FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Automação de Linha Industrial Flexível para Demonstração

Nuno Fortunato Oliveira

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Orientador: Mário Jorge Rodrigues de Sousa (Prof. Dr.)

4 de Fevereiro de 2013

Resumo

A automação industrial é nos dias de hoje uma área com uma importância fulcral nas diversas unidades industriais espalhadas pelos diferentes sectores económicos. Tendo em conta este aspecto a Faculdade de Engenharia da Universidade do Porto, adquiriu um kit de uma Linha Industrial Flexível para que os alunos possam adquirir competências e conhecimentos nesta área cada vez mais importante no mundo da Engenharia.

Nesta dissertação, é modelada e desenvolvida uma aplicação de controlo para que possa ser utilizada na Faculdade de Engenharia para efeitos de demonstração. Na concepção do modelo utilizou-se ferramentas de modelação abstrata, recorrendo-se à *Unified Modeling Language* para produzir diagramas de estado e de classes que especificam o sistema. O desenvolvimento da aplicação foi baseado na Norma IEC 61131, nomeadamente na sua terceira parte, esta norma define características mecânicas e lógicas que devem ser seguidas pelas unidades lógicas de controlo programáveis, os Autómatos Programáveis.

No desenvolvimento da aplicação, no âmbito da automação industrial, existem inúmeros ambientes de desenvolvimentos lançados pelas diversas marcas fabricantes de autómatos. Estes além de serem pagos, têm inúmeras restrições a nível de plataforma de uso e de compatibilidade entre eles. Devido a estes factores, foi utilizado um *software* criado pela *Lolitech*, o Beremiz. Este produto é open-source, grátis, multiplataforma e segue as restrições da Norma IEC 61131. De modo a avaliar e validar o Beremiz, como um ambiente de desenvolvimento capaz de responder às exigências do mundo da automação industrial, realizaram-se inúmeros testes que avaliaram a sua facilidade de utilização, fiabilidade, robustez e adopção à norma IEC 61131-3.

Abstract

Industrial automation is nowadays an area of central importance in various industrial units spread across different economic sectors. Taking this into account the Faculty of Engineering of University of Porto, purchased a kit of a Flexible Industrial Line for students to acquire skills and knowledge in this increasingly important area in the world of engineering.

In this dissertation, is modeled and developed an application control so it can be used at the Engineering Faculty for demonstration purposes. In designing the model is used abstract modeling tools, resorting to the Unified Modeling Language to produce state diagrams and classes that specify the system. The development of the application was based on the IEC 61131 standard, especially in its third part, this standard defines mechanical and logic characteristics that must be followed by programmable logical units, the Programmable Logic Controllers.

In application development, within the industrial automation area, there are numerous environments developments launched by various brands of PLC manufacturers. These are paid well, have many restrictions in terms of platform use and compatibility between them. Due to these factors, is used a software created by Lolitech, the Beremiz. This product is open-source, free, multi-platform and follows the restrictions of IEC 61131. In order to evaluate and validate the Beremiz as a development environment capable of meeting the demands of the world of industrial automation, there have been done numerous tests that evaluated their ease of use, reliability, robustness and adoption of IEC 61131-3.

Agradecimentos

Em primeiro lugar gostaria de agradecer ao Professor Mário Sousa pela disponibilidade e confiança demonstradas, pelo acompanhamento do trabalho, principalmente nos dias que precederam a entrega do mesmo. O meu sincero agradecimento por tudo.

A todos os meus amigos que me acompanharam neste percurso difícil ao longo destes últimos 5 anos, sem vocês não estaria nesta fase certamente.

Gostaria de agradecer também à minha família, aos meus pais por me terem proporcionado a oportunidade de frequentar este Mestrado e por toda educação e apoio que me dão diariamente ao longo destes 23 anos, ao Pedro, o meu irmão mais novo, por ser o meu melhor amigo.

Por fim, mas com uma especial importância, agradeço à Ana, o teu apoio e ajuda durante todo o projeto será algo que jamais irei esquecer.

A todos vós, o meu Muito Obrigado.

Nuno Fortunato Oliveira

I've missed more than 9000 shots in my career. I've lost almost 300 games.
26 times, I've been trusted to take the game winning shot and missed.
I've failed over and over and over again in my life.
And that is why I succeed.

Michael Jeffrey Jordan

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

LIF	Linha Industrial Flexível
PLC	*Power Line Communication*
TCP	*Transmission Control Protocol*
POU	*Program Organization Units*
FUN	*Function*
FB	*Function Block*
PROG	*Program*
ST	*Structured Text*
LD	*Ladder Diagram*
FBD	*Function Block Diagram*
SFC	*Sequential Function Chart*
RTU	*Remote Terminal Unit*
PDU	*Protocol Data Unit*
IP	*Internet Protocol*
ADU	*Application Data Unit*
UML	*Unified Modeling Language*
MBAP	*Modbus Application Header*

Capítulo 1

Introdução

O trabalho desenvolvido nesta Dissertação tem como objetivos principais o desenvolvimento de um algoritmo de controlo para uma linha de produção flexível, e a avaliação do ambiente de desenvolvimento utilizado, *open-source*, para a automação industrial, o Beremiz.

1.1 Automação Industrial

O termo automação pode ser definido como um processo onde são realizadas diversas operações industriais com o auxílio de diversos dispositivos eletrónicos e/ou mecânicos que controlam os seus próprios processos.

Os sistemas de automação foram evoluindo ao longo dos últimos anos, passando de sistemas baseados em controle automático, mecanizado (como as primeiras linhas de montagem do século XX) até aos sistemas baseados nas atuais tecnologias como a microeletrónica [10]. Tudo isto foi surgindo devido às necessidades sentidas, em busca do aumento da produtividade e consequentemente melhor resultados a nível de produção. Como consequência, as máquinas foram-se tornando cada vez mais modernas, mais precisas e mais completas pondo o trabalho humano, de parte.

A automação industrial surge então como um conjunto de aplicação de técnicas que juntamente com diversos softwares e equipamentos específicos numa determinada máquina executam um processo industrial. São sistemas automatizados caracterizados por altos níveis de precisão e sincronismo. Para se realizar um projeto de produção de máquinas são necessárias várias ciências envolvidas como a Física, Mecânica, Hidráulica e Pneumática. Estas em ligação com modelos mecânicos e matemáticos, maquetes para estudo e visualização e sistemas computacionais dão origem a sistemas industriais automatizados [11].

Com a crescente importância da automação industrial, e com a sua homogeneização em todos os setores da economia, surgem inúmeros desafios com que os responsáveis por estes sistemas se deparam. A segurança e confiabilidade dos sistemas, as comunicações seguras e a identificação de falhas são exemplos desses desafios [10].

1.2 Objetivos e Motivação da Dissertação

Com o crescente aumento da importância da automação no mundo industrial e com a constante preocupação em providenciar aos seus alunos melhores ferramentas para estudo da mesma. A FEUP adquiriu um *kit* de uma Linha Industrial Flexível (LIF), com o objetivo de proporcionar um ambiente similar ao industrial para que os alunos possam aprender com ferramentas o mais próximo possível da realidade. Surgiu deste fator a necessidade de desenvolver uma aplicação de controlo de modo que os novos alunos possam interagir com o *kit* sem que tenham grandes conhecimentos sobre o mesmo, e assim, familiarizar-se-ão mais rapidamente com os equipamentos presentes na faculdade.

A conceção da aplicação de controlo foi baseada na norma IEC 61131, esta norma é o*standard* para o mundo industrial, nomeadamente no campo dos autómatos programáveis. Numa altura em que os fabricantes de autómatos tentam o melhor possível seguir a norma, este trabalho não foi exceção. Assim o estudo detalhado da norma IEC 61131, mais em particular a sua terceira parte, é um fator determinante na boa execução da conceção da aplicação.

Como ambiente de desenvolvimento existem no mercado múltiplas escolhas, sendo elas todas proprietárias e com um custo elevado. Desta forma adotou-se neste trabalho, o uso do Beremiz, *software* grátis, *open-source* e multiplataforma como ambiente de desenvolvimento. Assim do uso do Beremiz surgiu outro objetivo, a avaliação detalhada deste *software* e de todas as suas vantagens e desvantagens. De seguida apresenta-se os objetivos na forma de uma lista para melhor perceção dos mesmos, estando destacados os objetivos principais:

- Estudo detalhado da LIF;
- Estudo detalhado da norma IEC 61131, principalmente da terceira parte;
- **Conceção da arquitetura do programa usando uma ferramenta abstrata**;
- Desenvolvimento da aplicação de acordo com a norma;
- **Testes e avaliação do ambiente de desenvolvimento Beremi**z.

Em suma, este projeto incorpora um amplo leque de tecnologias, todas elas fundamentais na área da automação industrial que está em expansão e com cada vez mais preponderância no ambiente empresarial e industrial sendo sem dúvida uma grande motivação para o decurso deste trabalho.

1.3 Estrutura da Dissertação

A Dissertação é estruturada em cinco capítulos, constituindo o Capítulo 1, uma contextualização ao tema abordado no âmbito deste trabalho, cuja descrição pormenorizada é apresentada nos capítulos subsequentes.

No Capitulo 2 é descrita de forma detalhada a LIF iniciando com uma visão geral sobre a linha flexível, passando posteriormente para a descrição pormenorizada de cada um dos módulos, armazém, maquinação série e paralela e zona de carga e descarga. Aquando da descrição dos módulos é descrito também o funcionamento de cada tipo de células presentes na LIF, tapetes, máquinas e pushers.

O Capítulo 3, tem como título principal "Tecnologias Presentes"onde são descritas as tecnologias presentes na conceção e desenvolvimento do trabalho. Incia-se uam apresentação da norma IEC 61131, de seguida expôem-se as caractrísticas do ambiente de desenvolvimento utilizado, Beremiz. Por último o protocolo de comunicação entre a aplicação de controlo e a LIF, protocolo *Modbus*.

No Capítulo 4, é descrito o desenvolvimento da aplicação de controlo desde do seu início, arquitetura até à última fase de testes e validação. Está também presente uma secção dedicada aos testes, problemas e resolução dos mesmo no ambiente de desenvolvimento, Beremiz.

O Capítulo 5 encerra a Dissertação apresentando as principais conclusões do trabalho desenvolvido. As limitações do trabalho desenvolvido são ainda revistas e analisadas criticamente. Mais, são propostos possíveis progressos e trabalhos futuros em consequência do trabalho desenvolvido.

Capítulo 2

Linha Industrial Flexível

Este capítulo apresenta inicialmente uma visão geral da LIF que serve de base a esta tese, para posteriormente ser descrito pormenorizadamente cada subsistema da mesma.

2.1 Visão Geral

A LIF é composta por diferentes subsistemas com funções distintas entre eles. Os subsistemas presentes são:

- Módulos da linha de montagem – Conjunto de células, tapetes, máquinas e *pushers*;
- Ilhas de *inputs*/*outputs*;
- Botão de emergência e respetivo sinalizador;
- Circuito de alimentação;

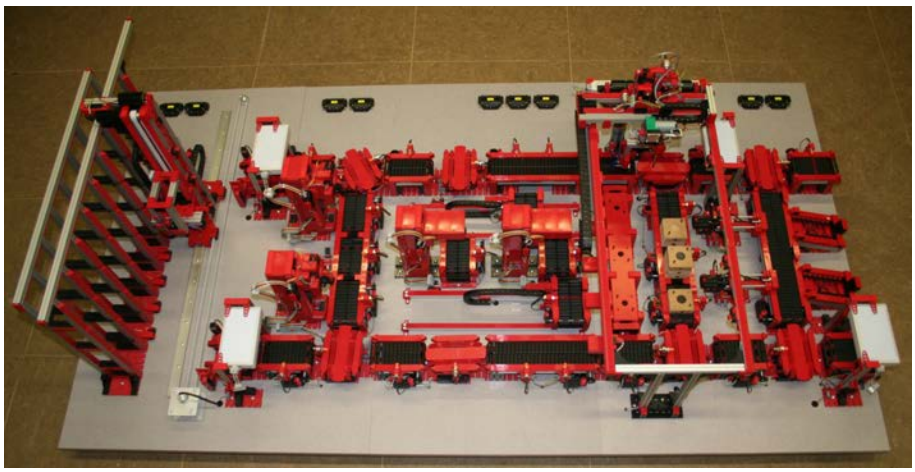

Figura 2.1: Linha Industrial Flexível [1].

A LIF, figura 2.1 tem dois modos de funcionamento : modo local e modo remoto. No modo local esta é controlada através de ligações diretas até ao *Program Logic Controller* (PLC) presentes

no laboratório. No modo remoto a linha é controlada através da rede usando o protocolo *Modbus Transmission Control Protocol*(TCP), abordado mais à frente no Capítulo 3. Na parte inferior da LIF, estão ligadas um grupo de ilhas, às quais estão conectados todos os sensores e atuadores. Cada uma é composta por uma interface *ethernet* e várias cartas de entradas e saídas digitais.

Outro módulo que não se encontra visível na foto, é um sistema, baseado em *Linux*, que é responsável pelo controlo de comunicação entre a linha e a aplicação de controlo da mesma. Este sistema tem implementado um sistema de *interlock* que não permite que ordens contraditórias sejam passadas à LIF, para que os seus componentes não sejam danificados por utilizações indevidas. Como exemplo, se é dada a ordem de movimentar o mesmo tapete em direções opostas, este sistema vai bloquear esta ordem de modo a não danificar o motor do presente no tapete. Apesar de ser um sistema de aprendizagem e de demonstração, existe um botão de emergência capaz de parar todos os processos que estão a decorrer na linha de montagem. Por fim, o sistema luminioso que indica o estado em que se encontra a LIF:

- Verde – o sistema está iniciado e pronto para ser usado;
- Laranja – dispara sempre que surge um aviso de *interlock*;
- Azul – caso o *interlock* seja ativado, esta luz acende-se e permanece neste estado durante uns segundos para que o utilizador possa se aperceber do aviso, isto se o erro tenha ocorrido durante breves instantes e o utlizador não tenha reparado na luz laranja acesa;
- Vermelha – Sempre que o botão de emergência é ativado.

Nas seguintes secções serão descritos os módulos da LIF. Não será descrito o circuito de alimentação da linha de montagem, visto que para este trabalho considerou-se que não tinha relevância.

2.2 Módulos

A LIF foi criada e desenvolvida pela marca *STAUDINGER* sendo projetada e desenhada de acordo com os requisitos exigidos pela FEUP, encontra-se dividida em 4 módulos:

- Armazém
- Zona de montagem
- Zonas de maquinação
- Zonas de carga e escarga

Decidiu-se, neste projeto, ignorar a zona de montagem, visto que do ponto de vista mecânico esta não funciona convenientemente.

2.2.1 Armazém

O armazém é composto por dois tapetes lineares que trabalham como estações de entradas e saída de peças e por um empilhador. O armazém tem a capacidade para 24 peças espalhadas ao longo de 3 linhas e 8 colunas. Cada posição é munida de um sensor de modo a saber qual o estado de ocupação.

A zona de carga e descarga de peças é realizada por 2 tapetes lineares, no entanto, estes tapetes possuem características diferentes dos restantes tapetes da linha de montagem. Estes, possuem um sensor óptico do tipo *Radio-Frequency IDentification*(RFID), responsável pela deteção do material da peça a armazenar, uma zona especial onde o empilhador encaixa para carga ou descarga da peça pretendida.

De modo a armazenar as peças o empilhador move-se ao longo de três eixos cartesianos. O movimento é realizado sempre a velocidade constante, permitindo assim o uso de saídas discretas para o movimento em cada eixo. No entanto no eixo X, o empilhador possui duas velocidades, de modo a que antes de chegar à posição desejada, diminua a velocidade para que consiga parar na posição correta. No eixo Y, o movimento é linear e para cada coluna existe um sensor. Comportamento semelhante seria de esperar para o eixo Z, mas tal não acontece, pois para cada posição no eixo Z existem dois sensores, um superior e outro inferior. Assim para colocar o empilhador na posição correta, este é inicialmente colocado na posição superior e por fim desce até à posição inferior, estando assim pronto para armazenar a peça pretendida.

Figura 2.2: Armazém da LIF [2].

2.2.2 Maquinação Série e Paralela

A zona de maquinação série é composta por três tapetes lineares, dos quais dois estão acoplados às respetivas máquinas ferramenta. Por sua vez, a maquinação paralela é composta por dois tapetes lineares e respetivas máquinas ferramenta existindo ainda dois tapetes deslizantes responsáveis pelo transporte das peças até à zona de maquinação.

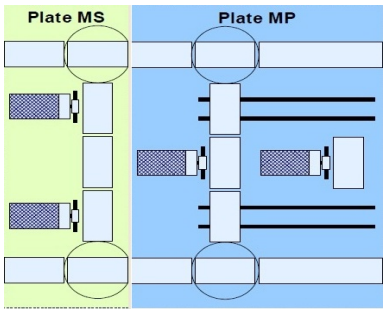

Figura 2.3: Esquema representativo da Maquinação Série e Maquinação Paralela [1].

Os tapetes lineares presentes nestes módulos têm dois sentidos de movimento e um sensor indutivo de modo a detetar a presença de peça. O facto de ter um só sensor no centro de cada tapete limita o número de peças que pode ser movimentada em cada tapete, por isso, apenas uma peça pode ser movimentada de cada vez por cada tapete.

Os tapetes deslizantes, figura 2.4, presentes somente na maquinação paralela possuem as mesmas capacidades dos tapetes lineares, ou seja, o movimento em dois sentidos. Além disso estes tapetes têm da capacidade de se movimentarem noutra posição. Imaginando que a capacidade de movimentação semelhante aos tapetes lineares é realizada no eixo X, estes tapetes possuem a capacidade de se moverem no eixo Y. Esta capacidade permite que sejam estes tapetes a abastecer de peças as duas máquinas que se encontram dispostas paralelamente.

Figura 2.4: Tapete Deslizante.

Por fim, existem as máquinas ferramenta, figura 2.5. Estas máquinas estão, como referido anteriormente, acopoladas a um tapete linear. Quando detetam a presença de peça, é escolhida uma das três ferramentas rotativas presentes em cada máquina até que o sensor de ferramenta seja ativo e, depois, por um tempo pré determinado pelo utilizador efectua-se a maquinação. Todas as operações nas máquinas tal como no armazém e nos restantes módulos, são realizadas a uma velocidade contínua para que a utilização de atuadores discretos seja possível.

Figura 2.5: Máquina Ferramenta [2].

2.2.3 Zona de Carga e Descarga

A zona de carga e de descarga é composta por tapetes lineares e tapetes rotativos. A zona de carga é composta por oito tapetes lineares e dois tapetes rotativos, cada um destes responsáveis pela introdução de peças nas respetivas zonas de maquinação. A zona de descarga, figura 2.6 além dos componentes existentes na zona de carga, possui na sua constituição dois *pushers*, reponsáveis pela descarga de peças para o exterior da LIF.

Figura 2.6: Zona de Descarga.

Os tapetes rotativos, figura 2.7, tal como os deslizantes, desempenham as mesmas funções dE um tapete linear, acrescentando a característica de rodarem sobre si próprios. A rotação máxima que podem realizar é de 90 graus até atingirem o sensor que indica o final de rotação existente nos dois sentidos de rotação.

Figura 2.7: Tapete Rotativo.

Os *pushers*, figura 2.8, presentes na zona de descarga são usados para "expulsar" as peças da LIF, tendo estes a capacidade de empurrar as peças através de um braço mecânico, para um suporte com capacidade para duas peças. O braço mecânico é composto por dois atuadores: um de retração e outro de extensão do mesmo sendo controlados por dois sensores de final posição. Para transportar as peças até à posição para expulsão por parte do *pusher*, existe um tapete linear, que executa esta tarefa.

Figura 2.8: Pushers.

Capítulo 3

Tecnologias Presentes

Este capítulo irá abordar as tecnologias incorporadas no projeto. Por um lado, a norma IEC 61131, enquanto modelo usado na programação da aplicação de controlo, como também o Beremiz, utilizado enquanto ambiente de desenvolvimento. Por outro lado, protocolo *Modbus* implementado em TCP, responsável pela comunicação entre a aplicação de controlo e a LIF também será alvo de estudo.

3.1 Norma IEC 61131-3

3.1.1 Visão Geral

A norma IEC 61131 criada em 1982 por um comité internacional, tem como intuito criar uma interface comum ao uso dos autómatos programáveis. Está dividida em cinco partes, sendo a terceira parte a que aborda o aspeto da programação de controladores industriais e define o modelo de programação. Os principais conceitos e características apresentados pela IEC 61131-3 [4] são:

- Base de dados com declaração de variáveis e alocação dinâmica;
- Tipos de dados;
- Estruturação, modularização, reutilização e portabilidade de software;
- Orientação a objetos;
- Processamento multitarefa;
- cinco linguagens de programação.

Embora a norma defina um conjunto de regras nas quais os fabricantes de PLC deviam aderir, tal não acontece de uma forma rígida tendo os fabricantes bastante liberdade em implementar componentes que pretendam, mesmo quando estes componentes não seguem a norma.

Nos últimos anos e com a crescente necessidade de uniformizar o mercado dos PLC, a aceitação da norma tem aumentado, trazendo assim inúmeras vantagens aos fabricantes mas principalmente ao consumidor final. Desta forma, é possível normalizar a estrutura funcional dos equipamentos, criar um ambiente de desenvolvimento independente do equipamento ou fabricante, que vai, por sua vez, simplificar o fabrico de equipamentos e reduzir os custos associados [3].

3.1.2 *Program Organization Units*

Os *Program Organization Units* (POU), sãs as unidades básicas de uma aplicação de controlo. Estes podem ser de três tipos diferentes: *Function* (FUN), *Function Block* (FB) ou *Programs* (PROG). Segundo a norma o POU é composto por cabeçalho e corpo estando as variáveis presentes no cabeçalho e as instruções de execução no corpo, como ilustra a figura 3.1.

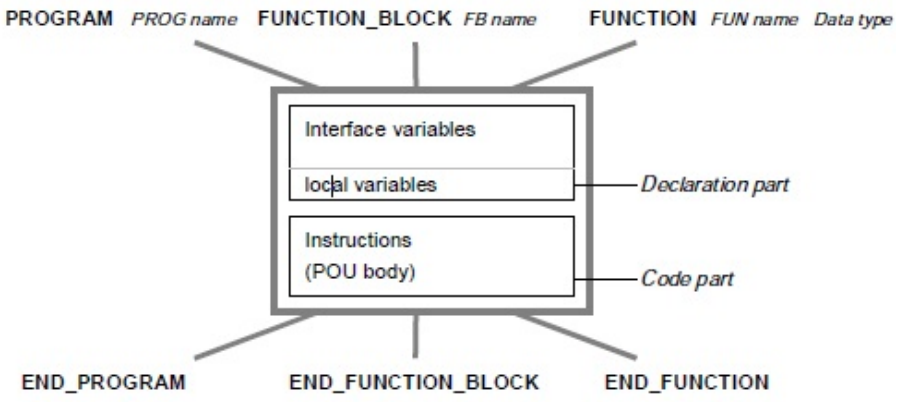

Figura 3.1: Estrutura detalhada de um POU [3].

Os três diferentes tipo de POU referidos anteriormente formam uma hierarquia entre si. Os PROG estão no nível mais elevado desta cadeia podendo incluir nas suas intruções FB e FUN, no entanto são os únicos que não podem incluir no seu corpo uma instância de outro PROG. No segundo nível da hierarquia aparecem os FB. Estes podem incluir no seu corpo FUN e FB, ao contrário dos PROG que não permitem instâncias do mesmo tipo de POU. Por útlimo estão as FUN que só podem incluir instâncias delas próprias, além disso as FUN diferem dos restantes tipos de POU porque não possuem memória, ou seja, o estado passado não é tido em conta na presente execução da mesma.

Para além dos tipos de POU já apresentados, a norma define alguns modelos de FUN e FB, os *Basic Building Blocks*. Estes são POU pré-definidos capazes de realizar funções de conversão, numéricas, entre outras.

3.1.3 Tipos de Dados e Variáveis

3.1.3.1 Tipos de Dados

A norma IEC 61131-3 define vários tipos de dados tais como se apresenta na tabela 3.1.

Tabela 3.1: Tipos de Dados segundo a norma IEC 61131-3 [4].

Data Type	Size (bits)	Default Number
BOOL	1	False
BYTE	8	0
WORD	16	0
DWORD	32	0
LWORD	64	0
SINT/UINT	8	0
IN/UINT	16	0
DINT/UDINT	32	0
LINT/ULINT	64	0
REAL	32	0
LREAL	64	0
STRING	8 (por caracter)	string vazia
WSTRING	16 (por caracter)	string vazia
TIME	Implementação Dependente	T#0S
TIME_OF_DAY	Implementação Dependente	TOD# 00:00:00
DATE	Implementação Dependente	D# 0001-01-01
DATE_AND_TIME	Implementação Dependente	DT# 0001-01-01-00:00:00

Além deste tipo de dados a norma permite a criação de novos tipos de dados derivados dos tipos de dados apresentados anteriormente. A criação de novos tipos de dados pode acontecer de diferentes formas [3]

- Direta – Cria um tipo de dado elementar com um particular valor inicial;
- Com intervalo – Cria um tipo de dado com um determinado limite. Por defeito, assume o valor mais baixo do intervalo;
- Enumeração – O tipo de dado pode assumir um valor de uma determinada lista;
- *Array* – Vários elementos do mesmo tipo de dado combinam e formam um *array*;
- Estrutura – Vários elementos combinam e formam um tipo de dados.

Desta forma pode-se definir qualquer tipo de dado pretendido pelo utilizador. Os tipos de dados são fundamentais, visto que estão ligados a todas as variáveis do projeto. Ou seja, todas as variáveis estão definidas com um tipo de dados. Esta caracteristica confere robustez ao programa, evitando assim, a realização de operações inválidas como dividir uma data por uma *string*.

3.1.3.2 Variáveis

A norma especifica também difere tipos de variáveis, interna, entrada, saída e entrada/saída. As variáveis internas são utilizadas num POU mas que o seu valor é irrelevante para o restante programa. As de entrada são alimentadas externamente por outras variáveis presentes noutro POU, acontecendo o contrário com as variáveis de sáida, isto é, estas alimentam outras variáveis de outros POU. A declaração da variável é efetuada tendo em conta os seguinte atributos [3]:

- Nome;
- Tipo de variável;
- Tipo de dado;
- Valor inicial;
- Atributos.

Quando são usadas variáveis remotas, como por exemplo, entradas e saídas de um PLC a sua definição é realizada recorrendo à expressão *AT* e concatenando os seguintes parâmetros:

"% + localização + tamanho + um ou mais inteiros separados por '.' [3]"

A norma refere ainda que quando a representação inclui os inteiros separados por pontos finais, estes devem ser interpretados como um endereçamento numa lógica hierárquica, com o inteiro mais à esquerda a representar o mais alto nível e decrescendo da esquerda para a direita.

Na tabela em baixo, pode verificar os Prefixos para a localização e tamanho das *directly represented variables*:

Tabela 3.2: Prefixo para a localização e tamanho das *directly represented variables* [8].

Type	Prefix	Meaning
Location	I	Input
	Q	Output
	M	Memory/Flag
Length	X or none	1 bit
	B	8 bits
	W	16 bits
	D	32 bits
	L	64 bits

3.1.4 Linguagens de Programação

Uma das principais vantagens da adoção da IEC 61131-3 é a normalização das linguagens de programação para PLC's. A norma define cinco linguagens:

- Structure Text (ST);
- Instruction List (IL);
- Ladder Diagram (LD);
- Function Block Diagram (FBD);
- Sequential Function Chart (SFC).

Destas linguagens, duas são textuais, ST e IL, e as restantes são gráficas. Estas linguagens não possuem qualquer tipo de restrição ao uso de variáveis ou tipos de dados, além disso as diferentes linguagens interagem entre elas sem qualquer conflito.

O ST é uma linguagem que consiste numa lista de procedimentos, figura 3.2. Cada procedimento é processado determinando valores que controlam o fluxo do programa. É uma linguagem de alto nível orientada a máquinas apresentando uma vasta gama de procedimentos abstratos e com grande complexidade num modo bastante compresso [3].

Tem como vantagens a sua forma bastante compacta de programação, as suas instruções claras e em blocos, e uma potente construção do fluxo do programa. Apesar disto apresenta algumas desvantagens como a sua tradução para código de máquinas não ser direta visto que é processada através dum compilador e sendo assim, e o seu alto grau de abstração pode trazer alguma falta de eficiência [3].

```
fedge := fall AND (old_signal AND NOT signal);
redge := rise AND (signal AND NOT old_signal);
edge := fedge OR redge;
old_signal := signal;
```

Figura 3.2: Excerto de código ST [4].

O IL é uma linguagem de baixo nível, semelhante ao *assembly* usado na programação de microcontroladores. É muitas vezes usada em controlo de processos simples, bem como na da tradução de linguagens gráficas para textuais.

```
LD fall
AND old_signal
ANDN signal
OR (
LD rise
AND signal
ANDN old_signal
)
```

Figura 3.3: Excerto de código IL [4].

O LD provém do campo dos sistemas de eletrónicos e representa o fluxo de energia da esquerda para a direita dum POU. È uma linguagem histórica [4] visto que era utilizada nos primeiros PLC equipados principalmente com *relays*, razão pela qual os seus símbolos são idênticos a circuitos elétricos.

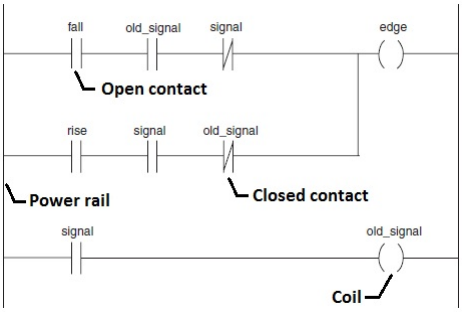

Figura 3.4: Excerto de código LD [4].

O FBD surgiu da área do processamento de sinal e descreve as interações entre os sinais de entrada e os sinais de saída que são processados através dos diferentes blocos. A utilização deste sistema permite que seja uma linguagem modular. As entradas de um bloco só podem ser conectadas às saídas de outro bloco [4]. Além disso, não é possível conectar variáveis com diferentes tipos de dados. É uma linguagem poderosa e versátil e que tem vindo a chamar à atenção do mundo da automação industrial, prova disso é a norma que foi criada especialmente para estas linguagem de forma a incluir novas instruções, norma IEC 61499.

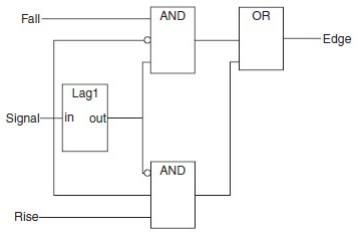

Figura 3.5: Excerto de código FBD [4].

O SFC é uma linguagem gráfica que permite ações sequencias paralelas e alternativas num programa. Este fornece os meios para estruturar a organização do programa num conjunto de etapas separadas pro transições. Cada transição está associada uma expressão que tem de ser satisfeita para que o programa continue o seu fluxo [3].

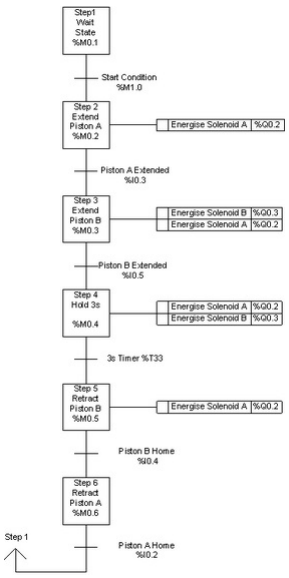

Figura 3.6: Excerto de código SFC [5].

3.2 Beremiz

3.2.1 Visão Geral

O Beremiz é um ambiente de desenvolvimento integrado, com a finalidade de desenvolver programas para PLC. Além de ser grátis, multiplataforma e *open-source* este segue a norma IEC 61131-3, mencionada no capítulo anterior. A necessidade da criação do Beremiz teve como base diferentes fatores [6] [12]:

- Falta de soluções *open source* nesta área;
- A dificuldade na portabilidade de programnas desenvolvidos entre diferentes plataformas, apesar dos esforços de normalização;
- Os elevados custos de aquisição das licenças dos diversos fabricantes, dificultando assim o processo de aprendizagem;
- A dificuldade de acesso aos compiladores e *runtime* impede a exploração dos mesmo e que se desenvolvam modos de operação pretendidos pelo utilizador.

O Beremiz foi desenvolvido de uma forma modular, tendo como componentes principais:

- *PLC Builder GUI*–Visão geral dos projetos;
- *PLCOpen Editor* – Editor de programas;
- *MATIEC*–Compilador;
- *Plugins*–permitem interação com diversos tipos de tecnologias.

Nesta última versão do programa os componentes *PLC builder GUI* e o *PLCOpen Editor* estão misturados numa só interface, logo serão apresentados em conjunto

3.2.2 *PLC builder GUI e PLCOPen Editor*

Na versão atual do Beremiz, versão 1.1 *Release Candidate*, o *PLC Builder GUI* surge como uma interface separada do *PLCOpen Editor* deixou de existir, estes dois componentes estão agora integrados numa só interface de modo a facilitar a interação com os mesmos, ou seja, a obtenção da informação seja mais rápida e intuitiva.

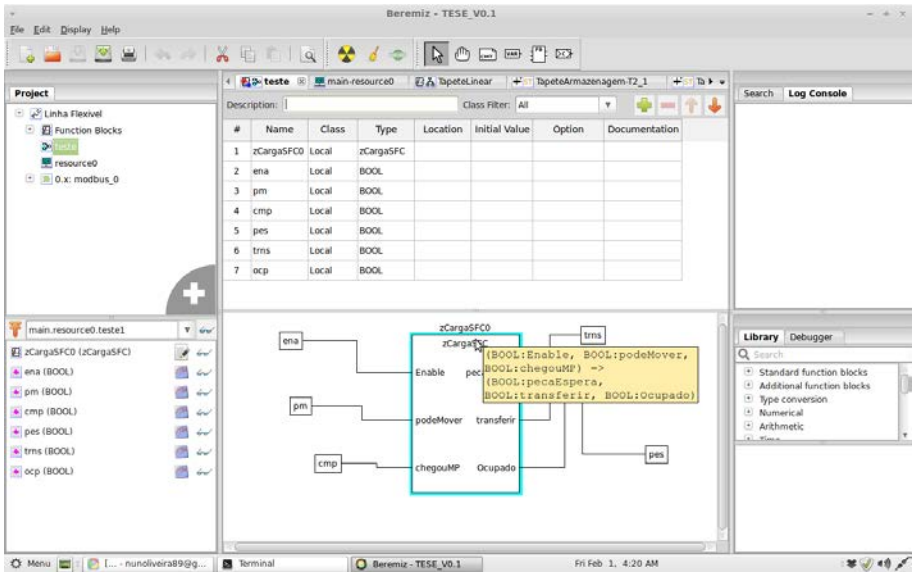

Figura 3.7: Interface Gráfica do Beremiz.

A figura 3.7 apresenta a interface gráfica do Beremiz. Todas as divisões são ajustadas facilmente conforme a necessidade do utilizador. A divisão mais à esquerda está subdividida em duas, na parte superior encontram-se árvores com tipos de dados definidos pelo utilizador, as POU e ainda as configurações, recursos e tarefas. Encontra-se também um botão (“mais”) que apresenta um menu de contexto onde se pode acrescentar desde tipos de dados a algum *plugin* que esteja disponível.

Na parte inferior estão presentes as instâncias presentes no projeto. Esta divisão apresenta uma visão global de todo o projeto que vai desde os tipos de dados até aos *plugins* existentes. Na divisão central é possível a escrita das intruções que cada POU deve executar. Como o Beremiz

segue a norma IEC 61131-3, esta escrita é feita de forma textual, utilizando por exemplo ST ou recorrendo a uma linguagem gráfica como o SFC. Na parte superior desta divisão, é possível adicionar ou eliminar variáveis associadas ao POU ativo. Encontra-se também uma lista de todas as variáveis já criadas.

Na divisão mais à direita está presente uma biblioteca de funções padrão ou já definidas pelo utilizador que podem ser utilizadas a qualquer momento, bastando para isso que a função seja arrastada para a zona de escrita. Por fim na divisão inferior enconta-se uma *Log console* que contém diversas informações quanto à compilação e execução do programa. Os projetos são guardados em formato *XML*, seguindo a estrutura definida no *XML Official Schema* do comité técnico TC6 da organização *PLCOpen*.

3.2.3 Compilador Matiec

O compilador MATIEC tem como ação principal consumir o resultado da conversão textual de um determinado projeto IEC 61131-3. Desta forma, o código realizado em C é organizado com a representação ilustrada na figura 3.8.

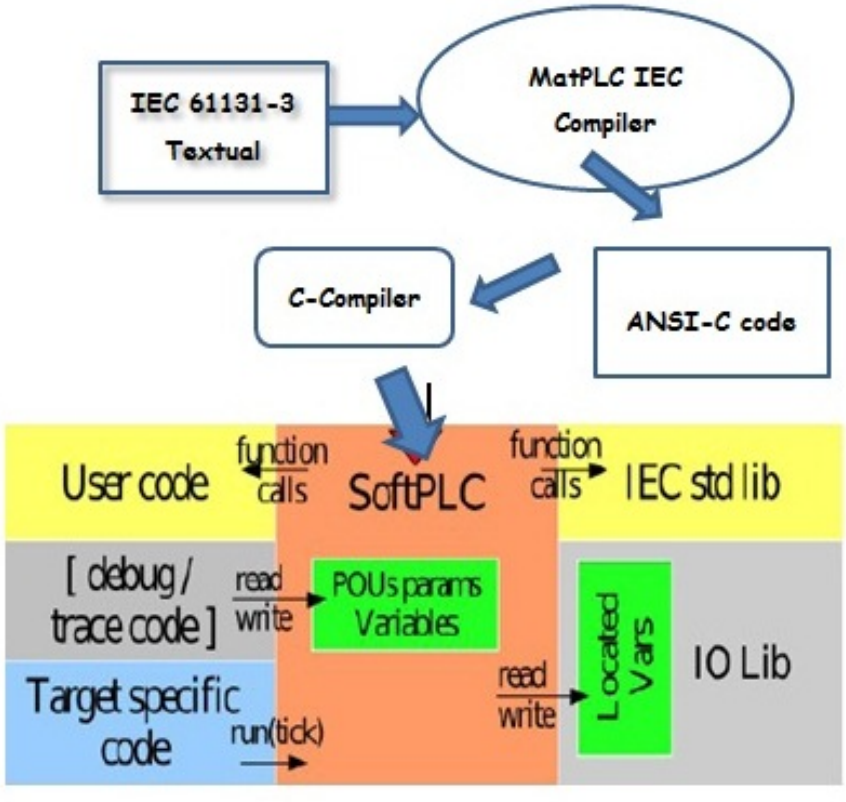

Figura 3.8: Etapas gerais de compilação e organização do código [6].

O compilador funciona em quatro etapas principais [13]:

- *lexical analyzer*

- *syntax parser*
- *semantics analyzer*
- *code generator*

Através da figura, pode-se verificar que tanto as variáveis como os parâmetros dos POU são declarados como estruturas C em árvore, enquanto as variáveis onde se define uma localização são declaradas como externas. O algoritmo de controlo do *SoftPLC* é executado por iniciativa de um módulo próprio (*Target Specific Code*), responsável por gerir o relógio específico da plataforma alvo e gerar interrupções.

O programa também é capaz de aceder a um conjunto de Funções e FB, sejam eles da biblioteca padrão (*IEC std lib*) ou criados pelo utilizador (*User Code*), que se encontram definidos num módulo próprio e que recebem como parâmetros as estruturas C [8].

Dispõe ainda de um outro módulo, cujo objetivo é o de consumir o estado atual de todos os parâmetros do programa, reproduzindo esse estado de funcionamento ao utilizador em *runtime*, através do *PLCOpen Editor* [8].

Importa salientar que o Beremiz apresenta uma restrição face ao disposto na norma IEC 61131. Esta refere no seu modelo de programação que uma dada configuração contém um ou mais recursos cada um dos quais contém um ou mais programas executados sob controlo de zero ou mais tarefas. No Beremiz, é possível instanciar um único recurso e um único programa sob controlo de uma tarefa, sendo que esse programa pode instanciar outras funções.

3.2.4 *Plugins*

Os *plugins* permitem de uma forma generalizada, que exista comunicação entre o *SoftPLC* com o mundo exterior. Assim sendo, existem ações de controlo que são necessárias para a interligação com diversos dispositivos (físicos e lógicos) que podem ter um contacto direto com o processo que se pretender controlar. Estes poderão ser de vários tipos como: sensores, atuadores, interfaces entre outros.

Além destes *plugins* serem compostos por uma interface com o utilizador como também por uma componente de código em C, é importante referir que o uso destas está definido através de uma simbologia própria para a associação das variáveis aos tais dispositivos na estrutura de entrada, saída ou até memória do PLC. Isto consoante o referido na norma IEC 61131-3 sobre as *directly represented variables*.

3.3 Modbus

O protocolo Modbus de comunicação desenvolvida pela Modicon em 1979, usada para estabelecer comunicação entre os dispositivos cliente-servidor/ “escravo- mestre”. Em termos simples, é

um método utilizado para a transmissão de informações através de linhas em série entre dispositivos eletrónicos. Estes dispositivos comunicam utilizando um princípio, onde o cliente pode iniciar as transações e os servidores respondem de acordo com o pedido ou com a tarefa em questão [14].

Trata-se então de um protocolo de comunicação aberto, com um domínio muito concentrado na automação industrial.

3.3.1 Visão Geral

Tendo em vista que o Modbus é um protocolo aberto, de alguma simplicidade e gratuito para os seus fabricantes, este tornou-se no meio de comunicação mais comum e mais utilizado na indústria sendo assim a forma mais utilizada para conexão de dispositivos eletrónicos. Este protocolo é muito utilizado para conectar, por exemplo, um computador de supervisão com uma unidade terminal remota (RTU) no controlo de supervisão e aquisição de dados SCADA. Este protocolo, como já referido acima, estabelece uma comunicação entre um cliente e um servidor, não havendo precedentes com históricos de trocas de informação anteriores.

O padrão *Modbus* emprega os dois modos de transmissão:

- *ASCII Mode*–Caracteres codificados em 7 bits + 1 bit de paridade;
- *RTU Mode*–Caracteres codificados com 8 bits + 1 bit de paridade.

Nas informações transmitidas denominadas por transações, os dados são enviados como uma série de zeros ou outros, chamados bits. Cada bit é então enviado como uma tensão positiva ou com uma tensão negativa. A velocidade de transição ronda os 9600 bits/segundo.

Tanto os pedidos como as respostas baseiam-se em tramas simples, denominados por *Protocol Data Unit*(PDU). Existem assim, três tipos de PDU [9]:

- *Request* PDU
  - um código que indica uma função (1 byte);
  - dados correspondentes à função (tamanho variável);
- *Response* PDU
  - código da função correspondente ao pedido (1 byte);
  - dados correspondentes à resposta (tamanho variável);
- *Exception Response* PDU
  - código da função correspondente ao pedido + 0x80 (128) (1 byte);
  - um código identificador do tipo de exceção (1 byte).

É importante referir que este protocolo é definido por vários tipos de dados, representados na tabela em baixo:

Tabela 3.3: Tipos de Dados usados em Modbus [9]

Nome	Tipo	Acesso
Entrada Discreta	1 bit	somente leitura
Saída Discreta	1 bit	leitura/escrita
Entrada de Registos	16 bits palavra	somente leitura
Saída de Registos	16 bit palavra	leitura/escrita

3.3.2 Serviços

Este protocolo define um conjunto de funções, cada uma das quais com o seu código único. Estes códigos encontram-se no intervalo [1-127], sendo que o intervalo [129-255] está reservado para os códigos de exceção na resposta. São definidas também três categorias de códigos de funções [9]:

- *Public* – São garantidamente únicos, e estão associados a funções bem definidas e documentadas;
- *User Defined* – A especificação define os intervalos [65-72] e [100-110] como códigos de funções para implementação livre por parte dos utilizadores;
- *Reserved* – Estes códigos são usados por alguns fabricantes para soluções proprietárias e não estão disponíveis para uso público.

É também importante referir que uma determinada função deste protocolo é definida essencialmente por descrição, um código associado como também um formato de cada três PDU (*Request, Response e Exception Response*).

3.3.3 Implementação *TCP/IP*

Modbus TCP/IP (*Transmission Control Protocol/Internet Protocol*) é simplesmente o protocolo Modbus RTU com uma interface TCP que funciona em *Ethernet*. Este surgiu com o objetivo de aumentar a velocidade do protocolo Modbus bem como incrementar a versatilidade deste. Modbus TCP/IP acrescenta à simplicidade do Modbus as vantagens do TCP/IP sobre *ethernet*. De uma forma mais simples, pode-se dizer que Modbus TCP/IP permite encapsular as tramas Modbus na *Ethernet*.

O protocolo Modbus conhece diversas implementações, sendo que as mais populares trabalham sobre TCP/IP e sobre transmissões série assíncronas (onde os meios físicos mais comuns são *EIA/TIA-232* e *EIA/TIA-485*). Define um PDU independente do meio físico a utilizar. Assim, a construção da mensagem do protocolo Modbus num determinado meio físico é feita com a introdução de campos adicionais ao PDU. O cliente que inicia a comunicação constrói a PDU ao qual adiciona os campos necessários à transmissão da mensagem no meio em questão.

O Modbus/TCP acresce uma verificação de erros na *frame* a cargo dos protocolos TCP/IP e *ethernet* como também um cabeçalho específico, formando assim uma trama própria da implementação, denominada de *Application Data Unit (ADU)* [8], como mostra a figura seguinte:

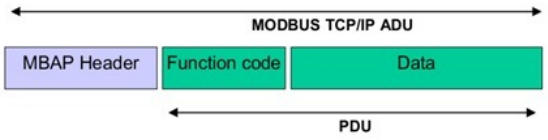

Figura 2.6: Application Data Unit do Modbus/TCP (Fonte: [3])

Figura 3.9: *Application Data Unit (ADU)* [7].

Este cabeçalho, denominado de *Modbus Application Header* (MBAP) na especificação, tem 7 bytes de comprimento, e é composto por [7]:

- *T*ransaction identifier (2 bytes)– usado para identificação unívoca das mensagens (pedidos e respostas), pois é copiado pelo servidor na resposta a um determinado pedido;
- *P*rotocol identifier (2 bytes) – tem o valor zero por defeito para o Modbus, existindo essencialmente na expectativa de expansões futuras;
- *l*enght (2 bytes)– contém o número de bits que se seguem na trama, por forma a ajudar na deteção dos limites da mesma;
- *U*nit identifier (1 bytes)– usado para identificação de um dispositivo remoto localizado numa rede distinta da rede TCP/IP, sendo o elemento que é usado pelas *gateways* para direcionar um pedido que lhe seja endereçado.

No Modbus TCP/IP qualquer equipamento pode ser cliente e servidor simultaneamente. Esta particularidade é importante e vista como uma grande vantagem desta implementação face a todas as outras.

Na configuração *ethernet* com a utilização de *bridges*, *routers* e *gateways* é possível, no mesmo sistema, ter equipamentos ligados em *ethernet* e outros ligados em linha série. Desta forma, qualquer servidor ligado na rede *ethernet* pode comunicar com outro equipamento ligado em qualquer local do sistema, se este estiver ligado na *ethernet* ou ligado num barramento série que por sua vez estará ligado à *ethernet* através de uma *gateway*.

Capítulo 4

Desenvolvimento

Este capítulo apresenta inicialmente a arquitetura adoptada para o desenvolvimento da aplicação de controlo. De seguida, descreve-se a aplicação com pormenor e por fim uma avaliação do ambiente de desenvolvimento utilizado, Beremiz.

4.1 Arquitetura do Sistema

Um dos objetivos deste trabalho é a modelação e desenvolvimento duma aplicação de controlo para a linha flexível apresentada anteriormente. Esta aplicação será usada para sessões de demonstração da mesma. Como tal, tentou-se reduzir a complexidade da aplicação para que em termos demonstrativos seja fácil de observar cada um dos seus módulos em funcionamento. Tendo em conta estes fatores definiu-se 3 serviços essenciais para demonstração:

- Maquinação de peças, nas duas zonas de maquinação paralela e série;
- Carga e Descarga de peças para o armazém;
- Carga e Descarga de peças no sistema.

De modo a aproximar o sistema com a realidade foi definido que apenas existe um tipo de peças, ou seja, apenas uma matéria-prima. Desta peça inicial é possível realizar qualquer tipo de operação, visto que, é o utilizador que comanda o tempo de maquinação em cada máquina. Logo todas as peças criadas serão diferentes consoante os tempos de maquinação das mesmas. De seguida, e para garantir o bom funcionamento da LIF foi elaborado um esquema com o fluxo de movimento das peças. Este ilustra quais os caminhos que as peças executaram desde o momento de entrada do sistema até à sua saída. Este processo pretende dar simplicidade e robustez ao mesmo.

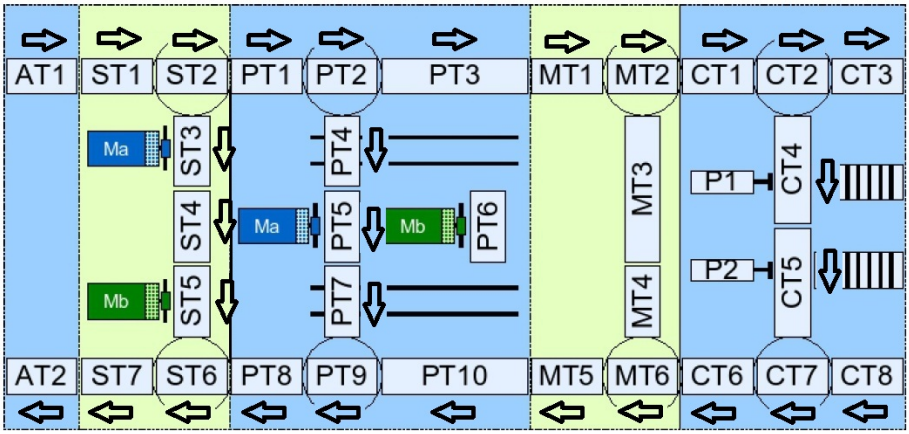

Figura 4.1: Fluxo do movimento das peças pela LIF.

Através da figura 4.1, verifica-se que na parte superior do sistema as peças são movimentas sempre da esquerda para a direita sendo este o sentido da descarga de peças. Na parte inferior acontece exatamente o oposto. As peças movimentam-se da direita para a esquerda, sentido da carga de peças. No interior da linha as peças movimentam-se de cima para baixo, onde são efetuados os processos de maquinação. Assim que foi descrito o fluxo, foi necessário descrever o funcionamento da LIF desde que o sistema é iniciado até que é desligado.

Quando o programa é iniciado existem dez peças presentes no armazém. Estas estão prontas para qualquer pedido que o utilizador decida efetuar. Para que o sistema efetue a operação de armazenagem basta para isso, colocar peças, que são consideradas matéria-prima no tapete inicial da zona de carga (canto inferior direito). Assim que o tapete deteta presença nessa peça, começa de imediato e automaticamente a transportar as peças para o armazém. Este processo é efetuado simultaneamente com os outros processos e sem intervenção do utilizador.

A maquinação de peças começa com um pedido do utilizador que inclui o número de peças a maquinar, o tempo de processamento nas máquinas e os tipos de ferramentas em cada maquinação. Para efeitos de simplicidade, definiu-se que as zonas de maquinação são compostas por dois tipos de máquinas, ou seja, na zona de maquinação em série temos uma máquina A e outra máquina B, acontecendo o mesmo na zona de maquinação em paralelo. Ambos os tipos de máquinas possuem três tipos de ferramentas diferentes. Antes de iniciar o processo de execução do pedido, é verificado se o número de peças no armazém é superior ao número de peças pedidas pelo utilizador, caso não seja, o pedido é considerado inválido e é apresentada uma mensagem de erro ao utilizador.

O pedido continua com a descarga das peças para o tapete de remoção (canto superior esquerdo) a partir deste, as peças seguem o seu percurso pela LIF. Caso ambas as maquinações estejam livres a aplicação escolhe a maquinação paralela para começar o pedido, por outro lado, caso alguma esteja ocupada o sistema escolhe a outra maquinação livre. Quando a peça chega à máquina, esta coloca a ferramenta escolhida pelo utilizador pronta para começar a maquinação, admitindo-se que sempre que o sistema é iniciado a ferramenta pré-definida é a ferramenta 1. Depois de maquinadas, as peças encaminham-se para o armazém à espera de uma ordem de descarga,

para que não exista confusão entre peças de diferentes pedidos, o sistema guarda uma variável com o número do pedido efetuado.

Finalmente quando é efetuado um pedido de descarga, o utilizador tem de especificar o número do pedido e qual a célula de saída, o tapete linear (canto superior direito) ou um dos *pushers*. Depois de verificado se o número de pedido existe, a descarga é iniciada. As peças são colocadas no tapete de remoção e são movimentados através do sistema até à célula de saída.

Para que o sistema funcione sem problemas existem algumas particularidades a referir. Caso alguma tapete, máquina ou *pusher* esteja ocupado durante o processo, a peça permanece no tapete anterior à espera que a célula de destino esteja disponível. Outra das particularidades é que não existe intercalação de pedidos, ou seja, um pedido de maquinação só é efetuado quando um pedido de descarga estiver completo. Por completo, considera-se que todas as peças já foram retiradas do armazém, assim vários pedidos podem ser processados simultaneamente na LIF, mas o processo de colocar peças na LIF nunca é intercalado.

No desenvolvimento da arquitetura do programa, e de modo a que esta conseguisse satisfazer os três serviços essenciais para a aplicação final, foram definidos alguns requisitos. A arquitetura da aplicação deve permitir boa escabilidade para que a adição de novos módulos ao sistema não seja um problema. Um controlo realizado através de várias camadas, permite que estas operem de forma independente passando informações e serviços às camadas num nível superior e inferior.

Tendo em conta estes requisitos e todo o sistema obteve-se o seguinte modelo:

Figura 4.2: Modelo da arquitetura do sistema.

Na figura 4.2 apresenta-se o modelo que se seguiu no desenvolvimento de toda a aplicação. No nível mais elevado do sistema temos o utilizador, onde é definida interface Homem-Máquina responsável por dar todas as ordens de execução ao sistema, desde o tipo de operação a realizar (carga, descarga, produção), até aos parâmetros de cada operação (tipos de matéria-prima, produtos finais e quantidade). Envia pedidos aos níveis inferiores, recebendo informações dos mesmos acerca do estado do processo. Imediatamente no nível inferior tem-se a coordenação que é responsável por definir a sequência de operações necessárias para realizar cada operação requisitada

pelo nível superior.

Para além disso, tem como função garantir a coexistência de todas as operações dos níveis inferiores. Adiante, serão apresentadas várias sequências que ilustram o tipo de ação executada por este nível. No nível seguinte estão presentes os módulos, que são responsáveis por dividir o sistema em diferentes secções por forma, a flexibilizar o mesmo. Assim, consegue-se simplificar a modelização individualmente, fazendo-os interagir entre si e integrando-os numa determinada tarefa. Por fim, as células, onde se considera todo o *hardware* constituinte da fábrica, entre os quais estão tapetes, máquinas e *pushers*, ou seja, os elementos mais básicos do sistema. Para além disso são, também, o que compõe os módulos referidos no nível anterior.

4.1.1 Diagramas UML

4.1.1.1 Diagrama de Classes

De modo a especificar as ideias apresentadas anteriormente elaboram-se vários diagramas designados por*Unified Modeling Language* UML - Classe, Sequência e Estados [15] para uma ilustração mais clara das interações entre as diferentes camadas.

Para identificar todos os "blocos" intervenientes no processo, utilizou-se um diagrama de classes. A realização dos diagramas começou-se do nível mais baixo, as células:

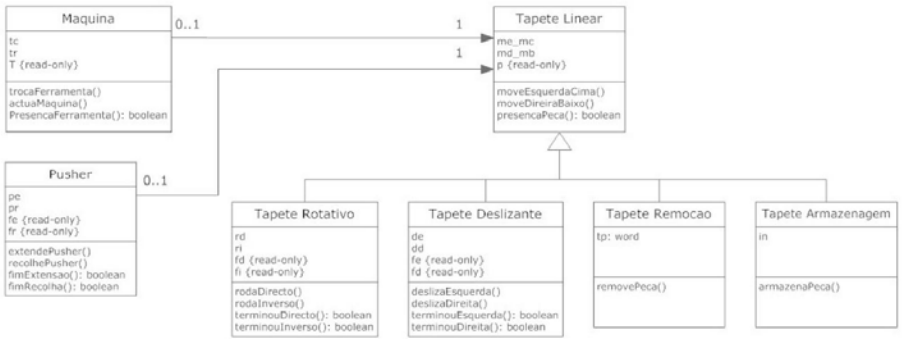

Figura 4.3: Diagrama de classes - Nível das células.

Neste nível temos uma classe central e base, o tapete linear. A partir desta classe é possível especificar todas as outras classes representadas na figura. De modo a melhor entender este diagrama, toma-se como exemplo um tapete linear e um deslizante. O tapete linear tem a capacidade de mover para a esquerda (moverEsquerdaCima()), mover para a direita (moverDireitaBaixo()) e detetar a presença de peça (presencaPeca()). O tapete deslizante possuiu todas estas capacidades, mas para além destas possuiu ainda a capacidade de deslizar quer para a esquerda quer para a direita (deslizarEsquerda(), (deslizarDireita()) como também detetar o término de deslizamento (terminouEsquerda(), terminouDireita()). Este raciocínio foi usado na especificação das restantes células.

Depois de terminada a especificação das células foi necessário realizar o mesmo processo para o nível dos módulos, para que tal fosse possível foi essencial efetuar a divisão da LIF em secções

de células. Esta divisão foi pensada para prover o programa de flexibilidade. O resultado está apresentado na figura 4.4:

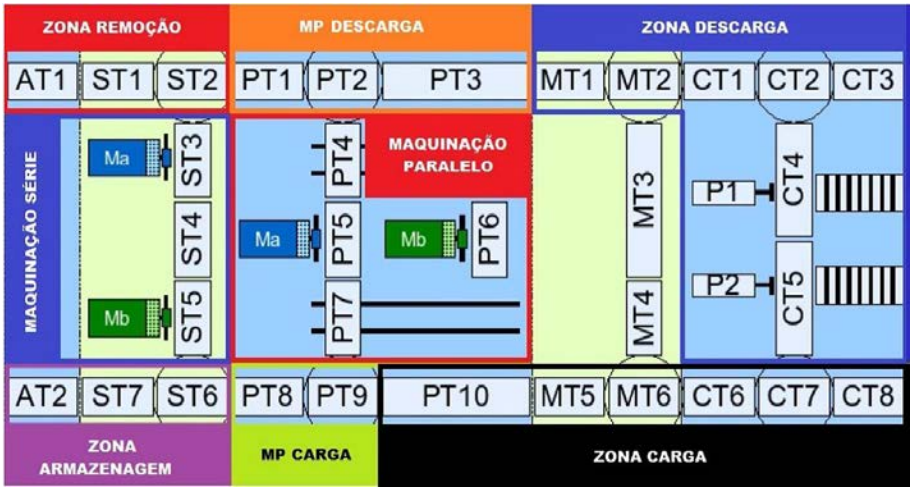

Figura 4.4: Visão da LIF dividada em módulos.

Como se pode conferir pela imagem, foram formadas oito secções: duas zonas de maquinação, uma zona de entrada de peças no armazém e outra de saída, duas zonas de operações com o armazém, armazenagem e remoção e, por fim, uma de entrada de peças na maquinação paralela e outra de saída de peças da mesma. Para melhor compreensão dos procedimentos realizados por cada secção, apresenta-se de seguida, uma breve descrição de cada uma:

- **Zona Remoção** – Ao receber uma peça no primeiro tapete linear (AT1), começa a transportar a mesma até ao fim desta zona (ST2). Ao chegar ao último tapete, um nível superior, decide se a peça segue para a maquinação série (e neste caso o tapete roda para enviar a peça) ou para a zona à frente (MPDescarga), fazendo esta escolha com base numa variável de entrada. Ao estar em posição de enviar a peça para a próxima zona, dá informação à última de que está pronto a enviar, e ao receber uma resposta de que pode iniciar a transferência, começa a executar a mesma.

- **MP Descarga** – MP Descarga – O funcionamento desta secção é idêntico ao da zona de remoção. Ao receber uma peça no primeiro tapete (PT1) procede ao transporte da peça até ao último tapete, que nesta zona pode ser o segundo (PT2) ou o último (PT3) conforme a finalidade da peça, sendo que no primeiro caso o destino da peça é a Maquinação Paralelo, e no segundo a Zona descarga. A seleção do destino é feita por uma variável de entrada definida por um nível superior. A transferência de peça para a zona seguinte é realizada da mesma forma que a Zona remoção.

- **Zona de Descarga** – Nesta secção é executada a parte final da descarga. Ao receber a peça no seu primeiro tapete (MT1) transporta a peça até ao tapete de decisão do local de descarga (CT2), e ao chegar a este último procede ao envio do tapete para o tapete CT3 ou

para a zona dos *pushers* baseado na ordem de um nível superior através de uma variável de entrada. Caso a descarga seja feita num dos *pushers*, a peça é levada até ao mesmo e ao chegar é ativado o pusher.

- **Zona de Carga** – Esta zona é ativada quando uma peça é colocada no primeiro tapete (CT8), e é depois realizado o transporte da peça até ao último tapete (PT10). Ao chegar ao último tapete é enviada uma informação à zona seguinte (MPCarga) de que uma peça está pronta para a transferência, realizando a mesma quando uma resposta de que a zona seguinte pode receber a peça for recebida.

- **MP Carga** – O programa produzido para o funcionamento desta zona, tal como as outras, tem como principal função transportar uma peca desde o primeiro tapete (PT9) até ao último (PT8) procedendo depois a transferência da peça para a zona seguinte. A peça pode ser recebida da Maquinação Paralelo ou da Zona Carga, dando prioridade à Maquinação Paralelo, e sendo a recepção processada como na Zona Armazenagem.

- **Zona Armazenagem** – Ao receber a peça no seu primeiro tapete (ST6), procede ao transporte da mesma até ao último tapete (AT2). A recepção pode ser pela Maquinação Série ou pela Zona MPCarga, dando prioridade a Maquinação Série. Quando a ordem de execução é dada o primeiro tapete roda para a zona que o requisitou dando, depois, a informação de que está pronto a receber. Quando receber volta a rodar para a posição horizontal e procede ao transporte da peça. Quando a peça atinge o tapete final, é dada a ordem ao armazém para recolher a peça.

- **Maquinação Série** – Nesta secção as peças são guiadas pelo módulo de maquinação em série, introduzindo uma peça no módulo se a máquina A estiver livre. Após a recepção da peça é verificado o tempo de maquinação na máquina A e máquina B, levando a peça até às máquinas onde será processada, quer seja numa só máquina quer seja nas duas. No final desta maquinação esta é guiada para o exterior deste módulo.

- **Maquinação Paralela** – Quando recebe uma peça no primeiro tapete (PT4) verifica os tempos de maquinação, tal como na maquinação em série. Após esta verificação decide qual a máquina que vai atuar primeiro sobre a peça, que é depois transportada até a mesma. Quando a peça chega a máquina desejada, é então dada a ordem à mesma para atuar e no fim do processo a peça é transportada até a último tapete (PT7) que será transferida para a outra máquina caso esta ainda tenha que ser maquinada ou dando ordem de peça em espera para a Zona MPCarga e inicia a transferência aquando da recepção de uma resposta da última zona.

  Com o nível dos módulos já segmentado e descrito especificou-se então o diagrama de classes (este está dividido em três figuras diferentes):

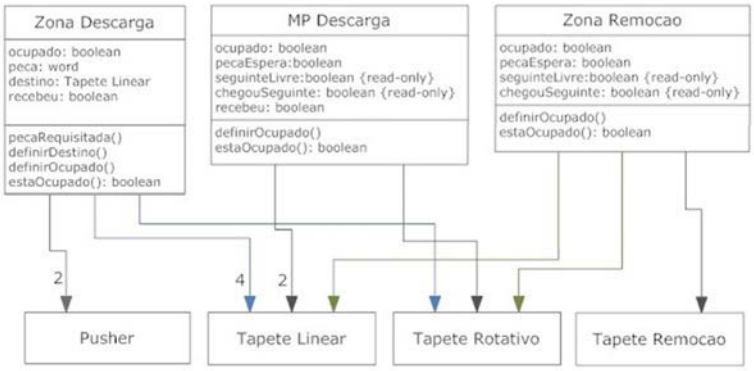

Figura 4.5: Diagrama de classes - Nível dos Módulos. (Parte 1)

Nestes digramas estão ilustrados os módulos e os diferentes elementos necessários que constituem os constituem. Todos estes módulos têm características em termos de funções que os caracterizam bastante idênticas, ou seja, todos eles têm como objetivo informar os outros módulos do seu estado, ocupado ou não ocupado.

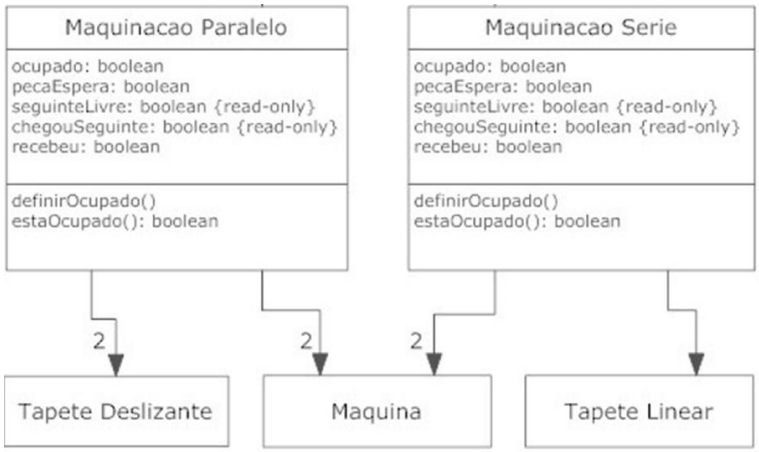

Figura 4.6: Diagrama de classes - Nível dos Módulos. (Parte 2)

Por exemplo, a secção MP Carga tem como variáveis, ‘ocupado’, esta variável fornece informação aos outros módulos do estado de ocupação do módulo; pecaEspera, com esta variável é possível informar que existe uma peça em espera pronta a seguir o seu fluxo para o seguinte módulo; seguinteLivre, depois de dada a informação que existe uma peça em espera, o módulo recebe a informação do módulo seguinte que está livre e pode assim começar com o processo de transferência de peça; chegouSeguinte, tal como o nome indica esta variável informa o módulo que a peça já chegou ao módulo seguinte, é neste momento que o estado passa de ocupado para não ocupado.

Todas as operações executadas dentro do próprio módulo são da responsabilidade das células, neste exemplo o tapete linear e rotativo são os responsáveis por fazer chegar a peça ao seu destino.

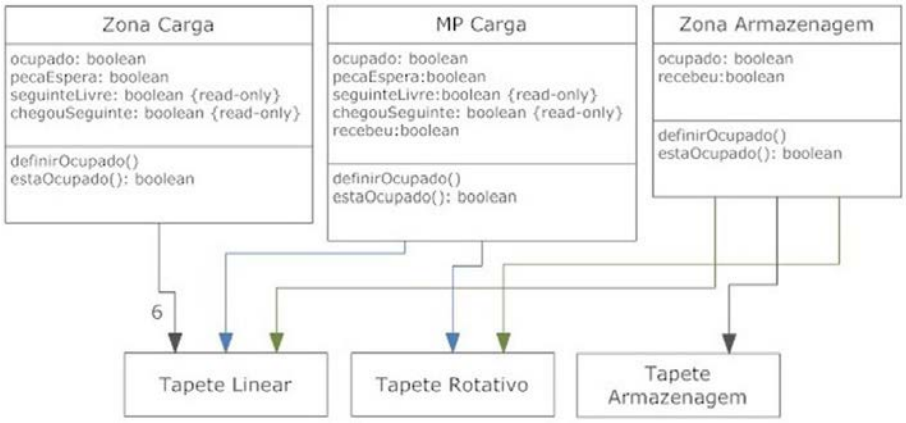

Figura 4.7: Diagrama de classes - Nível dos Módulos. (Parte3)

Para o nível de controlo foi elaborado os seguinte diagrama:

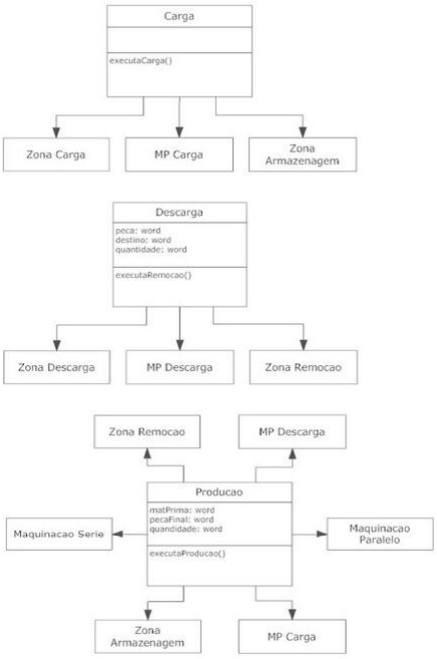

Figura 4.8: Diagrama de classes - Nível de Controlo.

Neste nível da arquitetura estão representadas as três operações, definidas no início deste capítulo que o sistema terá que efetuar: produção, carga e descarga de peças.

Tomando como exemplo o processo de descarga, pode-se verificar que para efetuar uma descarga são precisos incluir três módulos: Zona Remoção, responsável por remover a peça do armazém; MP Descarga, módulo que transfere a peça para a zona de Descarga, Zona de Descarga, responsável pela descarga da peça pela correta célula de saída escolhida pelo utilizador.

Por último tem-se o nível do utilizador, para este nível não se efetuou um diagrama de classes porque o utilizador apenas interage com o sistema fazendo pedidos de carga, descarga ou produção. Nestes pedidos o utilizador tem de definir algumas variáveis para que o pedido possa ser efetuado corretamente. No pedido de produção, é necessário especificar o número de peças a produzir, o tipo de peça pretendido como matéria-prima e qual o tempo de maquinação nas respetivas máquinas. O pedido de descarga apenas inclui a peça a descarregar e a quantidade pretendida. No pedido de carga o utilizador não interage com a aplicação como um pedido normal, apenas coloca a peça no tapete de entrada da LIF e esta automaticamente carrega a peça para o armazém.

4.1.1.2 Diagramas de Estados

Para completar a arquitetura descreveu-se através de diagramas de estados o nível mais baixo, o nível das células. Esta análise focalizou-se mais propriamente na interação e na execução ao nível dos sensores e actuadores dos tapetes, máquinas e *pushers*.

Tapete Linear:

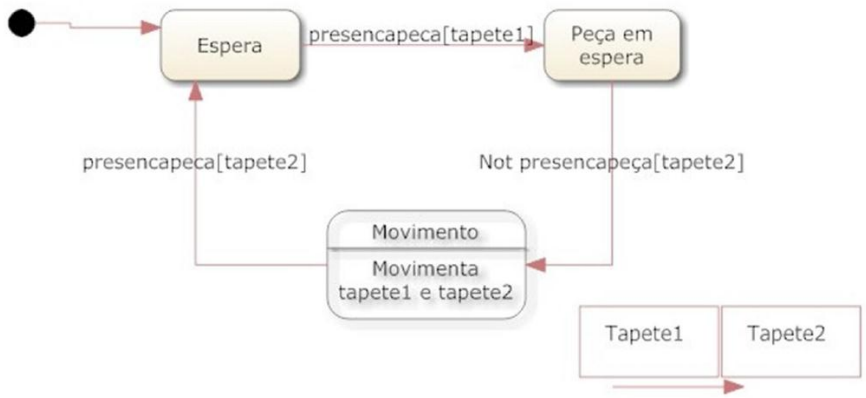

Figura 4.9: Diagrama de estados - Tapete Linear.

As operações no tapete linear começam quando o sensor de presença é ativado. Neste momento o tapete muda de estado à espera que o tapete seguinte esteja livre. Quando tal acontece o movimento dos dois tapetes começa e só acaba com quando o sensor de presença do segundo tapete é ativado, passando assim o primeiro tapete para o estado de repouso, o estado inicial.

Tapete rotativo ou deslizante:

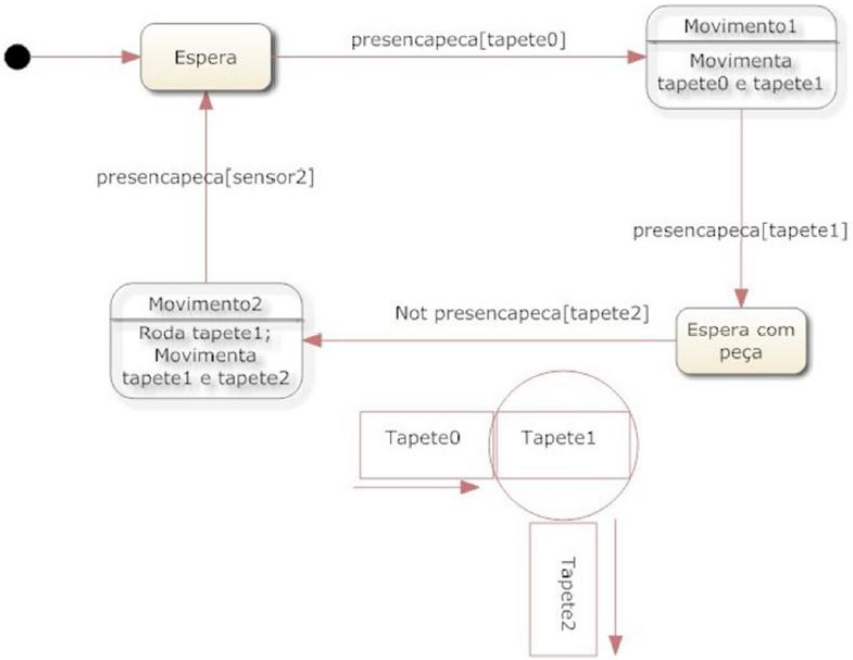

Figura 4.10: Diagrama de estados - Tapete Deslizante.

Este diagrama é representativo dos tapetes lineares presentes na LIF, no entanto os tapetes deslizantes podem ser descritos da mesma forma, havendo somente uma alteração. Quando o tapete rotativo efetua a operação Roda tapete1, o tapete deslizante efetua o Desliza tapete1. O início deste diagrama é em tudo idêntico às operações realizadas por dois tapetes lineares. A peça é transportada desde o tapete linear até ao tapete rotativo de igual forma.

Depois desta operação e quando o tapete seguinte estiver disponível, o tapete rotativo começa a sua rotação até atingir o sensor e de seguida o movimento da peça para o tapete final. Esta operação termina com a chegada da peça ao tapete final, voltando assim ao estado inicial, repouso.

Máquina/Tapete Linear:

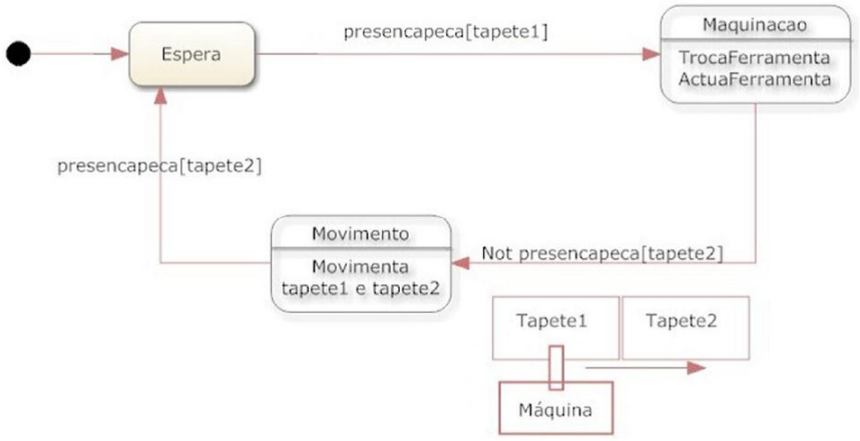

Figura 4.11: Diagrama de estados - Tapete Deslizante.

Este diagrama é em tudo idêntico ao de dois tapetes lineares, isto é, a peça é transportada entre dois tapetes da mesma forma como ocorre nos tapetes lineares. No entanto, quando a peça está no tapete um, ocorrem as ações efetuadas pela máquina, trocaFerramenta e ActuaFerramenta. Estas ações vão ser as responsáveis pela maquinação da peça e pela sua transformação. No final deste processo, ocorre o transporte da peça até ao tapete final da mesma forma dos diagramas a anteriores.

Pushers:

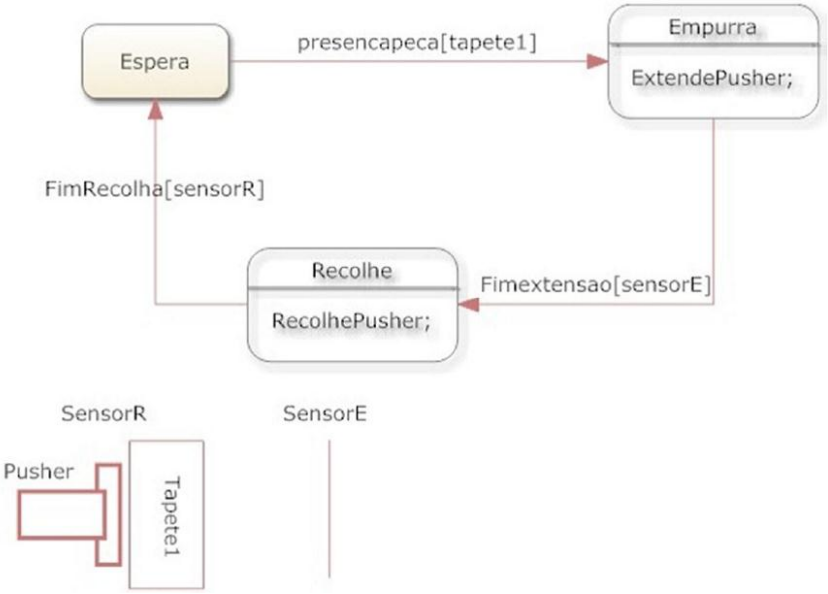

Figura 4.12: Diagrama de estados - *Pushers*.

O final da operação de descarga de peças do sistema é efetuado pelos *pushers*. Quando a peça a descarregar aciona o sensor de presença do tapete acoplado ao*pusher*, este muda de estado começando o processo de extensão. Quando o sensor E é ativado, a peça já se encontra descarregada e o *pusher* começa com a recolha do braço até que o sensor R fique ativo. Neste momento o processo chegou ao seu final e o *pusher* volta ao seu estado inicial.

4.2 Estudo do Beremiz

Um dos objetivos deste projeto, o de maior importância e relevância, é o teste e validação do Beremiz enquanto alternativa válida no desenvolvimento de aplicações de controlo no âmbito da automação industrial.

Nesta subsecção serão abordados todos aspectos positivos e negativos no uso desta ferramenta.

O Beremiz, como referido anteriormente, é um *software open-source*, grátis e multiplataforma começando aqui as suas grandes vantagens em relação aos seus concorrentes, soluções proprietárias com custos elevados. Além disso, este implementa a norma IEC-61131-3 de forma rigorosa.

Inicialmente foi necessário escolher em que plataforma se iria desenvolver o programa, visto que o *software* é *open-source*. As opções disponíveis eram o *Linux* ou o *Windows* como plataformas de desenvolvimento. A escolha foi feita com base numa limitação do protocolo de comunicação porque o *plugin* que permite a comunicação através de Modbus só estava disponível para *Linux*.

Prosseguiu-se então com a instalação do *software*, usando como plataforma o *Linux Mint 13*. Em paralelo, decidiu-se instalar a versão *Windows (Windows XP)* para que as duas versões pudessem ser comparadas em simultâneo. Esta comparação podia ser benéfica porque caso a versão *Windows* estivesse melhor optimizada e com melhor desempenho seria vantajoso portar o *plugin* Modbus para a plataforma *Windows*. Após terminada a instalação em ambas as plataformas foi possível verificar que a versão *Linux*, disponibilizada no repositório do *software*, era idêntica à versão *1.1 Release Candidate no Windows*. Esta dúvida surgiu depois de na versão *Linux* não existir qualquer tipo de indicação quanto à versão.

De seguida, iniciou-se uma pesquisa por documentação como manuais do utilizador ou tutoriais para que a adaptação à aplicação e aos seus componentes fosse mais suave e no menor espaço de tempo. No entanto, a documentação fornecida no *site* é escassa e pouco explicativa. Esta só ilustra quais os componentes da aplicação e a sua finalidade de forma reduzida. Contudo, é elucidativo que o *software* segue à risca a norma, como tal em termos de implementação de POU, tipos de dados e configurações foi seguido o que a norma sugere e determina.

O primeiro contacto com o ambiente de desenvolvimento foi positivo, o *software* apresenta uma disposição arrumada e simples. Todas as componentes são possíveis de redimensionar, o que faz com que se possa aumentar ou diminuir a área de trabalho conforme as necessidades o que é de facto uma grande vantagem. Um aspecto menos positivo é que não existe nenhuma opção para desativar a visualização de alguns dos componentes quando estes não são necessários, é somente possível diminuir a sua dimensão até ao mínimo, o que sendo útil não é de todo o ideal.

Depois de conhecidas todas as componentes do *software* implementou-se um pequeno programa de teste com o objetivo de testar os dois tipos de linguagens mais utilizadas SFC e ST. Este programa tem como finalidade partir do repouso e quando detetada a peça no tapete transportá-la para o seguinte e parar o movimento. Na implementação deste pequeno programa deparou-se com algumas particularidades que o Beremiz obriga.

Quando se pretendia voltar da última transição para o estado inicial tal não era possível com uma ligação normal, este processo só podia ser realizado com o recurso a um *jump*. Na implementação de uma transição, este não permite a sintaxe: "presencapeca;", este excerto de código pretende que a transição fique ativa quando presençapeca tome o valor 1. Para que não existam erros de compilação deve-se usar: ":= presencapeca;". Esta particularidade aproxima o Beremiz mais da norma que as restantes soluções.

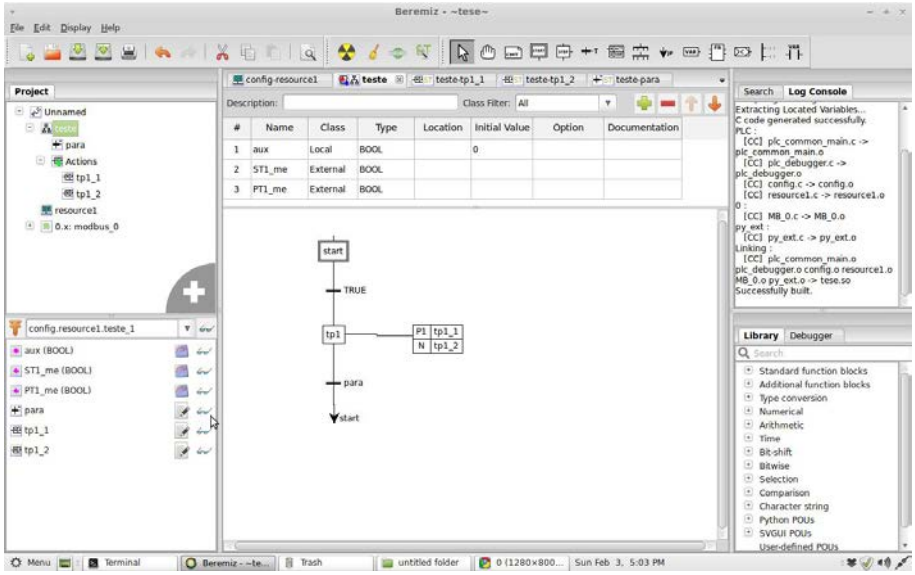

Figura 4.13: Interface do Beremiz - programa de teste.

Na figura 4.13 apresenta-se a interface do Beremiz e o programa de teste implementado. Como se pode verificar do lado esquerdo está a árvore do projeto: Programa -> Transição e Ações. Na parte inferior têm-se as variáveis já listadas neste *resource*. No centro o editor, na parte superior a gestão de variáveis e na parte inferior o código. Do lado esquerdo estão representados a *Log Console* e a biblioteca de funções padrão, na parte superior e inferior respetivamente. Esta configuração foi a utilizada durante o projeto, no entanto a colocação das janelas é configurável.

Depois da compilação efetuada com êxito, seguiu-se para a fase de testes. Nesta fase começou-se pela adição do *plugin* Modbus. Para adicionar o *plugin*, é necessário adicionar os ficheiros do *plugin* à pasta do Beremiz e de seguida efetuar a compilação do mesmo para que esteja pronto para comunicar. Esta fase foi demorada, isto porque após inúmeras tentativas o *plugin* continuava a não aparecer na interface do programa. Mais tarde, descobriu-se que no processo de compilação do *plugin* não é dada a instrução para o adicionar visualmente na interface. Ou seja, o protocolo de comunicação Modbus encontrava-se funcional, apenas não estava presente a opção de adição na

interface de Beremiz. Este foi um dos grandes obstáculos que surgiram no decorrer deste trabalho, no entanto não se trata de um problema do *software* mas do *plugin*. Para contornar o problema foi adicionada a seguinte linha de código ao ficheiro com as funcionalidades, *features.py*:

('modbus', _('Modbus Client support'), _('Map located variables over Modbus'),
'modbus.modbus.RootClass'),

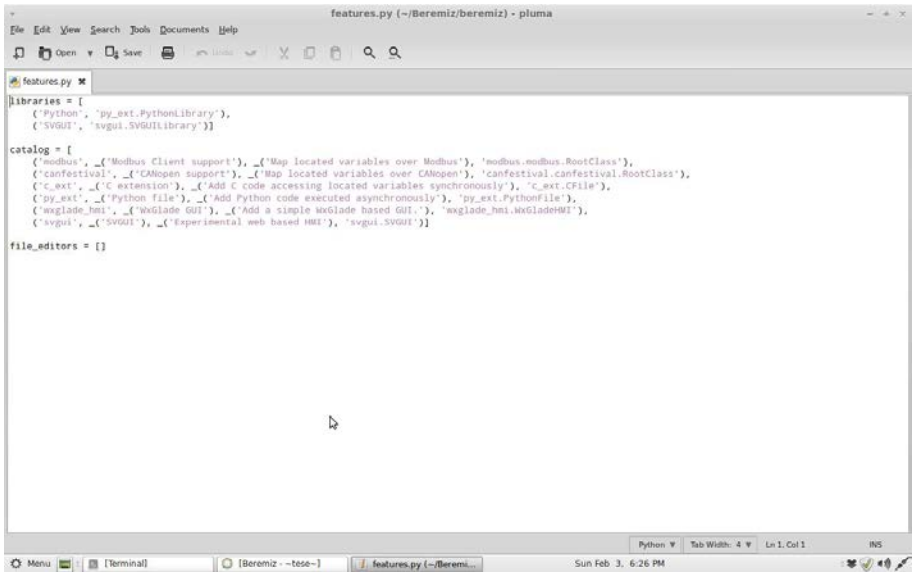

Figura 4.14: Ficheiro com as funcionalidades do Beremiz.

Para testar a comunicação foi utilizado um simulador disponibilizado pela FEUP [16]

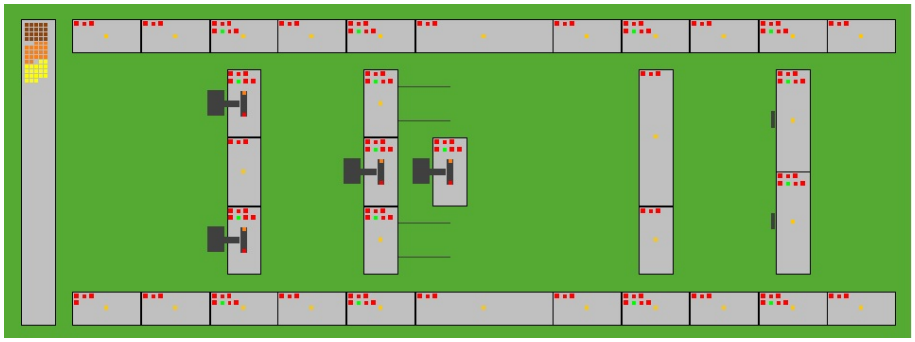

Figura 4.15: Simulador representativo da LIF.

Para que a comunicação com o simulador fosse possivel foi necess mapear as variáveis necessárias para os tapetes que se pretendiam mover. Mais uma vez na interação com o *plugin* o Beremiz não foi bem-sucedido, visto que quando se queria adicionar a localização da variável era apresentada uma janela para escolher quais dos módulos Modbus se pretendia escolher, escrita ou leitura. Quando selecionado um desses módulos o Beremiz respondia com erro, "*Location must*

be selected", figura 4.16. A localização mesmo depois de escolhida não era reconhecida, este problema foi rapidamente ultrapassado com a descrição manual da localização: "IW0.0.0.XX".

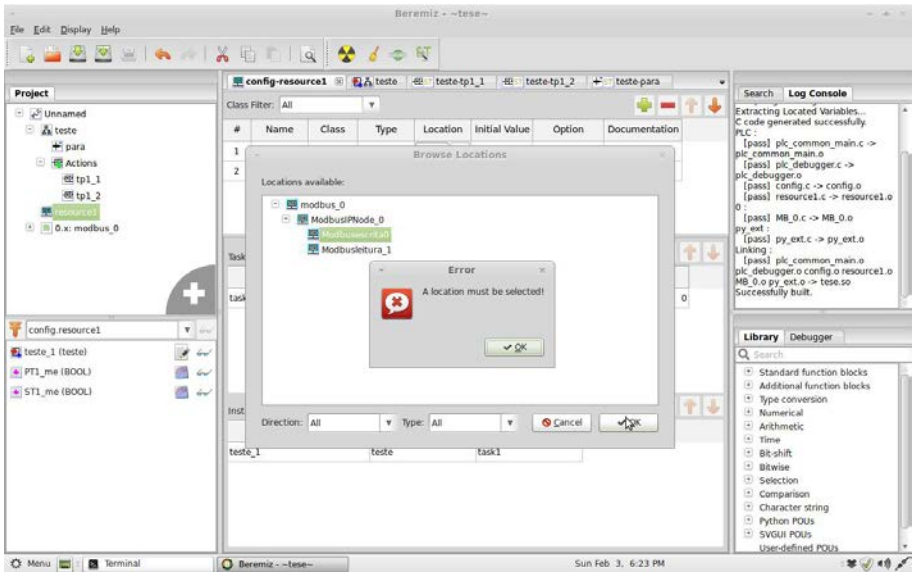

Figura 4.16: Interface Beremiz - erro na definição da localização das variáveis.

Antes da compilação ser concluída com êxito foi necessário modificar o ficheiro de compilação, adicionou-se o comando "-fpic", pois sem este comando o compilador não conseguia concluir o processo com êxito. Por fim, o teste foi realizado com êxito. Este processo de teste demorou mais tempo que o esperado aquando do planeamento do trabalho, nomeadamente o processo de adição do *plugin* à interface do Beremiz.

Depois de concluída a fase de testes, começou-se com o desenvolvimento da aplicação de controlo. O primeiro passo no desenvolvimento da mesma foi o mapeamento de todas as variáveis referentes ao simulador, este tornou-se o grande problema no desenvolvimento da mesma.

As aproximadamente cento e sessenta variáveis de entrada e saída do simulador tornaram o programa muito lento e com o qual era difícil interagir, tanto na versão *Linux* como na versão *Windows*. Para tentar contornar o problema, testou-se em *Windows* uma versão mais antiga do Beremiz, versão 1.04 só disponível para *Windows*. Nesta versão o problema da lentidão não ocorria, logo pensou-se em desenvolver o programa em *Windows* e testá-lo no *Linux* devido à disponibilidade do *plugi*n Modbus. Teoricamente não deveria existir qualquer problema porque o *software* tem como uma das características principais ser multiplataforma.

Quando realizada a primeira importação do programa percebeu-se que não seria possível utilizar este método alternativo porque a versão presente em *Linux*,*1.1 Release Candidate*, implementa uma diferente visão das definições do projeto o que provocou uma importação das variáveis de forma errada, tornando inviável esta opção. Este fator atrasou o decorrer do projeto, visto que, em certas alturas tornava impossível a interação com o Beremiz devido ao seu elevado tempo de resposta.

Além destes erros bem documentados, vários erros de "*segmentation fault*" foram acontecendo de forma recorrente obrigando a que o programa fosse gravado múltiplas vezes para que não fosse perdida informação. Outros dos problemas detetados e que prejudicou o bom andamento deste projeto foi a falta de informação quando apareciam erros no compilador, em muitos casos a informação fornecida era "*internal error*", isto tornou uma tarefa quase impossível o trabalho de *debugging*.

Posto isto e de forma a que no futuro seja possível corrigir ou reportar estes bugs apresenta-se de seguida uma lista descritiva dos mesmos.

1. **Erro ao retornar ao Estado inicial** (SFC)

   Descrição: Não é possível retornar ao estado inicial através de uma ligação em fio.

   Solução: Para ser possivel retornar ao estado inicial utiliza-se um jump, característica da linguagem SFC.

2. **Função com string como variável de entrada** (ST)

   Descrição: Não é possivel utilizar uma string como variável de entrada de uma função.

   Solução: De modo a contornar este aspecto cria-se uma lista que faça a correspondência entre as strings a utilizar e um inteiro, apesar de não ser a solução ideal.

3. **Varáveis de Estado**

   Descrição: Não é possível a utilização de variáveis de estado. Este erro provém do facto de esta função não estar implementada no compilador MATIEC.

   Solução: Quando for necessária a informação de um certo estado ativo ou não, cria-se uma variável auxiliar dentro do estado que nos informará o estado do mesmo. Esta solução não é a ideal e aumentará o número de variáveis no programa.

4. **Erro na gestão de *plugins***

   Descrição: Não é possivel adicionar variáveis remotas na gestão de plugins, apesar de existir essa opção.

   Solução: Para adicionar variáveis remotas tem de se adicionar o endereço manualmente.

5. **Integração com o *plugin***

   Descrição: Quando se integra o *plugin Modbus* e depois de compilá-lo é necessário acrescentar manualmente a sua compenente visual não sendo autmática esta parte.

   Solução: Acrescentar manualmente ao ficheiro features.vy a seguinte linha de código: **('modbus', _('Modbus Client support'), _('Map located variables over Modbus'), 'modbus.modbus.RootClass'),**

6. **Elevado tempo de resposta**

Descrição: À medida que se vão adicionando variáveis e código no programa este vai ficando com um maior tempo de resposta. Apartir de aproximadamente 80 variáveis, o programa demora cerca de 1s a responder; 150 variáveis 3s.

Solução: De modo a contornar este problema poderia-se acrescentar as variáveis num ficheiro à parte, ou programar a aplicação de controlo por partes.

7. **Falhas de Segmentação**

   Descrção: À medida que a informação vai aumentando e com redimensionamento das diferentes componenetes visuais do programa ocorrem falhas de segmentação. Estas não são possíveis de definir exatamente o momento em que ocorrem.

   Solução: Não existe qualquer forma de impedir que esta situação aconteça.

Capítulo 5

Conclusões e Trabalho Futuro

5.1 Conclusões

Este trabalho tinha dois objetivos principais, a modelação de uma aplicação de controlo da Linha Industrial Flexível presente na FEUP e o teste e validação do Beremiz, como ambiente de desenvolvimento na área da automação industrial.

O primeiro passo no projeto foi o estudo da Linha Industrial Flexível e de todos os seus módulos e células, de modo a ficaram listadas todas as capacidades e funções da mesma, bem como todos os sensores e actuadores presentes.

Depois de estudada a LIF, foi possível começar a modelação da aplicação de controlo da mesma. Inicialmente foi necessário definir quais os serviços essenciais que devia executar, definir o fluxo de peças e o modo de funcionamento da mesma. Após definidas estas características, procedeu-se à elaboração de diagramas recorrendo a ferramentas de ilustração abstratas como o UML. Os diagramas de classes e de estados representam todos os módulos e funções que irão estar presentes na aplicação de controlo.

Concluída com êxito a fase de modelação do sistema prosseguiu-se para a etapa seguinte, teste e validação do Beremiz. Nesta foi necessário estudar a norma IEC 61131-3 de modo a se conhecer todas as suas características para que a implementação da aplicação siga a norma à risca. Foi também necessário o estudo do protocolo Modbus, para que o modo como a comunicação é realizada fosse entendido, no caso de surgirem eventuais *bugs*, ser mais fácil a sua resolução. No final deste estudo, iniciou-se a fase de testes do Beremiz. Esta fase de testes demorou mais tempo que o estabelecido inicialmente. Isto ocorreu devido aos muitos problemas entre o Beremiz e o *plugin* Modbus.

Mesmo ultrapassando os erros com sucesso, o tempo restante para a implementação da aplicação de controlo era escasso. A lentidão do Beremiz e o aumento dos *segmented fault* depois de declaradas todas as variáveis não contribuíram para que a aplicação de controlo fosse implementada na totalidade. No entanto, a implementação apenas não foi concluída totalmente devido à falta de tempo, pois com a modelação cuidada e pormenorizada que foi elaborada, com o correto

funcionamento do Beremiz e da integração do *plugin* Modbus esta não seria o maior desafio deste projeto.

Em suma, é possível afirmar que o Beremiz pode ser um substituto às soluções proprietárias presentes no mercado, no entanto a falta de documentação, a falta de informação na adição de *plugins* e a falta de informação nos erros de compilação podem ser um obstáculo difícil de transpor. O Beremiz ainda tem um percurso longo a percorrer para que possa ser usado em sistemas críticos.

5.2 Trabalho Futuro

Inicialmente foi planeado a conclusão da implementação da aplicação de controlo, tal não foi possível devido aos inúmeros obstáculos encontrados. É expectável que tal seja concluído no final deste trabalho. Por último ao longo do projeto outras ideias surgiram para melhoramento do mesmo:

- A modelação e implementação de uma interface de controlo da aplicação, para que o controlo da LIF seja realizado de forma mais intuitiva e facilite a demonstração aos alunos.
- A integração da aplicação de controlo num microcontrolador para que o transporte da aplicação com a LIF não seja um problema e o sistema fique mais compacto e robusto.

Referências

[1] Prof. Mario Sousa. Sistemas de informacao industriais, projecto de uma celula de producao. Guiao do projecto da cadeira informatica industrial, 2010.

[2] Daniel Andre da Silva Petim Batista. Automacao de linha de fabrico flexivel do deec. Tese de mestrado, Faculdade de Engenharia da Universidade do Porto, 2001.

[3] K.H. John e M. Tiegelkamp. *IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Decision-making Aids*. Springer, 2010.

[4] Prof. Mario Jorge Rodrigues de Sousa e Prof. Adriano Carvalho. *The Industrial Technology Information Handbook*, chapter 65. CRC Press, 2004.

[5] PLC Dev. Sequential function charts for all. Disponivel em http://http://www.plcdev.com/.

[6] Lolitech. Beremiz user manual. Disponivel em http://www.beremiz.org, 2008.

[7] ModbusIDA. Modbus messaging on tcp/ip implementation guide v1. 0a, 2004.

[8] Vasco das Neves Fernandes. Driver modbus para aplicacao iec 61131-3. Tese de mestrado, Faculdade de Engenharia da Universidade do Porto, 2009.

[9] ModbusIDA. V. 1.1 b. *Hopkinton, Massachusetts (www. modbus. org/docs/Modbus Application Proto col V1 1b. pdf)*, 2006.

[10] C. Neves, L. Duarte, N. Viana, e V. Ferreira. Os dez maiores desafios da automacao industrial: as perspectivas para o futuro. Em *II Congresso de Pesquisa e Inovacao da Rede Norte Nordeste de Educaçao Tecnológica, Joao Pessoa, Paraıba, Brasil*, 2007.

[11] A.A.B. Buccioli, E.R. Zorzal, e C. Kirner. Usando realidade virtual e aumentada na visualização da simulação de sistemas de automação industrial. Em *SVR2006-VIII Symposium on Virtual Reality*, 2006.

[12] E. Tisserant, L. Bessard, e M. de Sousa. An open source iec 61131-3 integrated development environment. Em *Industrial Informatics, 2007 5th IEEE International Conference on*, volume 1, páginas 183–187. IEEE, 2007.

[13] M. De Sousa e A. Carvalho. An iec 61131-3 compiler for the matplc. Em *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA'03. IEEE Conference*, volume 1, páginas 485–490. IEEE, 2003.

[14] Vitor Amadeu Souza. O protocolo modbus. Disponivel em http://www.cerne-tec.com.br.

[15] D.F. D'souza e A.C. Wills. *Objects, components, and frameworks with UML: the catalysis approach*, volume 1. Addison-Wesley, 1998.

[16] Prof. Andre Restivo. Simulador da linha industrial flexivel. Disponivel em http://paginas.fe.up.pt/ arestivo/wiki/.