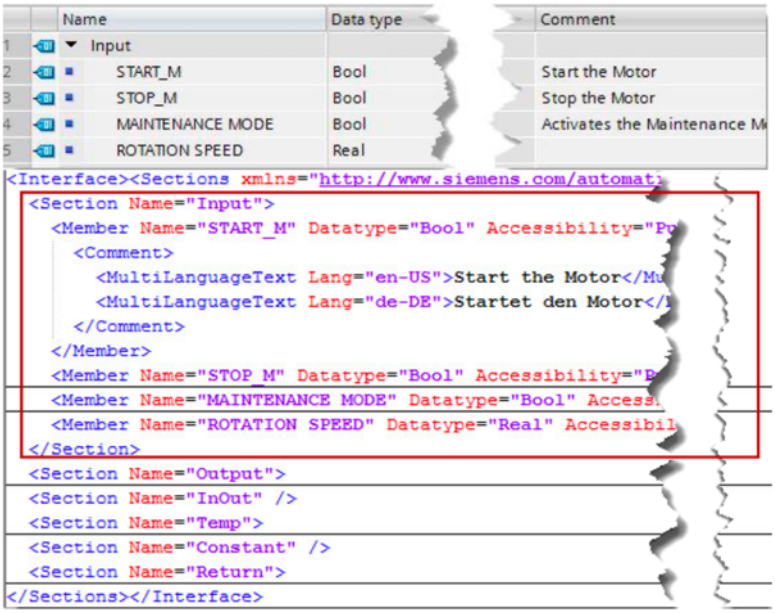


Figura 53 - Interface de um componente.

Cada área *SW.CompileUnit* pode conter uma área *ObjectList* e uma *AttributeList*. A *ObjectList*, caso exista, contém os comentários da *network*. A *AttributeList* inclui outras duas áreas: a *NetworkSource*, composta pelas *Parts* (variáveis e comandos usados na network) e pelos *Wires* (relação entre essas variáveis e os comandos) e a *ProgrammingLanguage*, que identifica a linguagem de programação utilizada na *network*.

As *Parts* e *Wires* de cada programa têm associado um *UId* (Unique Identifier), que deve ser único dentro da *SW.CompileUnit* (gama de valores entre 21 e 2147483647). A Figura 54 mostra a relação entre estes elementos e a estrutura de um documento XML.

Figura 54 - Exemplo geral da relação entre *Parts* e *Wires*.

Estrutura do documento XML de um OB

Na Figura 55 dispõe-se e analisa-se a estrutura do documento XML do organization block do programa desenvolvido.

Como se pode comprovar pela observação da Figura 55, neste caso, a área *SW.CodeBlock* é designada por *SW.Blocks.OB*, uma vez que se trata de um *organization block,* sendo composta por duas áreas já mencionadas nesta secção: *AttributeList* e *ObjectList*. A primeira, *AttributeList*, identifica a interface do bloco. A segunda, *ObjectList*, é maioritariamente composta por *networks* (*SW.CompileUnit*), que contêm o código presente no *organization block.*

Figura 55 - Estrutura do XML de um *organization block*.

5.4.2 Documentos XML modelo

Como descrito na Secção 5.3.1, a aplicação deveria executar a geração de código a partir da manipulação de documentos XML modelo correspondentes aos diversos elementos de software de um projeto do TIA Portal® (OBs, FBs, *instance* DBs, *global* DBs e *tagtables*).

De maneira a obter estes ficheiros XML modelo necessários ao funcionamento da aplicação, depois de definido o programa modelo no TIA Portal® e usando a aplicação *DemoOpenness* (Secção 3.4.1.3), exportaram-se os documentos XML correspondentes aos vários elementos de *software* desse programa de controlo modelo. Os ficheiros XML obtidos neste processo são semelhantes aos obtidos através da exportação dos documentos XML do programa de controlo desenvolvido na Secção 4.2, contudo possuem menos informação uma vez que não contêm ainda qualquer informação acerca dos *conveyors* de entrada e saída.

6 Conclusões e trabalhos futuros

6.1 Conclusão

A complexidade na elaboração de programas de controlo de PLCs na indústria de armazéns automáticos tem vindo, de facto, a aumentar ao longo dos anos. Com o aumento desta complexidade, acresce também a necessidade da utilização de ferramentas de programação destes equipamentos mais poderosas e competitivas. É neste sentido que se desenvolve este projeto de dissertação, que teve como principal objetivo o desenvolvimento de uma aplicação de software que possibilitasse a geração automática de código para o controlo do PLC de um elevador.

Identificaram-se e analisaram-se algumas alternativas que permitissem a flexibilização da programação de PLCs, como a reutilização de código e a geração de código propriamente dita, bem como diversas ferramentas existentes, que permitiam abordar cada uma destas opções.

Incluída na bibliografia consultada, encontra-se uma dissertação, elaborada em 2018 por Henrique Domingos, desenvolvida no contexto da mesma empresa, cujo tema e objetivo são bastante semelhantes aos do presente trabalho. Esta dissertação, seguida como base para o projeto proposto, demonstra que a geração de código para controlo de um *shuttle car* é possível, através de uma aplicação em C#. Esta solução é conseguida através da interação entre o TIA Portal® e o Microsoft Visual Studio®, por meio de uma API do primeiro software referido – o TIA Portal Openness®. No entanto, importa referir que a adaptação desse para o caso da programação um elevador não é trivial.

Na verdade, foram várias as dificuldades enfrentadas ao longo do percurso de desenvolvimento desta dissertação. Começando pela pandemia, COVID19, que se sentiu em todo mundo durante o tempo de execução do projeto, o distanciamento tanto da empresa como da faculdade e do orientador foram forçados. Assim, reduziram-se não só os apoios como os recursos que estavam previstos à execução do projeto em questão. Para além disso, também o programa de controlo do elevador se revelou mais complexo do que o esperado, atrasando todo o processo que daí deveria partir.

Uma vez que o sistema estudado em 2018 tem muito pouco em comum com o sistema de elevação tratado na presente dissertação, um programa totalmente novo teve que ser elaborado. Neste processo, foi possível reutilizar sem alterações, apenas dois FBs do programa já existente, tendo que ser implementada uma nova lógica para o controlo da elevação (subida e descida) do elevador e do funcionamento dos *conveyors* que transportam a carga até/da plataforma do elevador. Assim, conclui-se que a flexibilização de código PLC é muito limitada a contextos específicos, devendo ser sujeita a mais trabalho e atenção.

Apesar das visíveis diferenças entre os dois sistemas, optou-se por seguir o mesmo raciocínio que o utilizado por Domingos (2018) no que diz respeito à obtenção de um produto

capaz de gerar código automaticamente, partindo de um conjunto de parâmetros que descrevem o sistema a controlar, como o proposto para o controlo do *shuttle car*.

Como tal, o TIA Portal Openness® foi, das ferramentas estudadas, a selecionada para o desenvolvimento do projeto em causa devido às suas vantagens de possibilidade comunicação com o TIA Portal®, de importar/exportar documentos XML e de manipular estes documentos partindo de uma aplicação externa.

Uma vez que, aquando do início da presente dissertação, não existia ainda qualquer programa de controlo para o elevador, optou-se por elaborar um novo programa utilizando o TIA Portal®, de forma a que seja possível operar posteriormente com o TIA Portal Openness®. Este programa desenvolvido, que tenta aproximar-se ao máximo do funcionamento dos sistemas da Consoveyo, S.A., é concebido da forma o mais modular possível e abrangendo o maior número conseguido de variantes.

Assim, a aplicação a conceber deve gerar o programa de controlo do elevador com base no código modular desenvolvido e de acordo com a seleção de um conjunto necessários de variantes a definir na interface da mesma.

Em virtude dos fatos mencionados acerca das dificuldades enfrentadas no desenrolar desta dissertação nem todos os objetivos foram alcançados já que não se obteve uma aplicação funcional para geração de código. No entanto, a quantidade de trabalho a ser executado na conceção de um programa de controlo de um elevador foi reduzida significativamente em comparação com o método tradicional, utilizado na Consoveyo, S.A., e provou-se que é possível, partindo do estado atual e com conhecimentos em C#, programar a interface proposta e obter uma aplicação operacional.

6.2 Trabalhos futuros

Tendo em consideração o projeto desenvolvido e as complicações apresentadas para finalizar a aplicação, os trabalhos futuros passam pela implementação da solução apresentada nesta dissertação no Microsoft Visual Studio®, de maneira a obter uma aplicação que gere, na prática, o código para controlo de qualquer variante do sistema de elevação da Consoveyo, S.A..

Para além disso, seria interessante aplicar estes mesmos conceitos a um programa concebido para o controlo do elevador disponível na Consoveyo, S.A.. Para tal, o programa de controlo do elevador teria que ser inteiramente desenvolvido no TIA Portal® e deverá seguir a estrutura modular proposta nesta dissertação. Nesse sentido, haveria a possibilidade de testar o funcionamento real da solução que foi desenvolvida e possivelmente realizar testes práticos de forma a avaliar a sua performance.

Referências

Advanced Micro Controls. 2020. "What's a PLC". https://www.amci.com/industrial-automation-resources/plc-automation-tutorials/what-plc/.

AIDA. 2005. "Cost structure analysis of controls and robotics". https://www.automationml.org/o.red.c/story.html.

Alphonsus, Ephrem Ryan e Mohammad Omar Abdullah. 2016. "A review on the applications of programmable logic controllers (PLCs)". *Renewable and Sustainable Energy Reviews* 60: 1185-205. https://doi.org/10.1016/j.rser.2016.01.025. http://www.sciencedirect.com/science/article/pii/S1364032116000551.

Andersson, Mikael e Erik Helander. 2010. "Automatic generation of PLC programs using Automation Designer". Master of Science Thesis in the Master Degree Program, Production Engineering, Department of Signals and Systems, CHALMERS UNIVERSITY OF TECHNOLOGY.

Armentia, A., E. Estévez, D. Orive e M. Marcos. 2018. "A Tool Suite for Automatic Generation of Modular Machine Automation Projects". Comunicação apresentada em 2018 IEEE 16th International Conference on Industrial Informatics (INDIN), 18-20 July 2018. 10.1109/INDIN.2018.8472085.

Barbosa, Manuel Romano. 2019. "Apontamentos de AF - Programação de Comando Numérico Computorizado (CNC)".

Beek, D.A. van, W.J. Fokkink, A. Hofkamp D. Hendriks, J. Markovski, J.M. van de Mortel-Fronczak e M.A. Reniers. 2014. "CIF 3: Model-based engineering of supervisory controllers". *Tools and Algorithms for the Construction and Analysis of Systems* 8413: 575-80.

Blagojevic, Vladislav, Saša Ranđelović, Vlastimir Nikolić e Slobodan Dudić. 2019. "AUTOMATIC GENERATION OF THE PLC PROGRAMS FOR THE SEQUENTIAL CONTROL OF PNEUMATIC ACTUATORS". *Facta Universitatis, Series: Mechanical Engineering* 17: 405. 10.22190/FUME190123033B.

Cheng, Chih-Hong, Chung-Hao Huang, Harald Ruess e Stefan Stattelmann. 2014. "G4LTL-ST: Automatic Generation of PLC Programs".

Consoveyo. 2019a. *Manual do Colaborador*.

Consoveyo. 2019b. "Operation Manual - Conveying System".

Consoveyo. 2020a. "Consoveyo Official Website". http://www.consoveyo.com/en/homepage.html.

Consoveyo. 2020b. "Manual_Operação_EL8x_PT_V003".

Consoveyo. 2020c. "Manual Operação Periféricos Paletes".

Darvas, D., E. B. Viñuela e I. Majzik. 2016. "PLC code generation based on a formal specification language". Comunicação apresentada em 2016 IEEE 14th International Conference on Industrial Informatics (INDIN), 19-21 July 2016. 10.1109/INDIN.2016.7819191.

Domingos, Henrique Miguel Afonso. 2018. "Desenvolvimento de uma aplicação para geração automática de código para o PLC de controlo de um shuttle car". Mestrado Integrado em Engenharia Mecânica, Engenharia Mecanica, Faculdade de Engenharia da Universidade do Porto.

Falkman, P., E. Helander e M. Andersson. 2011. "Automatic generation: A way of ensuring PLC and HMI standards". Comunicação apresentada em ETFA2011, 5-9 Sept. 2011. 10.1109/ETFA.2011.6059201.

Fernández Adiego, Borja, Daniel Darvas, Enrique Blanco Viñuela, Jean-Charles Tournier, Simon Bliudze, Jan Blech e Victor Suarez. 2015. "Applying Model Checking to Industrial-Sized PLC Programs". *IEEE Transactions on Industrial Informatics* 11: 1400-10. 10.1109/TII.2015.2489184.

Frey, G. e L. Litz. 2000. "Formal methods in PLC programming". Comunicação apresentada em Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0, 8-11 Oct. 2000. 10.1109/ICSMC.2000.884356.

Garcia, Diego Rodriguez. 2017. "GENERACIÓN DE CÓDIGO IEC-61131-3 MULTIPLATAFORMA PARA SISTEMAS DE ALMACENAJE AUTOMÁTICO". *Dissertação de Mestrado*.

Güttel, Knut, Peter Weber e Alexander Fay. 2008. *Automatic generation of PLC code beyond the nominal sequence*. 10.1109/ETFA.2008.4638565.

Hajarnavis, Vivek e Ken Young. 2008. "An investigation into programmable logic controller software design techniques in the automotive industry". *Assembly Automation* 28: 43-54. 10.1108/01445150810849000.

Han, Kwan. 2010. "Object-Oriented Modeling, Simulation and Automatic Generation of PLC Ladder Logic". Em. 10.5772/7190.

Hasdemir, İ Tolga e Salman Kurtulan. 2006. "AUTOMATIC PLC CODE GENERATION USING MATLAB". *IFAC Proceedings Volumes* 39, no. 17: 131-36. https://doi.org/10.3182/20060926-3-PL-4904.00022. http://www.sciencedirect.com/science/article/pii/S1474667016352569.

IDC Technologies. 2002. *Pratical Industrial Programming using IEC 61131-3 for PLCs*.

International Electrotechnical Commission. 2020. "IEC 61131". https://webstore.iec.ch/searchform&q=61131#.

Jbair, M., B. Ahmad, M. H. Ahmad e R. Harrison. 2018. "Industrial cyber physical systems: A survey for control-engineering tools". Comunicação apresentada em 2018 IEEE Industrial Cyber-Physical Systems (ICPS), 15-18 May 2018. 10.1109/ICPHYS.2018.8387671.

Jbair, M., B. Ahmad, M. H. Ahmad, D. Vera, R. Harrison e T. Ridler. 2019. "Automatic PLC Code Generation Based on Virtual Engineering Model". Comunicação apresentada em 2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS), 6-9 May 2019. 10.1109/ICPHYS.2019.8780213.

Mastilovich, Nikola. 2010. "Automatisation in programming of a PLC Code". Masters of Engineering in Mechatronics, Massey University, New Zealand.

MathWorks. 2020. "Simulink Coder - Generate C and C++ code from Simulink and Stateflow models". https://www.mathworks.com/products/simulink-coder.html.

Otto, Andreas e Klas Hellmann. 2010. "IEC 61131: A General Overview and Emerging Trends". *Industrial Electronics Magazine, IEEE* 3: 27-31. 10.1109/MIE.2009.934793.

Parsons, Donald F., Kyle Gress, James M. Dempsey, Joseph Ross, Williams Parsons e Stephen Parsons. 1992. "Automated storage and retrieval system".

PLCOpen. 2009. "XML Formats for IEC 61131-3".

PLCOpen. 2020. "About PLCOpen". https://plcopen.org/what-plcopen.

Pluške, Ondřej. 2017. "Automated Design of Control Application in Control Systems with Programmable Controllers". Fakulta elektrotechniky a informatiky, VŠB – Technická univerzita Ostrava.

Qamsane, Yassine, Mahmoud El Hamlaoui, Tajer Abdelouahed e Alexandre Philippot. 2018. *A Model-Based Transformation Method to Design PLC-Based Control of Discrete Automated Manufacturing Systems*.

Schumacher, Frank e Alexander Fay. 2014. "Formal representation of GRAFCET to automatically generate control code". *Control Engineering Practice* 33: 84–93. 10.1016/j.conengprac.2014.09.008.

Schumacher, Frank, Sebastian Schröck e Alexander Fay. 2013. *Tool support for an automatic transformation of GRAFCET specifications into IEC 61131-3 control code*. 10.1109/ETFA.2013.6648109.

Siemens. 2016a. "Openness - Efficient generation of program code using code generators".

Siemens. 2016b. "Siemens Documentation: Automation Designer ". https://docs.plm.automation.siemens.com/tdoc/nx/11/ad_help#uid:index_Automation Designer.

Siemens. 2017. "Digitalize your engineering with TIA Portal Openness".

Siemens. 2018a. Openness: Automating creation of projects.

Siemens. 2018b. TIA Portal Openness: Generating a Modular Machine with S7-1500.

Siemens. 2019. TIA Portal Openness: Introduction and Demo Application.

Siemens. 2020. "Standardization - Digital Workflow". https://new.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal/highlights/standardization.html.

Smith, Patrick. 2019a. "Advantages & Disadvantages of Siemens' TIA Openness". https://www.dmcinfo.com/latest-thinking/blog/id/9877/advantages-disadvantages-of-siemens-tia-openness.

Smith, Patrick. 2019b. "How to Read an Openness XML File". https://www.dmcinfo.com/latest-thinking/blog/id/9908/how-to-read-an-openness-xml-file.

Takada, H., H. Nakata e S. Horiike. 2004. "A reusable object model for integrating design phases of plant systems engineering". Comunicação apresentada em The Fourth International Conference onComputer and Information Technology, 2004. CIT '04., 16-16 Sept. 2004. 10.1109/CIT.2004.1357337.

Techopedia. 2016. "Code Generator". https://www.techopedia.com/definition/17062/code-generator.

Thapa, Devinder, Chang Park, Sang Park e Gi-Nam Wang. 2009. "Auto-generation of IEC standard PLC code using t -MPSG". *International Journal of Control Automation and Systems* 7: 165-74. 10.1007/s12555-009-0202-z.

Thramboulidis, Kleanthis. 2012. *Towards an Object-Oriented Extension for IEC 61131*. 10.1109/ETFA.2012.6489673.

Thramboulidis, Kleanthis e Georg Frey. 2011. "Towards a Model-Driven IEC 61131-Based Development Process in Industrial Automation". *JSEA* 4: 217-26. 10.4236/jsea.2011.44024.

Vogel-Heuser, B., D. Witsch e U. Katzke. 2005. "Automatic code generation from a UML model to IEC 61131-3 and system configuration tools". Comunicação apresentada em 2005 International Conference on Control and Automation, 26-29 June 2005. 10.1109/ICCA.2005.1528274.

W3C. 2008. "Extensible Markup Language (XML) 1.0 (Fifth Edition)". https://www.w3.org/TR/REC-xml/#sec-prolog-dtd.

W3Schools. 2020. "Introduction to XML". https://www.w3schools.com/xml/xml_whatis.asp.

Walsh, Norman. 1998. "A Technical Introduction to XML

". https://www.xml.com/pub/a/98/10/guide0.html?page=2#AEN58.

ANEXO A: Programa desenvolvido no TIA Portal

Grafcets dos *function blocks* desenvolvidos

Conveyor de entrada

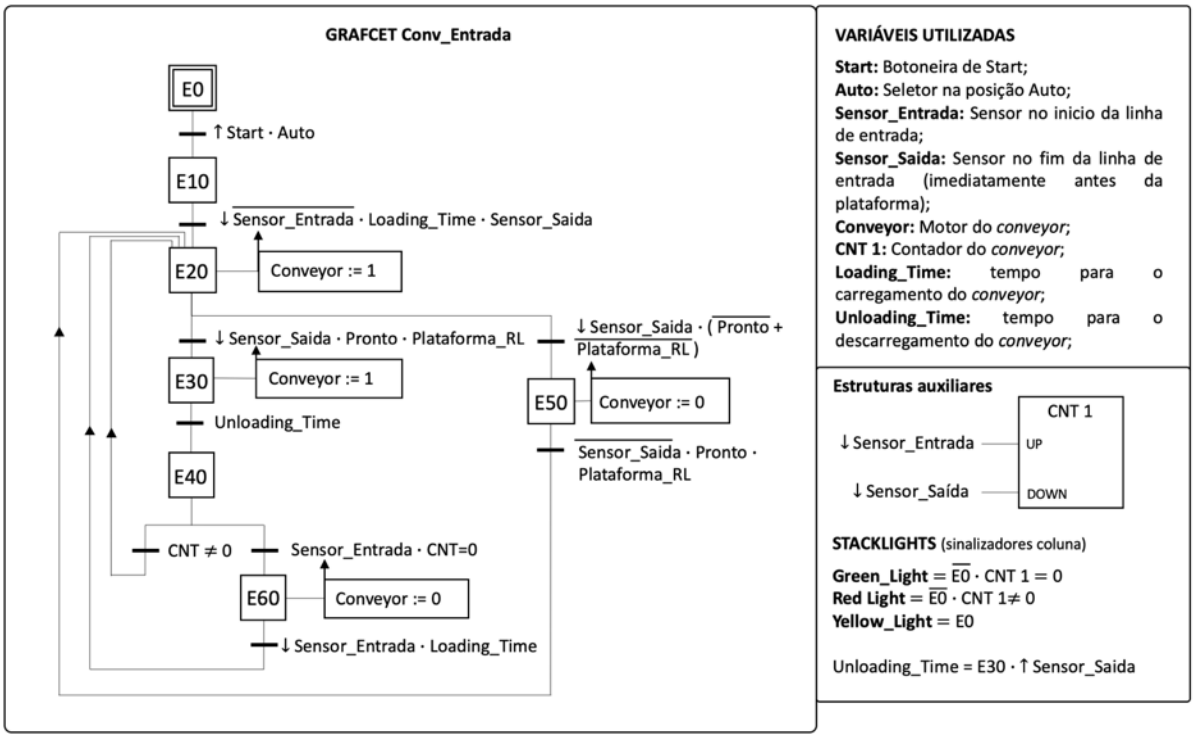


Figura 56 - GRAFCET do function block Conv_Entrada.

Conveyor de saída

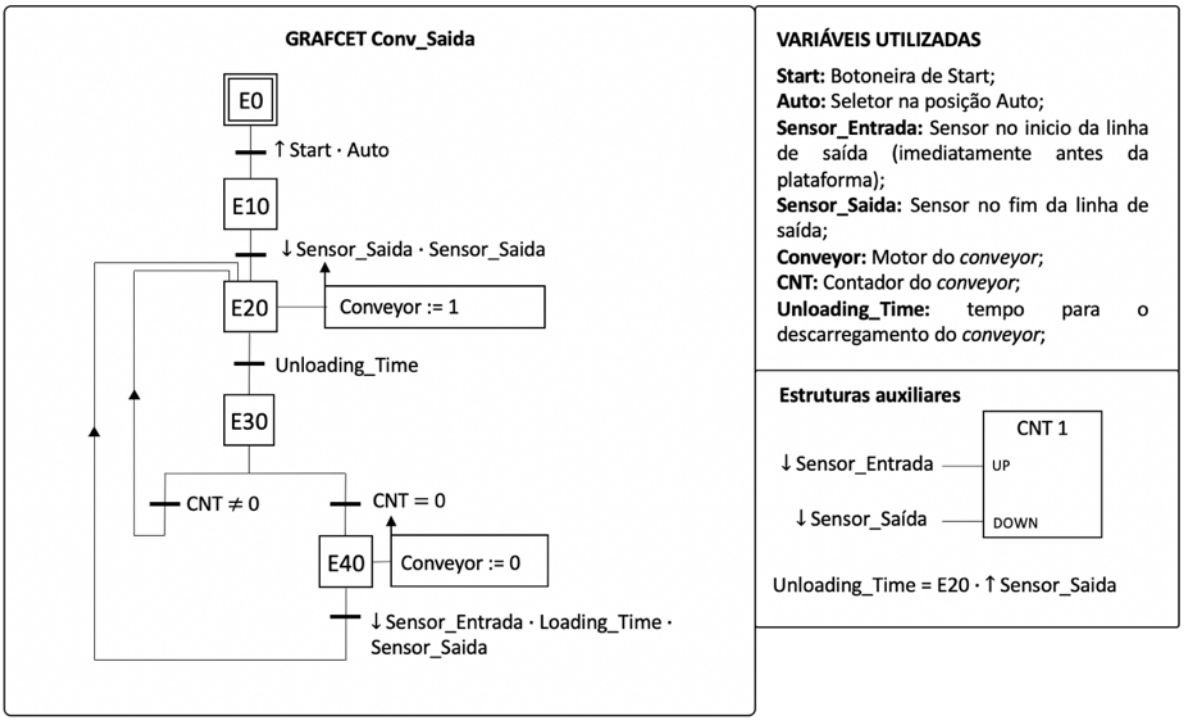


Figura 57 - GRAFCET do function block Conv_Saida.

Elevador Manual

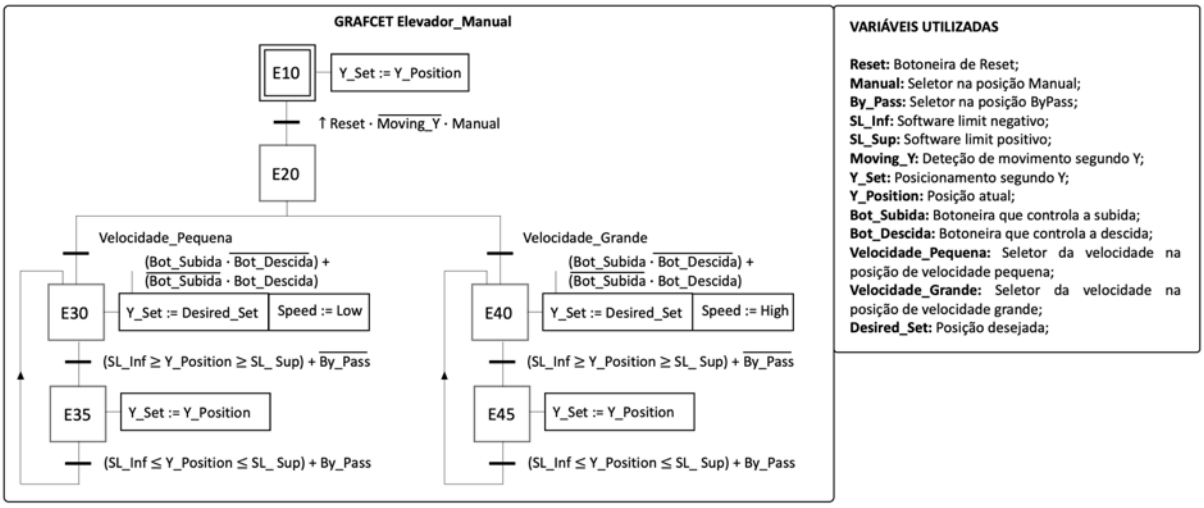


Figura 58 - GRAFCET do *function block* Elevador_Manual.

Reading_Array

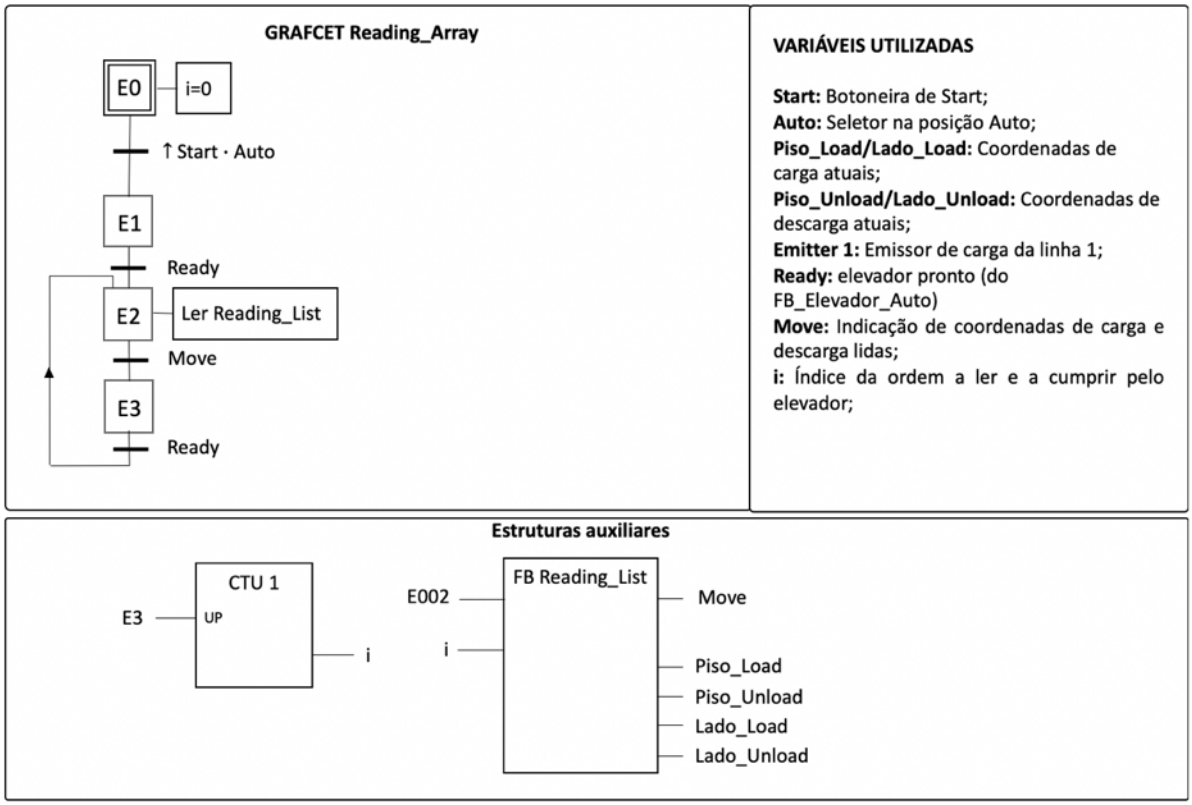


Figura 59 - GRAFCET do *function block* Reading_Array.

Simulation

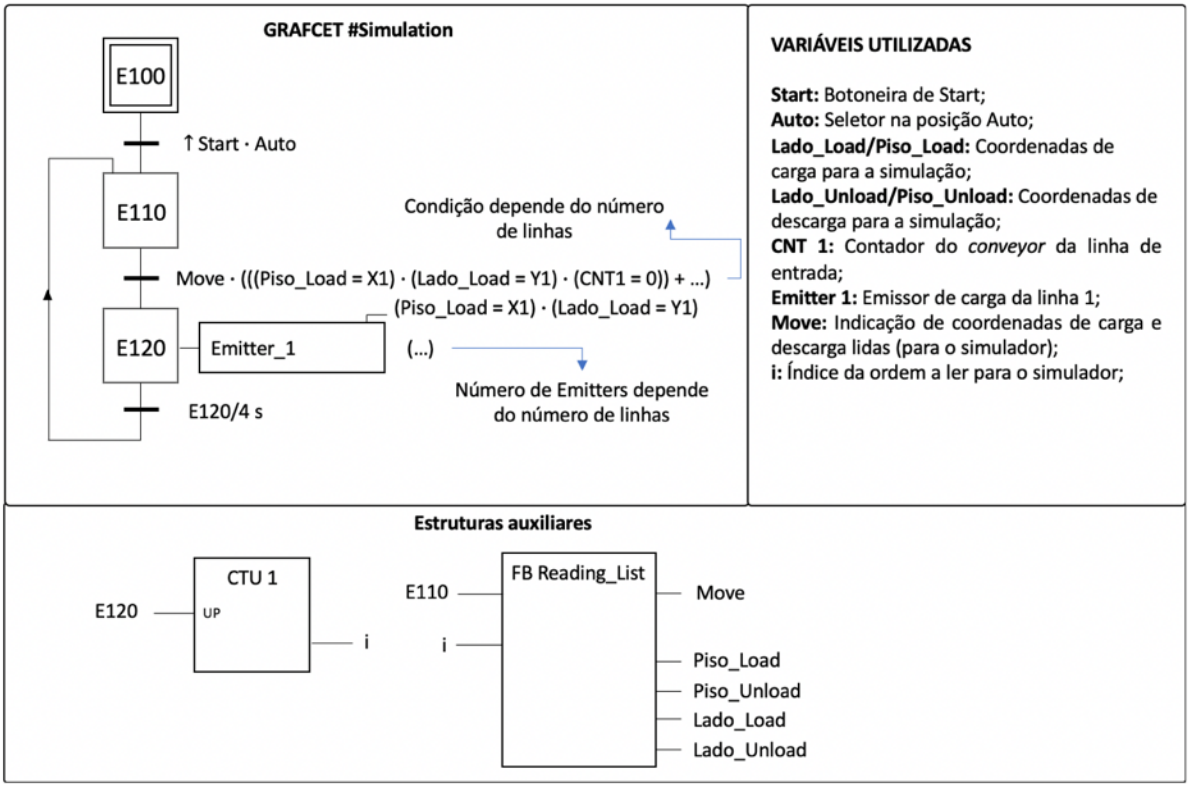


Figura 60 - GRAFCET do *function block* Simulation.