Universidade do Minho
Escola de Engenharia

Joel Maurício Rocha Galvão

Conversão sistemática do comportamento definido nos blocos funcionais da norma IEC 61 131-3 para autómatos finitos temporizados.

Joel Maurício Rocha Galvão
Conversão sistemática do comportamento definido nos blocos funcionais da norma IEC 61 131-3 para autómatos finitos temporizados.

UMinho | 2015

outubro de 2015

Universidade do Minho
Escola de Engenharia

Joel Maurício Rocha Galvão

Conversão sistemática do comportamento definido nos blocos funcionais da norma IEC 61 131-3 para autómatos finitos temporizados.

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao
Grau de Mestre em Engenharia Mecânica

Trabalho efectuado sob a orientação do
Professor Doutor José Mendes Machado

outubro de 2015

DECLARAÇÃO

Nome: Joel Maurício Rocha Galvão

Endereço eletrónico: a63363@alunos.uminho.pt Telefone: 918133914

Bilhete de Identidade/Cartão do Cidadão: 14239192 1ZY0

Título da dissertação: Conversão sistemática do comportamento definido em blocos funcionais da norma IEC 61 131-3 para autómatos finitos temporizados

Orientador:

Professor Doutor José Mendes Machado

Ano de conclusão: 2015

Mestrado Integrado em Engenharia Mecânica

Universidade do Minho, _____/_____/_____

Assinatura:

AGRADECIMENTOS

A realização desta dissertação de mestrado contou com apoios e incentivos sem os quais não se teria tornado uma realidade e aos quais estou eternamente grato.

Ao Professor Doutor José Mendes Machado, pela sua orientação, total apoio, disponibilidades, pelo *Know-how* transmitido, pelas opiniões e criticas, colaboração no solucionar de dúvidas e problemas que foram surgindo e pelo tempo despendido em reuniões e leitura desta dissertação.

Aos meus pais, por me darem a oportunidade de concluir este ciclo de estudos e pelo apoio e suporte na realização desta dissertação.

Á minha irmã pelo apoio e ajuda na elaboração desta dissertação e suporte nas durante os cinco anos deste ciclo de estudos.

Á Glennda pelo apoio, suporte e ajuda que sempre me deu em todas as fases de execução deste trabalho.

Por fim, agradeço também todas as pessoas que de forma direta ou indireta ajudar a concluir este curso.

RESUMO

Nesta dissertação é proposto um conjunto de modelos de alguns blocos funcionais da norma IEC 61 131-3 tendo em conta a Simulação e a verificação formal da especificação de comando de um sistema mecatrónico. A modelação destes blocos funcionais é feita utilizando TA (*Timed Automata*) e o *software* de simulação e verificação (UPPAAL), que permite simular e realizar *model checking* sobre modelos em TA.

Todos os comportamentos dos blocos funcionais considerados relevantes foram transponíveis para um modelo em autómatos finitos temporizados e foi possível tirar conclusões sobre a Simulação e verificação formal do comportamento dos mesmos. Um estudo de caso foi considerado para ilustração dos resultados obtidos e, também, permitiu extrapolar conclusões tiradas, para outros casos similares.

PALAVRAS-CHAVE

Blocos funcionais da norma IEC 61 131-3, Simulação de Sistemas Automatizados, Sistemas de Eventos Discretos, Verificação Formal, *Model-In-the-Loop*, *Model Checking*, Autómatos Finitos Temporizados, *Sequential Function Chart*, UPPAAL.

ABSTRACT

In this dissertation, a series of models of same function blocks of the standard IEC 61 131-3 are proposed, taking into account the simulation and formal verification of the command specification of a mechatronic system. The modelling of these function blocks is done using TA and software of simulation and verification (UPPAAL) that allows the simulation and model checking of models in TA.

The behavior of all the function blocks that were considered relevant were converted to TA models and it was possible to take conclusions about the simulation and formal verification of said models. A case study was considered to demonstrate the results obtained and also extrapolate the conclusions to other similar cases.

KEY WORDS

IEC 61 131-3 Function Blocks, Simulation of Automation Systems, Discrete event Systems, Formal Verification, Model-In-the-Loop, Model Checking, Timed Automata, Sequential Function Chart, UPPAAL

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

Lista de Abreviaturas, Siglas e Acrónimos

BIP	Behavior, Interactions, Priorities
CAE	Computer Aided Engineering
CC	Clearing Condition
CD	Decrement Input signal
CLK	Clock
CPU	Control Processing Unit
CTD	Down counter
CTDU	Up-Down Counter
CTL	Computation Tree Logic
CTL*	Full Computation Tree Logic
CTU	Up Counter
CU	Increment Input signal
CV	Current Value
DES	Discrete event system
DEVS	Discrete Event System Specification
ET	End Time output
FB	Function Block
FBD	Function Blocks Diagram
GI-SIM	GRAI/IDEF-Simulation
HiL	Hardware-in-the-loop
IDEF0	Icam DEFinition for Function Modeling
IEC	International Electrotechnical Commission
IL	Instruction List
IN	Input of function block
ISO	International Organization for Standardization
JR-net	Job Resource relation-net
LD	LoaD value
LD	Ladder Diagram

LT	Laboratory-testing
LTL	Linear Temporal Logic
MEM	Memory
MiL	Model-in-Loop
NCES	Net Condition/Event systems
NR	Not Running
PLC	Programmable Logic Controller
POU	Program Organization Unit
PT	Predefined Time
PV	Pre-set Value
Q	Generic Output of function block
QD	Output for counting Down
QU	Output for counting Up
R	Running
R	Reset
SDV	Simulink Design Verifier
SFC	Sequential function Charts
SiL	Software-in-the-loop
ST	Structured Text
STG	State Transition Graph
TA	Timed Automata
TC	Tempo de ciclo (Scan Cycle)
TCP	Modbus transmision control protocol
TCTL	Timed Computation Tree Logic
TO	Time Out
TOF	Time OFF delay
TON	Time ON delay
UT	Time units
X	Variável de atividade de etapa
XML	eXtensible Markup Language

Capítulo 1
INTRODUÇÃO

Este capítulo introdutório tem como finalidade apresentar o enquadramento e a motivação para este trabalho, bem como os objetivos e a estrutura do documento apresentado.

1. INTRODUÇÃO

1.1 Enquadramento

Atualmente, no desenvolvimento de equipamentos industriais, pretende-se responder a especificações de desempenho elevadas utilizado uma abordagem mecatrónica, onde é aplicada uma filosofia coordenada e simultânea, entre a mecânica e a eletrónica, e o controlo inteligente no fabrico de equipamentos [1]. Equipamentos mecatrónicos oferecem um potencial de sucesso elevadíssimo, mas ao mesmo tempo impõem requisitos especiais no seu desenvolvimento, porque integram várias áreas de conhecimento distintas. Os sistemas desenvolvidos com esta metodologia são bastante complexos, isto deve-se ao elevado número de componentes quando comparado com um sistema mecânico normal. Este problema deve ser tido em conta nas fases mais preliminares do desenvolvimento do produto, pois a interação entre os componentes eletrónicos, mecânicos e informáticos influencia o comportamento do sistema final [2]. Muitas das vezes os novos produtos partem de equipamentos já existentes, isto faz com que exista a necessidade de reutilizar códigos, do *software* de comando, e repetir soluções. Este facto pode, muitas vezes, levar a pequenos erros no processo de desenvolvimento do *software* de comando destes sistemas, devido à elevada complexidade, que tem motivado a busca de métodos que permitam a agilização deste processo e ao mesmo tempo garantir segurança para todas as partes intervenientes.

De facto, a maior parte das falhas existentes nos sistemas mecatrónicos são consequência de erros na conceção do código implementado nos autómatos. Normalmente, estes erros são fruto do uso múltiplo da mesma variável, iniciação e reiniciação erradas de variáveis, e falta ou implementação erradas da interligação lógica. Portanto, se o *software* desenvolvido for isento de erros e, realmente, responder as necessidades do caderno de encargos do projeto são eliminados a maior parte dos problemas de desenvolvimento dos referidos sistemas mecatrónicos.

1.2 Motivação

Existe uma crescente necessidade, por parte da indústria, de encontrar formas de estudar a interação entre as diferentes partes intervenientes num sistema mecatrónico e os seus programas.

Subsistem diversas formas de analisar este tipo de sistemas, sendo Simulação por MiL (*Model-in-the-Loop)* e Verificação Formal através de *Model Checking* duas das possíveis formas de fazer este estudo. Há quem defenda que a Simulação é a melhor forma de estudar sistemas mecatrónicos, pois permite verificar se o código desenvolvido realmente executa aquilo para que foi desenvolvido e, caso isso não se verifique, realizar alterações de acordo com as necessidades. Contudo, apenas uma parte do domínio dos comportamentos possíveis é testada.

Por outro lado, as técnicas de Verificação Formal por *Model Checking* permitem averiguar se o sistema desenvolvido responde as especificações do projeto em todo o domínio de comportamentos possíveis do sistema. Esta técnica permite desde logo garantir que o sistema nunca alcance um estado em que encrava, e é a única forma conhecida de garantir que o código esteja isento de erros [3]. Contudo é necessário utilizar lógica que é considerada difícil de utilizar e compreender [4].

Neste trabalho pretende-se desenvolver modelos para o comportamento dos blocos funcionais da norma IEC 61 131-1 tendo em vista uma abordagem que combina as vantagens das técnicas de Simulação por MiL com as de Verificação Formal através de *Model Checking* utilizando os mesmos modelos, para se conseguir uma análise mais cuidada da especificação de comando de sistemas mecatrónicos.

Antes de ser possível a aplicação de qualquer metodologia, é necessário converter o comportamento de todas as partes intervenientes num sistema mecatrónico para um formalismo apropriado. Como a maior parte deste sistema é automatizado através de microcontroladores, o seu comportamento é discreto, podendo se traduzido por DES (*Discrete Event Systems*) [5]. Para que os resultados obtidos nestas técnicas sejam possíveis de se transpor para a indústria, a correspondência dos modelos desenvolvidos com os comportamentos de todas as partes intervenientes toma extrema importância. Até a realização deste trabalho, nem todos os comportamentos importantes se encontravam modelados de forma adequada, limitando muito os estudos possíveis de serem realizados, porque não era possível converter uma boa parte dos comportamentos de controlador em modelos.

1.3 Objetivos

O objetivo desta dissertação prende-se com a criação de modelos dos blocos funcionais da norma IEC 61 131-3 que até agora não estão implementados numa metodologia abrangente de análise de sistemas mecatrónicos e seus programas, que envolve Simulação através de MiL e Verificação Formal

por *Model Checking* sobre os mesmos modelos. Está necessidade é fruto da existência de uma lacuna nos modelos das POU (*Program Organization Units),* que são unidades fundamentais de um programa de um PLC (*Programmable Logic Controller*), de acordo com a norma IEC 61 131-3 [6]. Sem as POU, uma série de sistemas não pode ser estudada, pois não é possível modelar o código implementado nos seus controladores. Existem três tipos de POU [7], mas os mais importantes são os blocos funcionais, por apresentarem comportamentos que influenciam as respostas dos PLCs,

Antes de realizar os modelos dos blocos funcionais da norma IEC 61 131-3, é necessário estudar os seus comportamentos e compreender como estes respondem, e selecionar os que correspondem a comportamentos necessários, sendo portanto este o primeiro objetivo. Os blocos considerados pertinentes para estudar são:

- *Time on delay* (TON);
- *Time of delay* (TOF);
- *Rising edge* (Flanco ascendente);
- *Falling edge* (Flanco descendente);
- *Up Counter* (Contador incremental);
- *Down Counter* (Contador decremental);
- *Up-Down Counter* (Contador incremental e decremental).

Constatou-se também que é necessário muito tempo para desenvolver os modelos essenciais à análise de um sistema mecatrónico, sendo importante desenvolver modelos que sejam reutilizáveis. Será também discutido: o porquê de se considerar como controladores apenas os PLCs e as suas vantagens, as formas para desenvolver programas de PLC e justificar a utilização de autómatos finitos temporizados e UPPAAL.

Tem-se também como objetivo testar a resposta dos modelos dos blocos funcionais desenvolvidos através da realização de um caso de estudo, para depois se poder validar os modelos acrescentados.

1.4 Organização da dissertação

O texto desta dissertação encontra-se num formato que melhor permite atingir os objetivos delineados. Numa primeira fase, é feito um levantamento de trabalhos anteriores, cujo âmbito se enquadra nesta dissertação e são analisadas as diversas hipóteses que podem ser consideradas. Depois,

é apresentado o trabalho já existente e aquilo que é necessário ser acrescentado, tendo em consideração as diferentes técnicas de Simulação.

A segunda parte desta dissertação concentra-se na apresentação do caso de estudo, utilizado para verificar os modelos dos blocos funcionais desenvolvidos.

No capítulo 2, faz-se um levantamento dos trabalhos, cujo seu âmbito é semelhante ao desta dissertação, e as considerações efetuadas na metodologia aplicada neste estudo.

No capítulo 3 são apresentados os modelos já considerados e os comportamentos que são necessários acrescentar; os formalismos utilizados e as considerações precisas para converter estes para o s*oftware* de Simulação e verificação; concluindo-se o capítulo com uma comparação entre as técnicas de Simulação de sistemas mecatrónicos, dando-se especial atenção à técnica utilizada.

Por sua vez, no capítulo 4, é apresentado o caso de estudo, os modelos desenvolvidos na parte física e no controlador, tendo em consideração especial, os modelos desenvolvidos para o comportamento dos blocos funcionais.

No capítulo 5 são discutidos os resultados do ponto de vista de Simulação numa primeira fase e posteriormente Verificação Formal.

Por fim, o capítulo 6 apresenta as conclusões gerais da dissertação e propostas de desenvolvimento de trabalhos futuros neste domínio.

A RETER DESTE CAPÍTULO

A indústria tem uma crescente necessidade de desenvolver especificações de comando de sistemas mecatrónicos seguras. Neste trabalho pretende-se desenvolver modelos dos blocos funcionais da norma IEC 61 131-3 para TA, tendo em vista uma metodologia abrangente de Simulação por MiL e Verificação Formal por Model Checking do software de comando de sistemas mecatrónicos utilizando os mesmos modelos.

Capítulo 2
TRABALHOS RELACIONADOS E HIPÓTESES CONSIDERADAS

Este capítulo tem como objetivo apresentar as diferentes abordagens para analisar sistemas mecatrónicos, dando mais ênfase às técnicas de Simulação e Verificação Formal por Model Checking. Apresentando-se as limitações de cada técnica, as razões que levaram a considerar PLC como o tipo de controlador utilizado e as formas de programação destes autómatos, selecionando a mais adequada para ser aplicada nesta dissertação. Aborda-se ainda os motivos da utilização de autómatos finitos temporizados, UPPAAL e TCTL (Timed Computation Tree Logic).

2. TRABALHO RELACIONADO E HIPÓTESES CONSIDERADAS

Existem muitas equipas de trabalho a desenvolver pesquisa no estudo e análise dos sistemas de controlo e respetivo *software*, pois subsiste a necessidade na indústria de garantir que não existem erros nas especificações de comando dos, cada vez mais complexos, sistemas mecatrónicos. Isto deve-se ao aumento exponencial do poder de processamento dos microcontroladores, permitindo assim que os sistemas utilizados sejam mais complicados, tornando o processo de desenvolvimento cada vez mais um desafio.

Podem considerar-se três formas de análise deste tipo de sistemas, sendo a primeira delas o teste sobre a implementação. Nesta técnica o controlador é isolado propositadamente e são introduzidos sobre este uma série de *inputs*, sendo depois analisadas as suas saídas para garantir que reage de acordo com as necessidades do sistema. Esta técnica tem como vantagem o teste ser feito sobre o equipamento real e, portanto, muito próximo da realidade, mas sem haver destruição da parte física do sistema mecatrónico. Contudo, apenas uma parte dos subestados do controlador é analisada e é necessário equipamento específico.

Outra forma possível é Simulação que é considerada uma das melhores formas de garantir que o sistema se comporta de acordo com o especificado. Esta metodologia é bastante útil quando os sistemas apresentam um tamanho considerável, contudo, apenas uma parte dos subestados do comportamento dos controladores é testada.

Por fim, a Verificação Formal que é um método que garante que todo o espaço de estados das possíveis evoluções do sistema é testado. Esta abordagem baseia-se em métodos formais que são linguagens matemáticas, técnicas e ferramentas para especificar e verificar sistemas. Especificar um sistema automático é descrever, numa linguagem matemática, o seu comportamento, e verificar é o passo seguinte, e consiste em provar formalmente que o modelo criado anteriormente satisfaz as especificações [8]. Para sistemas onde existam muitos estados concorrentes, esta análise pode levar muito tempo a ser realizada. Uma das formas de aplicar esta metodologia é através de *Model Checking,* que é uma técnica automática de verificação de estados finitos de sistemas concorrentes [9]. Basicamente existe um sistema de transições (M) e uma fórmula temporal (p), e o verificador testa se "M" verifica "p". O maior problema desta metodologia verifica-se em sistemas em que existam muitos componentes que podem interagir entre si sucedendo um número enorme de estados globais, sendo

por isso necessário grande espaço na RAM (*Random Access Memory*) do computador que realizar a análise.

2.1 Trabalho similar

Para esta dissertação considerou-se apenas relevante trabalhos que aplicam a técnica de Simulação e Verificação Formal por *Model Ckecking,* cujo seu âmbito se enquadra no mesmo deste trabalho.

2.1.1 Simulação

Existem diversas formas de simular um sistema mecatrónico, sendo um dos trabalhos pioneiros nesta área apresentado por Baresi em 1997 [10], que se baseia em criar modelos do controlador digital em FBD (*Function Blocks Diagram*) e a parte analógica do sistema no ambiente fornecido pelo Matlab/Simulink [11]. Os modelos desenvolvidos em FBD são convertidos automaticamente para redes de Petri temporizadas. Quando existe a necessidade de adicionar um modelo novo com determinado comportamento, este pode ser desenvolvido em redes de Petri ou em linguagem C, sendo reutilizável posteriormente.

A referência [12] apresenta uma metodologia de modelação baseada GI-SIM (GRAI/IDEF-*Simulation*) que foi desenvolvida para satisfazer as necessidades de análise e *design* de sistemas de controlo, combinando os três conceitos de modelação, considerados pelos autores deste trabalho relevantes (conceptual, funcional e simulação), numa única estrutura. Utilizam IDEF0 (*Icam DEFinition for Function Modeling*), que é um formalismo gráfico desenvolvido para funções de produção, que oferece uma linguagem funcional para análise, desenvolvimento e reengenharia de diversas áreas, entre as quais a engenharia de *software*. Para desenvolver os modelos foi criada uma aplicação, e para validar os resultados foi desenvolvido um caso de estudo.

Por outro lado, um grupo de trabalho [13], Korea Adwnced Institute of Science and Technology, propôs uma ferramenta chamada JR-net (*Job Resource relation-net*), que é um instrumento para modelação de sistemas de monotorização mais ou menos complexos. Este formalismo da JR-net suporta três fases de modelação: a modelação do objeto, o fluxo de trabalho e o modelo do supervisor de controlo.

Existe também trabalho desenvolvido, descrito pela referência [14], esta apresenta uma técnica para Simulação e verificação visual, que tem como ponto de partida o código escrito em LD (*Ladder Diagram*), uma das linguagens presente na norma IEC 61 131-3 [6]. Utilizam o formalismo de modelação

autómatos de estados finitos [15] e apresentam modelos 3D para visualizar os resultados que foram por estes desenvolvidos.

Existem algumas investigações que procuram conjugar as capacidades informáticas de hoje em dia para desenvolver automaticamente os modelos do código de controlo de um sistema mecatrónico. No caso do trabalho proposto por Barth e Fay [16] é apresenta uma técnica para automaticamente gerar modelos para Simulação através de aplicações CAE (*Computer Aided Engineering*). Existem também alguns *softwares* comercias como o Arena [17], um dos maiores na área da Simulação e existe há mais de 30 anos. DES são modelados através de uma série de eventos no tempo. Esta aplicação apresenta uma vasta gama de modelos que podem ser utilizados para construir o seu sistema sem necessidade de programação particular e permite a análise estatística e a utilização de modelos em 2D e 3D. Por outro lado, AutoMod [18] é uma ferramenta líder de Simulação gráfica de *software* que fornece demonstração fiel à escala 3D das suas operações de fabrico e distribuição. É a única ferramenta de Simulação no mercado que pode modelar sistemas de manufatura e automação de grande complexidade, podendo estar em funcionamento ou em fase de planeamento. Estas aplicações escrevem vagamente a lógica do controlo e portanto não são ideais para testar a especificação de comando [19].

Existe uma série de trabalhos [19,20,21] que utilizam o mesmo formalismo no processo de criação de modelos, nomeadamente DEVS (*Discrete Event System Specification*) [22], que é uma linguagem universal criada para especificar DES, é adequada para descrever comportamentos em que as entradas, estados e saídas são constantes, e as transições são identificadas como eventos discretos. Existe um esforço, nestes estudos, no sentido de reduzir o tempo necessário para simular um sistema, desde logo [19] propõem uma técnica inversa onde são utilizados os dados do histórico dos sinais e a tabela de sinais de entrada e saída do PLC já aplicado a um sistema mecatrónico para depois desenvolver os modelos para a Simulação. Por outro lado, em [21] constatou-se também a preocupação de reduzir o tempo do desenvolvimento dos modelos, mas desta vez utiliza-se uma técnica baseada em *templates*.

2.1.2 Verificação Formal por Model Checking

Moon em 1992 [23] propôs uma técnica de estudo da segurança (*safety*) e operabilidade, de sistemas químicos de controlo discreto. Esta técnica envolve; a criação de um modelo do processo e seu *software*; afirmações descritas em lógica que expressam questões sobre o comportamento do sistema, no que diz respeito a segurança (*safety*) e operabilidade e, por fim, um verificador de modelos (*Model Checker*) que testa se o modelo do sistema respeita as afirmações desenvolvidas anteriormente. Como

se pode ver na Figura 1 a esquematização desta técnica. O comportamento do equipamento, das operações e controlador (código de um PLC em LD) eram modelados através de STG (*State Transition Graph*) [24]. Para especificar as afirmações utiliza-se CTL (*Computation Tree Logic*) [25].

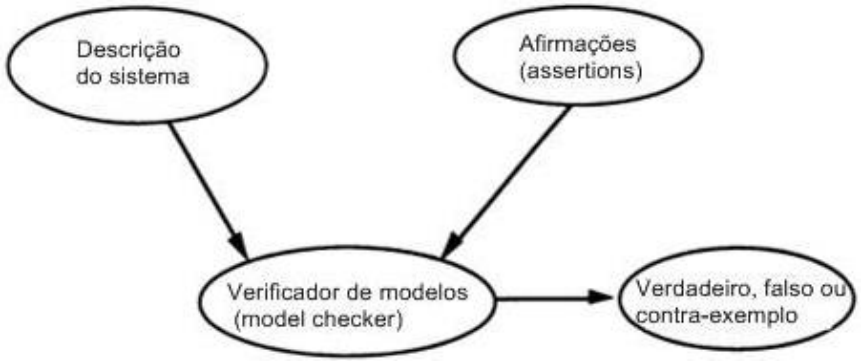

Figura 1 Princípio aplicado por Moon [adoptado de 23]

Esta metodologia de modelação não considera atrasos (tempos) entre os diferentes estados, o que não permite estudar uma parte importante dos sistemas de controlo. Além disso, a especificação das afirmações é muito dependente da interpretação dos utilizadores e da sua experiência, o que pode levar a erros sérios na especificação de comportamentos dos sistemas.

Este método deu origem a outros muito semelhantes, que se diferenciam no formalismo utilizado para especificar o comportamento do sistema, a forma como são especificadas as propriedades a estudar e aplicação/ambiente utilizado. Por exemplo em [26], que aplica um método alternativo, utilizando para especificar o comportamento das diversas partes do sistema a verificar a linguagem formal TRIO [27], que é um formalismo possível de ser usado para descrever sistemas de manufatura flexível. Este formalismo tem como característica ser uma linguagem temporal de primeira ordem que aceita considerações de tempo discreto e denso. As fórmulas desenvolvidas em TRIO são constituídas por operadores, conectores e quantificadores de primeira ordem e um único operador de modelação chamado de *Dist* que relaciona o tempo atual, que é deixado implícito na fórmula, com outro instante, conseguindo-se assim uma fórmula dependente de tempo. O objetivo da verificação utilizando TRIO é garantir que o sistema S satisfaz determinada propriedade R, ou seja S implica R. Para fazer a validação é utilizado o Zot *Model Checker* [28] que é uma ferramenta experimental de verificação por modelos em TRIO. Esta técnica tem a particularidade de as propriedades e comportamento do sistema de controlo serem modelados em lógica temporal. O trabalho referido considera a estrutura do programa de acordo com IEC 61 499 [29].

Por outro lado as referências [30,31,32] que apresentam a aplicação do mesmo princípio apresentado na Figura 1 em sistemas de produção flexível, de acordo com a norma IEC 61 499 [29], onde o código é escrito através da interligação de blocos funcionais e modelado no formalismo NCES (*Net Condition/Event systems*) [33]. Esta é uma extensão de redes de Petri [34], sendo o princípio de modelação de um sistema um conjunto de blocos com comportamento particulares e sua interligação através de sinais. Os benefícios desta linguagem são facilidade de modelação de sistemas distribuídos e ser muito intuitiva, além de se poder reutilizar os modelos quando se pretende representar o mesmo comportamento.

Para especificar as propriedades/afirmações a testar é utilizada CTL [25] e no processo de verificação dos modelos são utilizados as aplicações de *Model Checking* ViVe e SESA [35]. O objetivo deste trabalho prende-se com o teste contra segurança (*safety*), evolução (*liveness*) e funcionalidade e programas de PLC de acordo com IEC 61 449, pois a indústria apresenta grande relutância em aplicar a mesma norma nos seus sistemas de controlo. Normalmente, o modelo de *software* utilizado tem por base a norma concorrente IEC 61 131, consequentemente estas técnicas falham por ser pouco úteis para aplicar em sistemas industriais.

Em alguns trabalhos são apresentadas metodologias em que se considera modelos da parte física para analisar o comportamento do sistema, tal como acontece em [36]. Neste são estudadas as vantagens do uso do modelo da parte física na verificação, sendo que o programa escrito em SFC (*Sequential function Charts*) é convertido para IL (*Instruction List*). As propriedades são especificadas em CTL [25] ou LTL (*Linear Temporal Logic*) [37] e verificadas na aplicação NuSMV [38]. Em [39] é utilizado também NuSMV e é proposta uma metodologia de conversão de código escrito em FBD e a tradução para IL (*Instruction List*), que é depois convertida para o formalismo de entrada do NuSMV, seguindo-se a aplicação da metodologia referida anteriormente. Ainda o grupo de trabalho da referência [40] apresenta investigação utilizando o mesmo ambiente, mas como ponte de partida o código escrito em ST (*Structured Text*), sendo este depois convertido automaticamente através de uma aplicação protótipo para o formalismo do NuSMV. O trabalho desta equipa concentra-se em modelação de tempo de uma forma realista, pois como o tempo é representado de forma real implica também que este seja modelado de forma realística, para tal é apresentado um modelo para o bloco funcional TON (*Time ON delay*) definido pela norma IEC 61 131-3 [6]. Este modelo é composto por três estados possíveis, que correspondem aos três estados que o TON (Figura 15). Neste trabalho eles são chamados de NR (*Not Running*), TO (*Time Out*), R (*Running*), de acordo com a Figura 2. São utilizadas as mesmas variáveis

que o bloco e com o mesmo significado presentes nesta dissertação no capítulo 3.2.1, onde é descrito este bloco funcional.

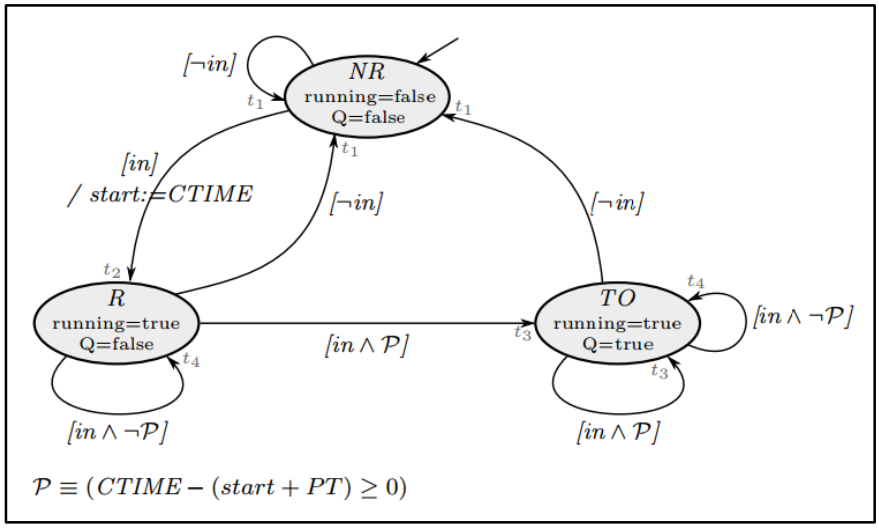

Figura 2 Modelo do TON aplicado no NuSMV [40]

Este modelo apresenta algumas características particulares para ser possível de ser interpretado pelo NuSMV, nomeadamente o tempo é representado por uma variável finita; o tempo é incrementado através da soma do tempo de ciclo do PLC; o tempo de ciclo é definido de forma não determinada.

Outra técnica é proposta por [41], mais uma vez de *Model Checking* baseada em modelos criados em BIP (*Behavior, Interactions, Priorities*) [42] onde o sistema é modelado em três camadas: Comportamento (*Behavior*), expresso por um conjunto de transições; Interações (*Interactions*) entre as diferentes transições; Prioridades (*Priorities*), usadas para escolher entre as interações. Neste artigo são apresentados modelos para as POUs (*Program Organization Unit*) definas pela norma IEC 61 131-3 [6], com especial atenção para o TON (Figura 3). A ferramenta utilizada para realizar a verificação formas é D-Finder [43], que permite detetar *deadlock* e verificar propriedades.

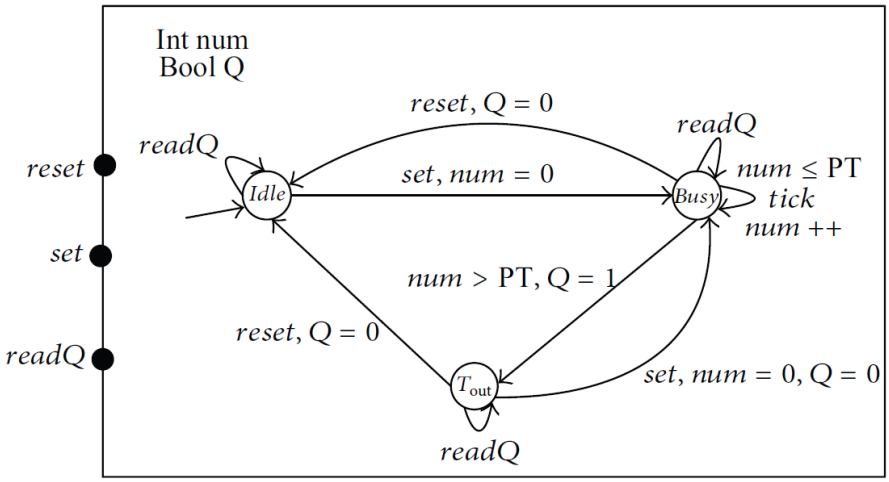

Figura 3 Modelo do TON em BIP [41]

Este modelo do bloco funcional (Figura 3), tal como no anterior, é composto por três partes, neste caso, são chamados de *idle* (inativo), *busy* (ocupado) neste estado do modelo o "num" (variável responsável por contar o intervalo de tempo), é incrementada em uma unidade sempre que recebe a ordem para atualizar. Quando "num" for maior que o intervalo de tempo "PT" (ver Figura 15), o modelo avança para o estado *Time out*, aqui a variável booleana de saída (Q) do bloco funcional atinge o valor lógico de um. Este modelo está de acordo com o diagrama apresentado na Figura 15 do capítulo 3.2.1, que apresenta o comportamento do bloco funcional TON, contudo este formalismo apresentado não permite constrangimentos de tempo o que não possibilita um estudo em que existam interações entre os tempos. No artigo [44] é modelado todo o sistema em BIP e depois convertido para TA [45] para poder estudar com maior rigor o sistema mecatrónico. Apesar da linguagem de modelação ser bastante intuitiva e fácil de utilizar, e existirem modelos de todas as partes importantes do sistema, esta não permite considerações de tempo, o que limita bastante o processo de Verificação Formal.

A referência [46] apresenta uma técnica em que são criados os modelos XML (*eXtensible Markup Language*) [47] a partir das especificações e simultaneamente do código do programa escrito em FBD, sendo depois estes convertidos em TA utilizando o UPPAAL TRON [48] para verificar se os dois modelos correspondem um ao outro, como é demonstrado na Figura 4.

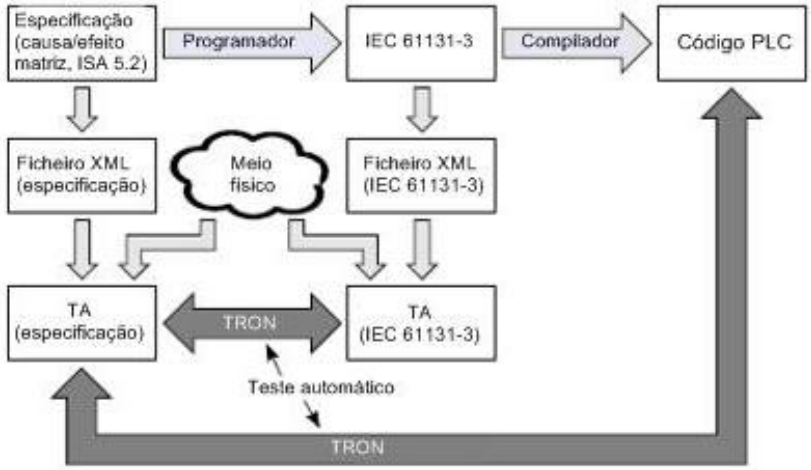

Figura 4 Metodologia de análise aplicada pela referência [46]

Esta técnica é útil pois não é necessário ter um PLC por perto para se verificar o seu código, pelo que em muitas situações pode ser vantajoso. Por outro lado, esta metodologia além de ser difícil de implementar pode ser limitadora em algumas situações. E a maior preocupação deste trabalho concentra-se sobretudo no programa do PLC.

Existem ainda trabalhos que aplicam uma técnica semelhante a anterior [49], mas o seu objetivo é criação de uma libraria dos blocos funcionais definidos pela PLCopen [50] e depois testar se esses mesmos modelos são de facto de acordo com o especificado pelo referido comité. Permitindo assim, a utilização destes modelos para Simulação e Verificação Formal em trabalhos futuros.

Por outro lado, [51] apresenta uma metodologia de Verificação Formal por *Model Checking* onde são considerados modelos quer para o controlador, quer para a parte física do sistema, tendo em atenção tempos. Nesta técnica são traduzidos programas descritos em SFC, uma das linguagens apresentada pela norma IEC 61 131-3, sendo posteriormente convertidos em TA [45] e a partir das especificações impostas pelo projeto são formuladas afirmações para testar nos modelos utilizando o formalismo TCTL (*Timed Computation Tree Logic*) [52]. Todo o processo desde da modelação até a Verificação Formal é realizado no ambiente do UPPAAL [53]. Pode-se ainda simular e verificar visualmente se os modelos reagem de acordo com o pretendido, devido às capacidades de Simulação de modelos do UPPAAL.

Os autores em [54] apresentam uma metodologia semelhante, utilizando a mesma técnica de modelação e especificação das afirmações a testar. Concentram-se em sistemas de malha fechada cujo seu comportamento pode ser expresso por DES, e que os estados dos modelos sejam o mais próximo possível dos sistemas reais. Este trabalho tem em atenção modelos para a parte física do sistema, o controlador e a sua interação, demonstrando que, em muitas situações, é vantajosa a utilização de variáveis para representar as entradas lógicas e as condições de estabilidade especificadas pelo controlador. Apresentam, também, modelos para o bloco funcional TON, mais uma vez considerando três estados, como demostrado Figura 5, neste caso os autores intitulam o estado inicial de "*OUT"*. O modelo está desativado nesse estado e ao receber a ordem para avançar e T_Xi_t1 estiver ativado avança para o estado "*RUN"*, onde é contado o tempo PT (ver capitulo 3.2.1). O terceiro estado nesta referência é chamado de "OK" e corresponde à situação em que o tempo PT foi atingido sem se deixar de registar T_Xi_t1 e, tal como nos anteriores, é ativado um sinal booleano.

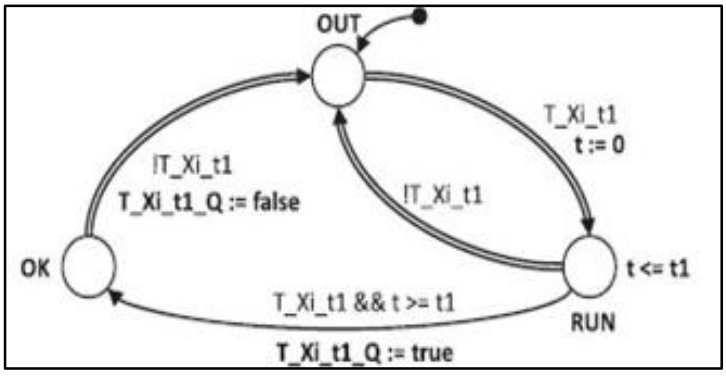

Figura 5 Modelo do TON em TA [54]

Por outro lado, [55] apresenta um método de conversão de programas escritos em FBD para TA, onde são considerados modelos para os elementos base de um programa escrito em FBD, particularmente blocos funcionais e funções. A sua técnica consiste em primeiro converter o programa de escrito em FBD para TA, e em seguida adicionar a sinemática do ciclo interno do PLC (*scan cycle*) e o ambiente em seu redor. Segue-se a interação entre os diferentes modelos e por fim são especificados os modelos dos elementos base de um código em FBD, nomeadamente, as funções e blocos funcionais, descrita na Figura 6.

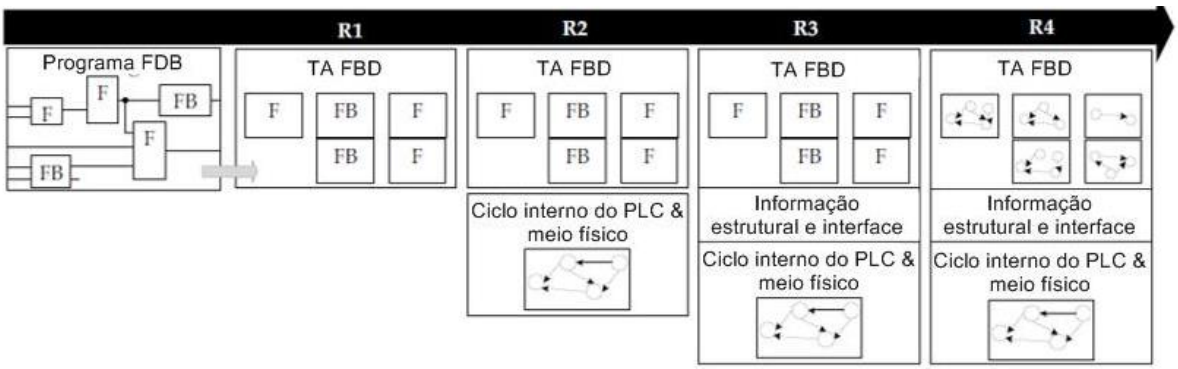

Figura 6 Metodologia de análise aplicada na referência [55].

2.2 Resultados obtidos

Existem muitas equipas de trabalho a desenvolver investigação nesta área, devido à necessidade que a indústria apresenta de encontrar uma forma de analisar os seus sistemas mecatrónicos. Apesar de nem todos se apresentarem como boas alternativas para utilização imediata, pois são demasiado académicos e/ou não têm em consideração as normas adequadas para serem transponíveis para a um ambiente industrial.

Considerando os trabalhos na área da Simulação, estes têm apenas em atenção o comportamento do programa sem dar especial cuidado ao comportamento da parte física. Noutros casos, pelo contrário, procuram simular a parte física sem dar especial atenção à lógica de controlo dos equipamentos. É importante ainda salientar o facto de não demostrarem a preocupação necessária às normas aplicáveis a estes equipamentos. Estas técnicas são portanto difíceis de ser transponíveis para a indústria. Os trabalhos que se concentram essencialmente em Simulação, como já foi referido, apenas analisam uma parte dos comportamentos dos sistemas, tornado a análise limitada. Para corrigir este efeito existe a preocupação de diversas equipas de investigação para tornar as técnicas de Verificação Formal por modelos aplicáveis e fáceis de utilizar.

Nos trabalhos de Verificação Formal por *Model Checking* existem muitos formalismos que não suportam considerações de tempo, pelo que estes apresentam grandes limitações nas propriedades que se pode testar. Por outro lado, existem investigações que seguem a norma IEC 61 499 que não é aceite pela indústria, tornando a possibilidade de aplicação das suas técnicas limitada. Diversas equipas de trabalho apresentam modelos para o FB (*Function Block*) TON, mas as restantes FB propostas pela norma IEC 61 131-3 apenas são alvo de estudo pelas referências [41,55], contudo estas não apresentam quais as considerações a ter para a sua conversão para o formalismo por eles utilizado e o método aplicado não dá a atenção necessária à parte física, pois é muito centrado na forma como é escrito o programa.

Não existe uma forma perfeita de analisar controladores indústrias, portanto para se conseguir uma análise mais cuidadosa procurou-se combinar as potencialidades de Simulação e Verificação Formal, para que não existam falhas quando os sistemas mecatrónicos em estudo sejam implementados.

2.3 Hipóteses consideradas para este trabalho

Na elaboração desta dissertação optou-se por uma metodologia quem tem como ponto de partida o caderno de encargos do projeto. Nesta abordagem, procura-se agrupar num mesmo modelo a parte física com estados fidedignos e modelos dos programas de controlo, também eles semelhantes aos sistemas reais, para assim se conseguir Simular e Verificar Formalmente toda uma gama mais ampla de sistemas mecatrónicos e melhorar os resultados destas análises (ver Figura 7).

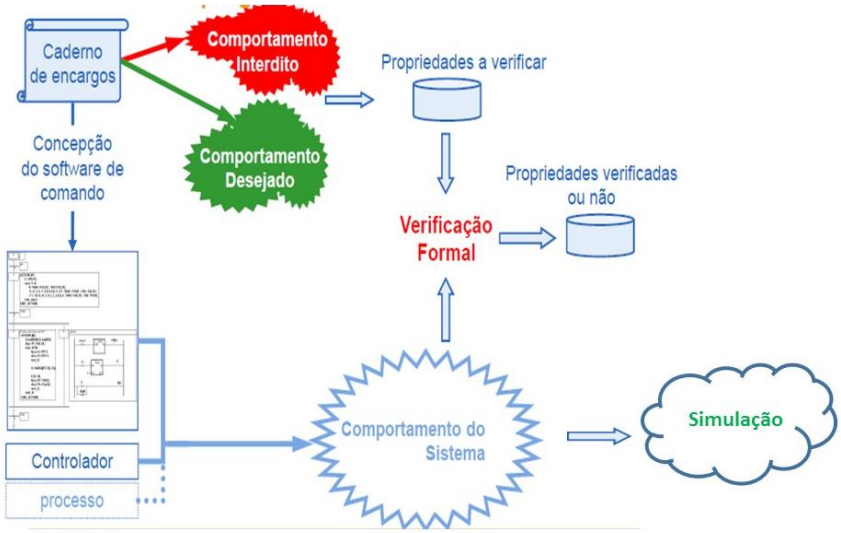

Figura 7 Princípio de análise aplicado a esta dissertação.

Após obtido o modelo do sistema, composto por vários submodelos dos comportamentos das diferentes partes constituintes de um sistema mecatrónico, este é sujeito a um processo de Simulação por MiL, conseguindo-se, desde logo, detetar pequenos erros que podem ter sido cometidos aquando da formulação do programa (ver Figura 7).

Terminada esta fase, os modelos serão sujeitos a uma série de "afirmações", que resultam de comportamentos que o sistema deve apresentar ou situações que nunca devem ocorrer, para garantir que o sistema mecatrónico e o código desenvolvido satisfazem as especificações do projeto em todos os estados do controlador, tal como descrito na Figura 7.

Este trabalho concentra-se na criação de modelos de programas de PLCs, que são o mecanismo de controlo mais utilizado pela maior parte dos sistemas de manufatura utilizados na indústria. Mas, existem algumas alternativas como por exemplo:

- Relés;
- Computadores indústrias;
- Computadores pessoais;
- Placas de Arduino.

Quando comparados com relés, os PLC permitem muito mais flexibilidade pois possibilitam realizar alterações apenas trocando o código lógico, afinal foi para isso que foram criados.

Computadores industriais são semelhantes aos tradicionais, mas têm especificações mais restritas em termos de viabilidade, os componentes são mais robustos, apresentam proteção contra fatores externos como calor, corrosão, humidade, vibração e interferência eletromagnética. Estes fatores tornam um computador industrial extremamente caro em comparação com um normal. Em pequenas empresas onde a aplicação não apresenta perigo para o operador, são utilizados os normais porque ainda é mais barato fazer a substituição e/ou reparo de um computador normal, do que comprar um industrial. A principal vantagem da utilização de computadores industriais é a capacidade de recolha e tratamento de dados ao mesmo tempo que se opera a máquina. Mas, quando comparado com PLCs, conclui-se que estes são:

- Mais fáceis de programar e instalar;
- A velocidade de processamento dos PLCs, normalmente é muito superior ao de qualquer sistema de controlo eletromecânico;

- Têm maior capacidade de suportar ambientes rígidos, tais como temperaturas ou humidade elevadas;
- São bastante mais baratos.

As placas de Arduino [56] são uma plataforma para a realização de protótipos eletrónicos de *hardware*, cuja sua licença de *software* é livre. Utiliza um microcontrolador com suporte de entradas e saídas embutidas e com uma linguagem de programação própria, mas que deriva de C/C++. O objetivo é criar uma ferramenta que seja acessível, de baixo custo, flexível e fácil de usar para aqueles que não tem acesso aos controladores mais sofisticados e às ferramentas mais complicadas. Este é muito usado em trabalhos académicos por ser uma ferramenta barata e permitir o treino de engenheiros, mas peca em termos de fiabilidade e segurança, que são requisitos importantes para aplicações industriais. Pelo contrário, os PLCs são pensados para ser robustos e fiáveis e é por isso que são tão utilizados.

2.3.1 PLC's

Os PLCs utilizam no processo de controlo das máquinas *inputs* (entradas) e *outputs* (saídas) sólidos em vez de relés de controlo [57]. Com o desenvolvimento dos microprocessadores permitiram que estes sistemas evoluíssem de simples substituição de relés para sistemas muito mais complexos existentes hoje em dia. Estes avanços tecnológicos, quer a nível do *hardware* quer do *software,* resultaram em uma melhor performance e dimensões mais reduzidas nos PLCs. Devido a isto, estão constantemente a aparecer novos mercados onde estes equipamentos podem ser aplicados.

É importante ainda referir que, apesar de grandes mudanças, os PLC continuam a manter as suas intenções originais, nomeadamente serem simples de manter e utilizar [58]. Atualmente, um PLC é considerado um sistema especial de microprocessadores utilizados para controlo, que usam uma memória programável para guardar instruções e para implementar funções lógicas, de sequência, de tempo, de contagem e aritméticas para controlar máquinas e processos.

O seu *design* é pensado para ser operado por engenheiros que, em certos casos, têm conhecimento limitado de computadores e linguagens de programação [59]. Existe um espectro muito alargado de PLCs, que vai desde sistemas para interligar computadores portáteis até sistemas onde o erro é intolerável e as únicas alterações são as suas especificações, como por exemplo, a velocidade de processamento. Todos os PLCs são compostos por uma estrutura interna semelhante [60], sendo esta composta por diversas partes, como é demostrando na Figura 8.

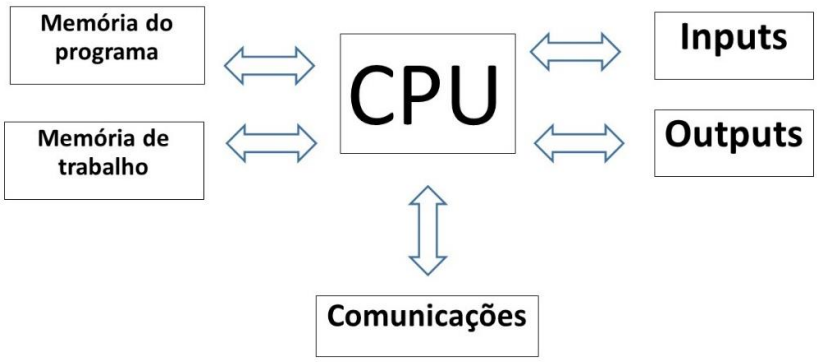

Figura 8 Estrutura interna de um PLC [adaptado de 60]

A CPU (*Control Processing Unit*) é o elemento central do PLC, responsável por monitorizar as entradas e alterar as saídas de acordo com o programa. Assim como, também é responsável por operações matemáticas e lógicas e outras funções especiais, como por exemplo, temporizadores e contadores.

A memória do programa está destinada para carregar o código a ser executado, inclui também alguma memória destinada para as operações de PLC. Tem uma bateria acoplada capaz de durar entre três a cinco horas, isso é importante porque se ocorrer uma falha de energia não terá de se carregar novamente todo o programa.

Memória de trabalho é o lugar onde se pode manter um registo atualizável das entradas/saídas e resultados das operações de PLC.

A função dos *inputs* (entradas) tem a responsabilidade de fazer a conexão entre o processo a ser controlado e a CPU. A sua principal tarefa é receber os sinais do processo, converte-los e formatá-los, para que a CPU possa usá-los. O número de entradas suportado é limitado pelo processador (CPU) e pela quantidade de memória disponível. A função dos *outputs* (saídas) é fazer o processo inverso da função dos *inputs*. Este módulo recebe sinais da CPU, transformando-os num formato adequado para que os atuadores possam utilizar as informações e manobrar o processo. Existe ainda um módulo de comunicações que permite que o PLC receba e envie informação para outros dispositivos, possibilitando assim a criação de redes industriais inteligentes.

Para que o PLC seja bem programado, é necessário definir corretamente todos os estados iniciais de todas as partes do sistema mecatrónico. Este após receber o *feedback* do processo, ou seja efetuar a leitura das entradas, é corrido o programa que calculará as saídas, que não são nada mais que as alterações necessárias para que a tarefa a ser controlada seja cumprida com eficácia [59] como demonstra esquematização da Figura 9.

Figura 9 Ciclo interno de um PLC

Este ciclo repete-se infinitamente, e quanto mais rápido for essa atualização mais preciso será o controlo.

2.3.2 Formas de programação de PLC's

Durante os últimos 30 anos foram apresentadas diversas formas de programação de PLC. Existem aplicações desenvolvidas nas mais diversas linguagens desde logo, Visual Basic, Fortran e Linguagem C, e muitas linguagens próprias como LD e SFC. A única coisa que tinham em comum era serem todas diferentes, consequentemente existia um desperdício muito grande de recursos no treino de técnicos e engenheiros para programar PLCs.

Felizmente, a comunidade industrial reconheceu o problema e propôs a criação de um grupo responsável por uniformizar a forma de programar este tipo de controladores, nomeadamente IEC (*International Electrotechnical Commission)* que foi criado em 1979 para estudar o referido problema. Esta, tal como a organização ISO (*International Organization for Standardization*) foi fundada em Genebra na Suíça e tem comités e grupos de trabalho formados por representantes da maioria dos países industrializados que põem em prática as normativas IEC.

Hoje em dia, existem duas normas concorrentes que podem ser seguidas para a programação de PLC, nomeadamente a IEC 61 131 [6] e a IEC 61 499 [29].

2.3.3 IEC 61 131

Antes de existir a IEC 61 131, diferentes marcas de PLC ofereciam diferentes tipos de comandos que podiam ser aplicados nos processos, e estes comandos muitas vezes não eram transponíveis de uma marca para outra o que tornava a tarefa de tradução de um programa um exercício bastante complexo ou até mesmo impossível. Então, em 1992 surgiu a IEC 61 131, que tinha como objetivo

tornar ao máximo o código de um PLC reutilizável, ou seja, que todas as marcas fossem programadas de acordo com esta norma [61]. Existiram diversas normas antes desta, mas eram todas diferentes de país para país o que as tornavam pouco eficazes.

Foram publicadas diversas partes da norma que dizem respeito às diferentes partes do IEC 61 131 que são [6,7]:

- Parte 1: uma visão geral com definições;
- 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 em PLCs;
- Parte 7: programação com controlo através de lógica *fuzzy*;
- Parte 8: guias de aplicação e implementação das linguagens de programação;
- Parte 9: interface de comunicação digital para pequenos sensores e atuadores.

Para a presente dissertação, a parte 3 será a obra de maior estudo, pois apresenta o modelo de *software* mais aceite pela comunidade industrial para a criação de programas de PLCs. Esta norma define cinco linguagens de programação de PLC que são:

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

Destas linguagens duas delas são textuais e as outras três são linguagens gráficas. Aquando do seu desenvolvimento estas foram pensadas para ser utilizadas por engenheiros com poucos conhecimentos na área da informática. Estas não apresentam qualquer conflito entre elas, permitindo a sua interação, ou seja, podemos ter a parte do programa principal escrita em SFC e, por exemplo, um bloco funcional escrito em IL.

Quando é possível usar diferentes linguagens para a criação dos programas, deve-se escolher a melhor para cada situação [62]. Se pretender uma linguagem universal e/ou fácil de modificar a melhor escolha é LD. Em aplicações que seja necessário operações matemáticas complexas e/ou o

programador é um engenheiro jovem a melhor escolha é ST. Quando a aplicação é para ser utilizada sobretudo na europa e/ou o objetivo é ter velocidade máxima de execução, pode optar-se entre IL ou ST. No caso de o responsável pela manutenção do programa for o utilizador final e/ou o programa necessitar de realizar varias ações em simultâneo, a melhor opção é SFC. Para projetos de grandes dimensões onde exista a necessidade de diversas equipas a trabalhar em simultâneo, em diferentes partes do código, deve-se utilizar FBD, pois cada bloco funcional pode ser escrito em qualquer uma das outras quatro linguagens.

2.3.4 IEC 61 449

Esta norma surgiu para responder às limitações apresentadas no estudo exposto pela universidade de Iacocca [63], onde se define que um sistema de controlo aplicado a um sistema de manufatura do século XXI deve apresentar as seguintes caraterísticas:

- Portabilidade: ferramentas de *software* podem ler e interpretar de forma correta elementos de *software* e configurações de sistema produzidos por outro vendedor;
- Interoperabilidade: ou seja dispositivos de diferentes vendedores podem cooperar uns com os outros para se conseguir as funções necessárias para aplicações distribuídas;
- Configurabilidade: as saídas de um dispositivo são compatíveis com as entradas de outro, mesmo que esses sejam de diferentes vendedores;
- Reconfiguração: capacidade de em tempo real fazer alterações no *hardware* e *software*;
- Distribuição: a capacidade de distribuir componentes de *software* entre diferentes dispositivos.

Pode ser dito que a norma IEC 61 449 propõe uma linguagem de base para o *design* a nível do sistema para monotorização e controlo de sistemas que tenta resolver as lacunas entre a linguagem de programação de PLC e sistemas distribuídos. De acordo com o modelo proposto pela norma, o sistema distribuído consiste num dispositivo equipado com interfaces para o ambiente do programa. O elemento universal da IEC 61 499 é o bloco funcional, este pode ser usado para descrever lógica de controlo descentralizado, mas também para descrever as propriedades do dispositivo, nomeadamente as suas interfaces [64].

2.3.5 IEC 61 131-3 vs IEC 61 449

A IEC 61 131-3 é considerada um dos mais importantes normas da automação industrial, mas nos últimos anos tem vindo a ser criticada por já não responder às necessidades de um sistema complexo de hoje em dia. Para responder a estas restrições, a IEC criou um comité para tentar resolver o problema que desenvolveu a norma IEC 61 449.

Os impulsionadores desta norma [65] defendem, mais especificamente, que requisitos de flexibilidade, adaptabilidade e robustez não são corretamente resolvidos pelos métodos atuais e que a IEC 61 449 vem para resolver o problema. Contudo em termos indústrias, ainda se encontra nas fases iniciais de desenvolvimento de expansão e realização prática [66], isto é mesmo admitido pelos seus próprios apoiantes [65]. A norma IEC 61 449 é ambígua, pelo que diferentes implementações fizeram desiguais suposições o que resulta que a mesma aplicação se comporte distintamente em diferentes implementações [67], atualmente não existe suporte para que os códigos sejam portáveis [68].

Por outro lado é defendido por [69] que a norma IEC 61 131-3 foi encontrando soluções para responder aos requisitos apresentados pelo estudo de Iacocca. No que diz respeito à portabilidade, esta normativa apresenta soluções utilizando FBD, SFC e ST para que não se perca o código quando este é transposto de um dispositivo para o outro. Entretanto, a PLCopen [50] apresentou um formato de exportação universal, o XLM [47], que foi aceite pela indústria.

No que diz respeito à interoperabilidade, foi desenvolvido uma norma para comunicações nomeadamente o IEC 61 131-5, e protocolos de comunicação foram fortalecidos, tais como Modbus e TCP (*Modbus transmision control protocol*). Atendendo à configurabilidade foram criados formatos de dados e configurações uniformes para os barramentos de campo. Do ponto de vista de reconfiguração em muitas aplicações, esta não é precisa, mas quando esta é necessária em grandes sistemas é perfeitamente eficaz. E por fim distribuição, que foi resolvido pelo bloco funcional de comunicações apresentado pela norma IEC 61 131-5.

Na mesma linha de raciocínio, a referência [68] demostra que a norma IEC 61 499 não pode ser considerada um sucessor efetivo da IEC 61 131-3, nem se demostrou capaz de ser uma alternativa viável pelo menos considerando a versão atual. Conclui-se então, que apesar do conceito de programação distribuída da IEC 61 449 ser bastante relevante, a sua aplicação prática é muito reduzida. Por outro lado, a IEC 61 131-3 está completamente testada e implementada industrialmente e já existem engenheiros treinados com as suas linguagens e a realização da troca de sistemas traria custos muito elevados a nível de equipamentos e treino dos programadores. Por isso, considera-se o modelo de

programas apresentado pela IEC 61 131-3 para aplicar nesta dissertação, por se apresentar como a alternativa mais viável e porque foi evoluindo para responder às necessidades que a indústria lhe foi apresentando ao longo das décadas.

2.3.6 Autómatos finitos temporizados

Para a modelação de DES pode ser utilizado um grupo grande de formalismo, desde logo redes de Petri, BIP, NCES, entre outras, como foi demostrado no capítulo 2.1, mas nem todos permitem a utilização de considerações de tempo representadas de forma real. Pelo que, nesta dissertação optou-se por utilizar TA. Este foi proposto por Alur e Dill [45] em 1990, é um formalismo matemático para modelar o comportamento de sistemas de estados finitos não sincronizados, cujo tempo tem um valor real que chamaram de Timed Buchi Automata. Basicamente, é um autómato combinado com mecanismos para expressar diferenças de tempo entre diferentes eventos. Os sistemas de tempo real são modelados por um conjunto de autómatos temporizados, cada um deles representa um comportamento dentro do sistema. Este facto facilita o processo de construção de modelos de sistemas com comportamentos concorrentes e/ou complexos, tornado este formalismo ideal para a modelação de sistemas de controlo, cujo seu programa foi devolvido em SFC. Porque este formalismo suporta a modelação de comportamentos em paralelo. Se a modelação for pensada para o efeito os modelos podem ser reutilizados de forma semelhante a dos FBs, em que apenas é necessário definir quais os seus *inputs* e *outputs* para cada situação, sem ser necessário fazer qualquer alteração no seu interior, reduzindo-se assim o tempo despendido na elaboração dos modelos e possíveis erros na sua elaboração (muitas vezes são difíceis de detetar).

Outra das razões que leva a utilização deste formalismo é a existência de uma ferramenta (UPPAAL) que permite realizar sobre o mesmo modelo de Simulação por MiL a Verificação Formal por *Model Checking* pois alguns autores defendem que a melhor maneira de analisar um controlador industrial é a Simulação, porque os formalismos de especificação das afirmações nas técnicas de *Model Checking* são difíceis de ser especificadas [4]. Mas, quando se executa unicamente a Simulação, só uma parte dos estados do controlador é testada. Portanto, neste trabalho, reúnem-se as vantagens das duas técnicas para uma análise mais cuidadosa.

2.3.7 UPPAAL

A utilização do UPPAAL, como já foi referido, permite Simulação e Verificação Formal sobre o mesmo modelo, garantindo, desde logo, a possibilidade de os testar de uma forma muito mais intensiva. Além disso, o processo de modelação para um utilizador familiarizado com o *software* é bastante simples, podem ainda fazer-se comparações entre os resultados da Verificação Formal e Simulação. Essas comparações são mais realistas, pois são efetuadas sobre os mesmos modelos.

O UPPAAL é composto por 3 áreas de trabalho, nomeadamente uma janela de edição, onde são criados os modelos, um separador de Simulação, onde estes são testados, possibilitando verificar se de um modo geral o sistema se comporta como é pretendido. Existe ainda um último separador que tem implementado um investigador de Verificação Formal, que permite testar se as propriedades definidas pelas especificações de projeto se verificam ou não. O princípio da ferramenta é modelar o sistema em TA [45] para depois se simular e verificar-se as propriedades previamente escritas em TCLT [52] satisfazem o modelo criado. Podendo testar-se, por exemplo, se um estado é acessível ou não (propriedade de acessibilidade), isto é, uma procura exaustiva que cobre todas as possibilidades de comportamento dinâmico do sistema [70].

2.3.8 *Timed Compution Tree Logic*

CTL (*Compution Tree logic*) é uma lógica de especificação ramificada, o que significa que o tempo é especificado numa estrutura de árvore em que o futuro não é determinado, pois existem caminhos diferentes que podem ser realizados [71]. Esta lógica é muito usada para Verificação Formal por *Model Checking* em *software* e sistemas de manufatura inteligente. Mas, como o formalismo utilizado para desenvolver os modelos neste trabalho permite representação de tempo real, é necessário uma lógica que permita também considerações de tempo para especificar as propriedades, para assim se conseguir uma análise mais cuidada. Optou-se, então, por uma extensão da CTL com operadores quantificadores de tempo, como por exemplo “E<> < 5” , o que significa possível dentro de 5 unidades de tempo. Resultando numa CTL temporizada que chamaram de TCTL (*Timed Compution Tree Logic*) [52].

Por outro lado, existem ferramentas como o UPPAAL, que tem algoritmos implementados que permitem verificar se os modelos em TA respeitam determinada propriedade expressa TCTL. Esta é uma grande vantagem quando se pretende analisar sistemas mecatrónicos e os seus programas, pois

trabalha-se sempre no mesmo ambiente e sobre os mesmos modelos. Tornando-se, portanto, vantajoso a utilização desta lógica temporal na elaboração desta dissertação.

Em suma, nesta dissertação é considerada uma metodologia de análise de controladores industriais que procura combinar as vantagens de Verificação Formal por *Model Checking* e Simulação em MiL, utilizando para descrever os comportamentos do sistema mecatrónico o formalismo autómatos finitos temporizados na ferramenta UPPAAL, tal como é demonstrado na Figura 10.

Novamente de acordo com a Figura 10, é desenvolvido a partir do caderno de encargos, o *software* de comando, utilizando SFC, uma das cinco linguagens da IEC 61 131-3. Este código é depois implementado no modelo do programa do PLC, juntamente com os modelos dos comportamentos de todas as partes intervenientes, como por exemplo, sensores e atuadores. Seguidamente, o modelo global é simulado, onde é possível testar se o sistema reage de acordo com o pretendido e, caso contrário, realizar as devidas alterações nas instruções de comando.

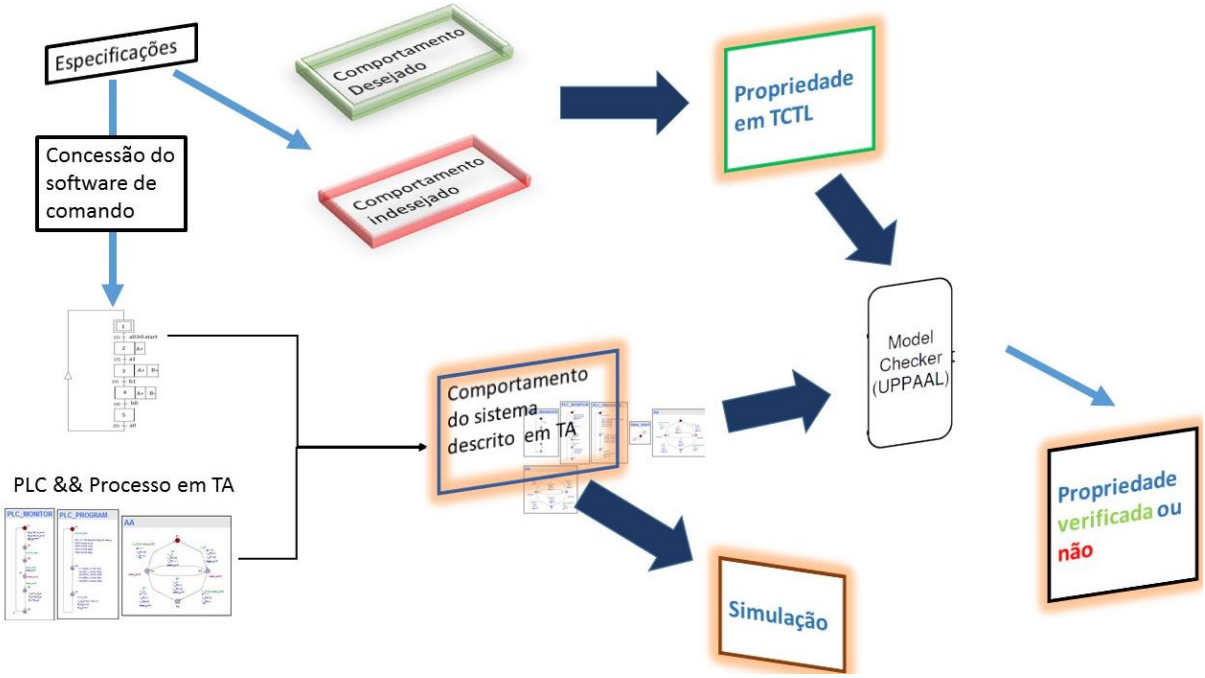

Figura 10 Considerações aplicadas nesta dissertação

Paralelamente e, mais uma vez, tendo como ponte de partida o caderno de encargos, são desenvolvidas propriedades em TCTL para verificar no modelo, estas não são mais que comportamentos desejados e situações que não devem ocorrer (ver Figura 10). O *Model Checker* do UPPAAL testa sobre o modelo anteriormente simulado se as propriedades são respeitadas ou não. Caso estas não estejam presentes nos modelos, o código do programa deve, mais uma vez, ser corrigido, repetindo-se até que as referidas propriedades estejam todas presentes no modelo.

A RETER DESTE CAPÍTULO

Existe um grande número de equipas de trabalho a desenvolver investigação na análise de controladores industriais, mas nem todos consideram as normas aplicadas a estes equipamentos ou o formalismo não suporta constrangimentos de tempo. Constatou-se a existência de uma lacuna no que diz respeito aos modelos dos blocos funcionais da norma IEC 61 131-3. Foi considerado para este trabalho sistemas controlados utilizando PLC, norma IEC 61 131-3 e autómatos finitos temporizados, como formalismo utilizado para desenvolver os modelos, devido às suas elevadas vantagens, nomeadamente, suportar análises sobre tempo representado de forma real e a existência de uma ferramenta capaz de simular e verificar formalmente os modelos desenvolvidos nesse formalismo (UPPAAL).

Capítulo 3

SIMULAÇÃO E VERIFICAÇÃO FORMAL

Este capítulo tem como objetivo apresentar os modelos que já são considerados nas tarefas de Simulação e Verificação Formal, e o que será acrescentado com a realização deste trabalho, nomeadamente modelos para os blocos funcionais da norma IEC 61 131-3 considerados relevantes. Apresenta-se ainda os formalismos utilizados para especificar os comportamentos dos sistemas e as propriedades, as considerações necessárias para converter estes formalismos no UPPAAL e, por fim, as técnicas de Simulação concorrentes.

3. SIMULAÇÃO E VERIFICAÇÃO FORMAL

3.1 O que já é considerado

No processo de modelação já são considerados modelos para o comportamento de um PLC, que são feitos através da criação de dois autómatos temporizados, que interagem entre si de acordo com o apresentado na referência [5].

No que diz respeito à parte física, os modelos a considerar podem ser muito vastos. Neste estudo considerou-se um modelo geral que pretende simular o comportamento de um cilindro e o seu respetivo sensor, como se pode ver Figura 11.

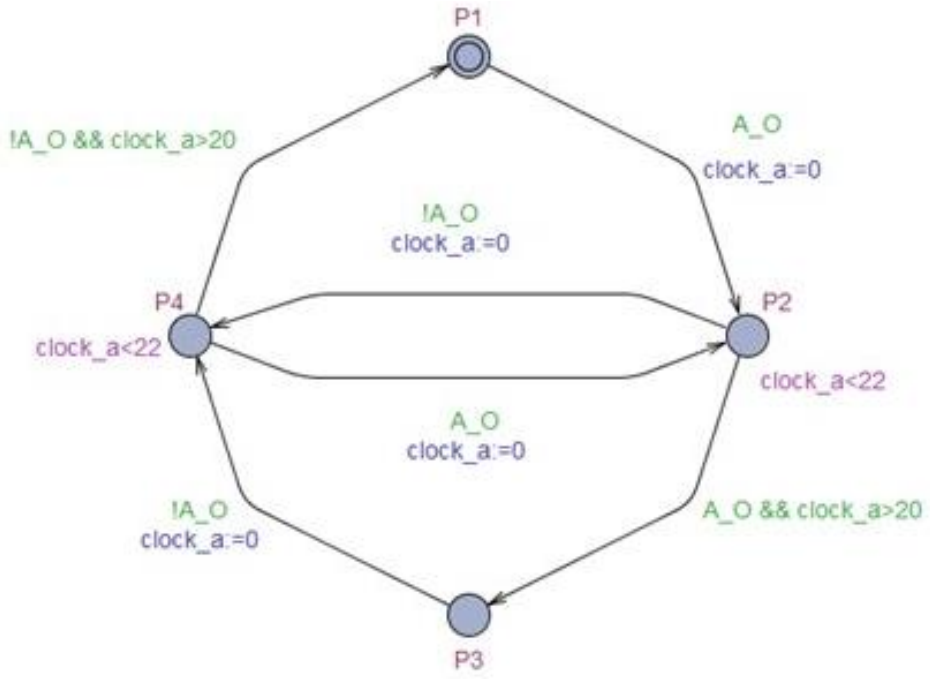

Figura 11 Modelo em TA do cilindro e sensor.

O modelo da Figura 11 é um elemento base utilizado para representar os atuadores, este é composto por quatro estados possíveis que correspondem aos comportamentos que um cilindro pode apresentar, como se pode ver na Figura 11. O lugar (*locations*) "P1" corresponde ao cilindro recuado e a "P3" ao cilindro avançado. "P2" representa o cilindro a avançar, tendo dois comportamentos modelados:

- Se a ordem de avanço (cilindro mono estável) do controlador deixar de existir, o modelo atinge o lugar "P4", que corresponde a ação de recuo;

- Se a ordem se manter tempo suficiente para que o cilindro avance totalmente, é atingida o lugar "P3".

O lugar "P4" apresenta um comportamento inverso da "P2". Se a ordem de avançar do controlador aparecer quando o modelo está a recuar, este atingirá a lugar "P2". Mas, se tal não acontecer, o cilindro avança para o lugar "P1". Para a representação dos sensores são utilizadas variáveis que atualizam de acordo com os lugares do modelo do cilindro.

Existem situações em que é necessário a modelação de alguns elementos específicos constituintes no programa do PLC, para se conseguir modelar uma série mais ampla de sistemas. Por exemplo, para se programar tempo do programa de um PLC é necessário a modelação do bloco funcional TON, que foi bastante estudado por várias equipas de trabalho.

3.2 O que pode ser incrementado

As POU (*Program Organization Unit*) são as unidades básicas de um programa. Antes da IEC 61 131-3 existiam diversos blocos diferentes, o que tornava a programação pouco uniforme e eficiente, como se pode ver na Figura 12.

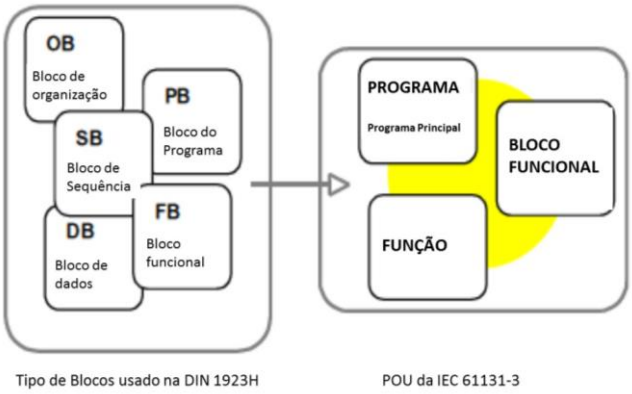

Figura 12 Tipos de blocos de programação de PLCs antes e depois da IEC 61 131-3 [adoptado de 7]

Este foi um dos grandes feitos da IEC 61 131-3 [7], pois reduziu o número de blocos dos programas. As novas POU são compostas por programas, blocos funcionais e funções.

O programa é basicamente a unidade onde o programa é escrito. Apresenta um comportamento semelhante ao dos blocos funcionais e ainda capacidade e interface para comunicar com outros programas.

Por outro lado, as funções são usadas para realizar operações matemáticas e lógicas, não apresentam memória, portanto com os mesmos *inputs* registam sempre as mesmas respostas.

Por sua vez, os blocos funcionais nas linguagens de programação de PLCs são unidades organizadas que quando executadas produzem um ou mais valores, e nem sempre com as mesmas entradas se produz o mesmo resultado. Isto porque os valores finais dependem sempre da memória que fica guardada. Por exemplo, no bloco funcional flanco ascendente, o *output* depende da entrada atual e do *input* da execução anterior (que foi guardada na memória do bloco). Só os *inputs* e *outputs* devem ser de fácil acesso fora de um bloco funcional, sendo que as suas variáveis internas devem ser escondidas do seu utilizador. No processo de modelação, esta particularidade à primeira vista pode ser considerada uma dificuldade, pois para ser realista o modelo tem de ser capaz de ser utilizado sem realizar alterações no seu código interno, mas para trabalhos posteriores, mostra-se uma vantagem pois é atingida a máxima reutilização dos modelos.

Estes elementos são compostos por 3 partes, uma estrutura onde se definem as variáveis de entrada e saída, as variáveis internas e por fim, o corpo do bloco, onde se descreve o seu comportamento, como se pode ver na Figura 13 [6,7]

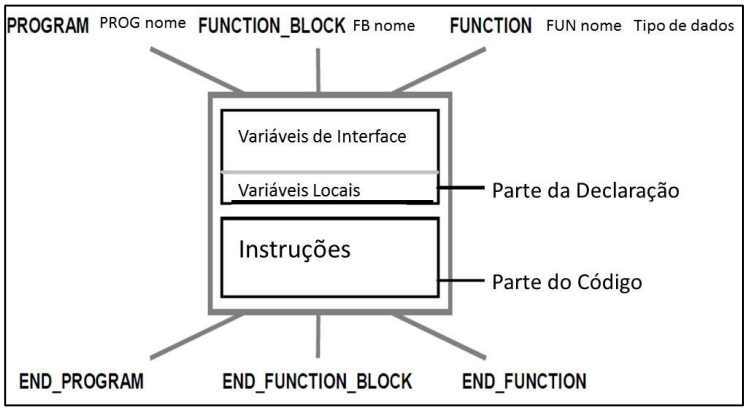

Figura 13 Estrutura detalhada das POU [adaptado de 7]

Para o processo de Simulação e Verificação Formal considerou-se relevante a criação de modelos dos blocos funcionais TON, TOF, flanco ascendente e descente e os três contadores (decremental, incremental e incremental-decremental), pois são comportamentos básicos indispensáveis para o desenvolvimento de programas de PLC de muitos sistemas de controlo aplicados na indústria.

3.2.1 TON

O bloco funcional TON (*Time ON delay*) é utilizado quando é necessário programações de tempo. Este apresenta dois *inputs*, que são o intervalo de tempo (PT) e o sinal sujeito ao atraso provocado por este bloco funcional (IN). Os *outputs* por sua vez são o sinal booleano (Q), que informa que o intervalo

"PT" foi atingido ou não. "ET" representa o tempo já contado, como se pode ver na representação gráfica da Figura 14.

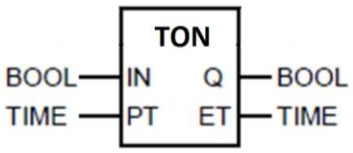

Figura 14 Representação gráfica do Time on delay [7]

Este elemento apresenta o seguinte comportamento: quando o *input* apresenta a mudança de estado do valor lógico zero para um o tempo, começa a contar o tempo "PT". Se o limite de tempo for atingido e ainda se registar o "IN" ativo, o seu output (Q) passará a ativo. Este comportamento serve, por exemplo, para se atrasar um estado do controlador. Mas, se o sinal de entrada deixar de existir sem se atingir o intervalo de tempo "PT", o sinal de saída "Q" não se altera mantendo-se desativado. E este, aliado com o comportamento anterior, é muitas vezes utilizado em sistemas de segurança, para que consiga reagir instantaneamente a alguma avaria. O comportamento descrito encontra-se representado no diagrama de tempo da Figura 15.

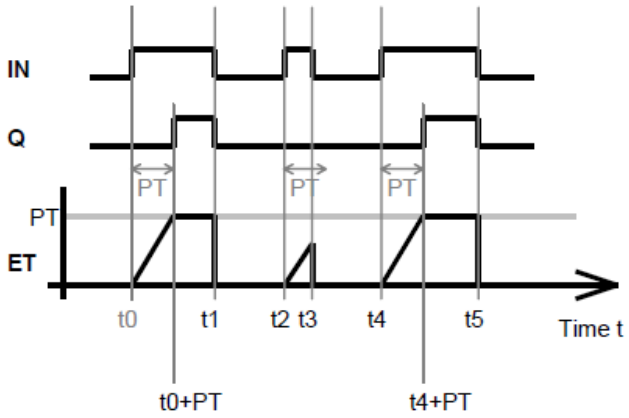

Figura 15 Diagrama com o comportamento do TON [7]´

3.2.2 TOF

Por outro lado, o bloco funcional TOF (*Time OFf delay*) apresenta as mesmas variáveis de entrada e saída que TON e o mesmo significado, como se pode ver na representação gráfica da Figura 16.

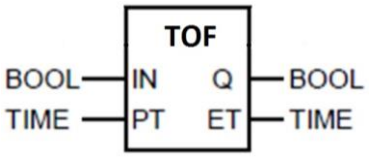

Figura 16 Representação gráfica do TOF [7]

Contudo, o comportamento deste bloco é exatamente o oposto do apresentado pelo TON, isto é, o processo de contagem só se inicia quando o booleano "IN" realizar a transição lógica de um para zero. Se o intervalo de tempo "PT" for atingido sem se voltar a registar "IN" o output booleano "Q" responderá voltando para zero. Contrariamente, se "IN" se registar novamente dentro do intervalo "PT" o output booleano "Q" não se altera, mantendo-se ativo. Este comportamento encontra-se descrito no diagrama da Figura 17.

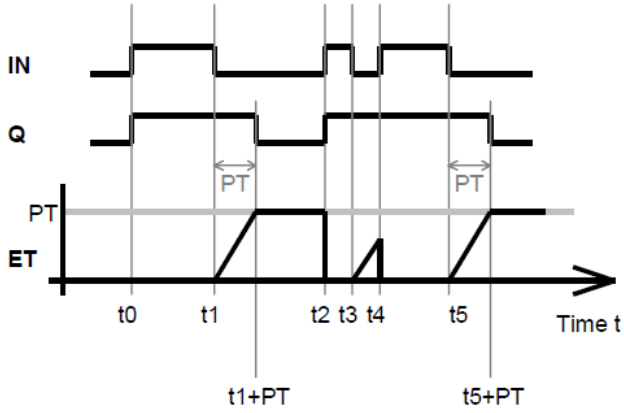

Figura 17 Diagrama do comportamento do bloco funcional TOF [7]

Este FB, tal como o anterior, é utilizado por razões de segurança.

3.2.3 Flanco ascendente

O flanco ascendente é um dos blocos funcionais com maior importância para esta dissertação, pois não existe, até a data, nenhum trabalho que se mostre preocupado com este elemento. Sem o referido FB, existe um grande número de programas que não podem ser implementados. Este é composto apenas por um *input* e um *output,* sendo o primeiro (CLK) a variável sobre a qual pretendemos estudar a mudança de estado de um para zero, que pode por exemplo ser um sensor ou uma condição de transposição. "Q" representa o sinal booleano que será acionado de acordo com variação de CLK, como se pode ver na representação gráfica da Figura 18. Este FB serve para registar a mudança do valor lógico de uma determinada variável de zero para um, sendo esta informação depois utilizada no programa do PLC.

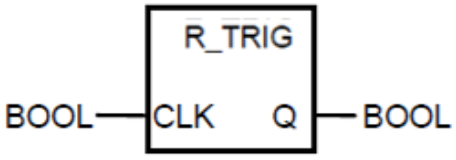

Figura 18 Representação gráfica do flanco ascendente [7]

O seu comportamento é descrito em ST, como se pode ver na Figura 19, onde se constata que a resposta do bloco não é só influenciada pela sua entrada, mas também pelos valores guardados na memória da execução anterior.

```
FUNCTION_BLOCK          R_TRIG
VAR_INPUT
CLK   :   BOOL
END_VAR
VAR_OUTPUT
Q   :   BOOL
END_VAR
VAR_RETAIN
MEM   :   BOOL := 0
END_VAR
Q   :=   CLK AND  NOT  MEM
MEM   :=   CLK
END_ FUNCTION_BLOCK
```

Figura 19 Código com o comportamento do bloco funcional flanco ascendente em ST [7]

O código demonstra que existe uma variável interna ("MEM") que guarda o valor de "CLK" em cada ciclo interno. Essa informação não é desperdiçada, pois é utilizada no cálculo de saída (Q) no próximo ciclo. Quando "Q" tem o valor lógico um, significa que o "CLK" fez a transição referida, como está assinalada na Tabela 1, onde se apresentam todos os valores lógicos das variáveis existentes no bloco funcional e como estas influenciam o *output* "Q".

Tabela 1 Valores lógicos das variáveis do flanco ascendente

Variáveis	Valores lógicos			
CLK	**1**		0	
MEM	1	**0**	1	0
Q	0	**1**	0	0

Por outro lado, é necessário também ter em atenção que o sinal "Q" apenas deve estar ativo durante um ciclo interno do PLC. Esta particularidade é de extrema importância para a estrutura de um programa e deve ser tida em consideração no processo de modelação.

3.2.4 Flanco descendente

O flanco descendente, por sua vez, serve para informar a CPU de que determinada variável realizou a transição lógica de um para zero. Este elemento apresenta uma representação gráfica semelhante à

utilizada para presentar o outro flanco, com as mesmas variáveis de entrada e saída, como se pode ver na Figura 20.

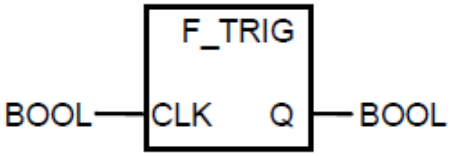

Figura 20 Representação gráfica do flanco descendente [7]

O seu comportamento é descrito em ST (Figura 21), onde se constata que o não é guardado na memória do flanco o valor da execução anterior, mas sim a negação da mesma.

```
FUNCTION_BLOCK          F_TRIG
VAR_INPUT
CLK   :   BOOL
END_VAR
VAR_OUTPUT
Q   :   BOOL
END_VAR
VAR_RETAIN
MEM   :   BOOL:=1
END_VAR
Q   :=    NOT  CLK  AND  NOT  MEM
MEM   :=    NOT  CLK
END_ FUNCTION_BLOCK
```

Figura 21 Código com o comportamento do bloco funcional flanco descendente em ST [7]

Neste bloco funcional, o raciocínio é exatamente o oposto do flanco ascendente. Neste caso, quando o valor guardado na memória (MEM) for igual ao da variável de entrada (CLK) e ao mesmo tempo os dois apresentarem valor lógico zero, é que será acionada a variável booleana "Q", como se pode ver assinalado na Tabela 2 que apresenta todos os possíveis valores de todas as variáveis do bloco funcional e como estas influenciam o seu *output*.

Tabela 2 Valores lógicos das variáveis do bloco funcional flanco descendente.

Variáveis	Valores lógicos			
CLK	1		**0**	
MEM	1	0	1	**0**
Q	0	0	0	**1**

3.2.5 Contador incremental (CTU)

Este é o bloco funcional pensado para realizar contagens através de incrementação. Apresenta os seguintes *inputs*: um booleano, que transmite ao bloco que deve incrementar (CU), uma variável

booleana (R), que dá ordem para se reiniciar a incrementação e, por fim, um inteiro (PV), que define o limite máximo que o programador predefine. Quanto ao que diz respeito às saídas, este bloco apresenta um booleano que é ativado quando o limite máximo PV for atingindo, e um inteiro (CV) que mostra o valor presente da incrementação. Este bloco funcional está representado graficamente na Figura 22.

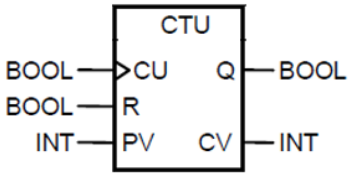

Figura 22 Representação gráfica do bloco funcional contador incremental [7]

O seu comportamento é descrito em ST, apresentado na Figura 23. Este elemento é composto por duas condições se (*if*) básicas: uma para colocar a incrementação a zero (reiniciar CV); e outra para realizar a incrementação propriamente dita, tendo em atenção que a incrementação deve parar quando o limite "PV" é atingido (Figura 23). O booleano "Q" apenas é validado quando limite é alcançado, ou seja, quando "CV" é igual a "PV".

```
FUNCTION _BLOCK CTU
VAR_INPUT
CU : BOOL R_EDGE;
END_VAR
VAR_OUTPUT
Q : BOOL;
CV : INT;
END_VAR
IF R THEN
CV := 0;
ELSIF CU AND (CV < PV) THEN
CV := CV + 1;
END_IF
Q := (CV >= PV)
END_FUNCTION_BLOCK
```

Figura 23 Código do comportamento do contador incremental em ST [7].

Há que salientar ainda, que o ato de contar pressupõe não só o uso do bloco funcional contador incremental, mas também, o uso do flanco ascendente para gerar o sinal CU, pois só se deve contar uma vez por cada evento.

3.2.6 Contador decremental (CTD)

Este bloco funcional procura realizar o processo inverso do anterior, ou seja, a contagem a partir de um limite superior decrementando progressivamente até um limite inferior. Neste bloco existem três

inputs, sendo "CD" um booleano que apenas se encontra ativo quando for para decrementar, este sinal é gerado por um flanco descente. Neste caso, reiniciar um bloco consiste em voltar a carregar o limite superior, esta informação é transmitida pelo booleano LD (*load value*) que dá a referida ordem, cujo valor é predefinido pelo inteiro PV. Por outro lado, apresenta duas saídas, um booleano (Q) que informa quando o limite inferior foi atingido e um inteiro que exibe o valor presente do contador (CV). Este bloco encontra-se representado graficamente na Figura 24.

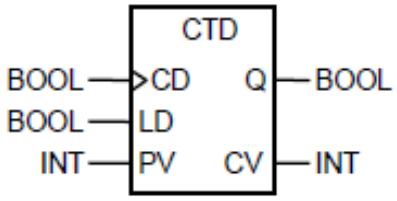

Figura 24 Representação gráfica do bloco funcional contador decremental [7].

Mais uma vez, o comportamento deste bloco é apresentado em ST apresentado na Figura 25. Este bloco foi pensado para se iniciar a decrementar a partir de um limite mínimo (PV_{min} Figura 25) que, nesta dissertação, será considerado zero.

```
FUNCTION _ BLOCK      CTD
VAR_INPUT
CD   :   BOOL   F_EDGE;
LD   :   BOOL;
PV   :   INT;
END_VAR
VAR_OUTPUT
 Q   :   BOOL;
CV :   INT;
END_VAR
IF   LD THEN
CV := PV;
ELSIF CD AND (CV > PVmin)  THEN
CV := CV − 1;
END_IF
Q := (CV <= 0)
END_FUNCTION_BLOCK
```

Figura 25 Descrição do comportamento do contador decremental [adaptado de 7].

No caso do contador decremental, é necessária a utilização de um flanco descendente, para que apenas se conte uma vez por cada evento, e o limite a partir do qual se inicia a contagem é dependente da aplicação.

3.2.7 Contador incremental e decremental (CTDU)

Este elemento é basicamente a combinação dos dois anteriores onde é possível incrementar e decrementar sobre a mesma variável. Apresenta basicamente os mesmos *inputs e outputs* do CTU e CTD combinados, como se pode ver na Figura 26. Contudo, existe uma diferenciação entre os dois sinais booleanos de saída, um para informar que o limite mínimo foi atingido (QD) e um segundo, para transmitir que o limite máximo foi alcançado (QU). Esta distinção não era necessária nos elementos anteriores, pois apenas existia um sinal booleano de saída.

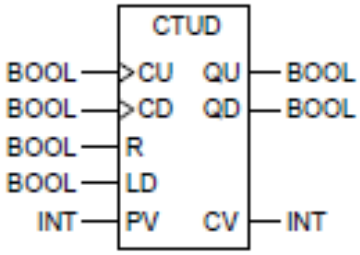

Figura 26 Representação gráfica do contador decremental e incremental [7].

Mais uma vez, o código encontra-se descrito em ST e é apresentado na Figura 27, como se pode ver é a combinação dos dois blocos anteriores.

```
FUNCTION _ BLOCK      CTDU
  VAR_INPUT
    CD   :   BOOL    R_EDGE;
    CU   :   BOOL    F_EDGE;
    LD   :   BOOL;
    R    :   BOOL;
    PV   :   INT;
  END_VAR
  VAR_OUTPUT
    QD   :   BOOL;
    QU   :   BOOL;
    CV   :   INT;
  END_VAR
  IF  R  THEN
      CV := 0;
  ELSIF LD  THEN
      CV := PV;
  ELSIF  CD AND (CV > PVmin)  THEN
      CV := CV – 1;
  ELSIF  CU AND (CV < PV)  THEN
      CV := CV + 1;
  END_IF
  QU := (CV >= PV)
  QD := (CV < 0)
END_FUNCTION_BLOCK
```

Figura 27 Descrição do comportamento do bloco em ST [adaptado de 7]

Este bloco combina a capacidade de incrementar e decrementar em um mesmo elemento, sendo necessário ter em atenção todas a particularidades dos dois anteriores. E ainda, como se está a incrementar e decrementar sobre a mesma variável, a atualização dos dois sinais booleanos de saída deve ser feita sempre que se conta. Este bloco é ideal, por exemplo, para aplicações de gestão de *stocks*.

3.3 Autómatos finitos temporizados, TCTL e UPPAAL

3.3.1 Autómatos finitos temporizados: descrição formal

Tal como já foi referido, os autómatos finitos temporizados é um formalismo ideal para expressar sistemas de tempo real e foi proposto na sua versão final em 1994 [72], em que os autores defendem ser simples e, ao mesmo tempo, uma poderosa maneira de modelar graficamente transições de estado com constrangimentos de tempo constantes, e cujo seu valor é real. De acordo com [73,74], um sistema é modelado através de uma linguagem gráfica de estados-transições, em que as transições são etiquetadas com eventos.

Podemos definir um autómato (A), como sendo uma função que depende do conjunto de estados (Q), das etiquetas associadas a cada transição (E), do conjunto de transições (T= $Q^{n-1} \times E^{n-1} \times Q^{n}$) e do estado inicial ($Q_0$), portanto virá na Equação 1:

$$A = (Q, E, T, Q_0) \tag{1}$$

Por outro lado, uma transição (q,a,r) pertencente a "T", sendo "a" um evento e "q" um estado e "r" o estado seguinte, isto significa, se o autómato estiver no estado "q" e o evento "a" ocorrer, o estado "r" é atingido.

Um sistema complexo, por ser expresso através da interação entre autómatos, considerando $A_1=(Q_1,E_1,T_1,Q_1^0)$ e $A_2=(Q_2,E_2,T_2,Q_2^0)$ dois autómatos consequentemente, o estado do sistema é o produto entre A_1 e A_2 e será o par (q_1,q_2) em que q_1 pertecente a Q_1 e q_2 pertecente a Q_2. As transições são, então, etiquetadas respetivamente por E_1 e E_2 e considerado novamente o evento "a", existem portanto três situações possíveis (ver Equação 2):

$$\text{(I)} \quad a \in E_1 \cap E_2 \Rightarrow q_1 \xrightarrow{a} q_1' \; e \; q_2 \xrightarrow{a} q_2' \tag{2}$$

$$\text{(II)} \quad a \in E_1 \Rightarrow q_1 \xrightarrow{a} q_1' \; e \; q_2 = q_2'$$

(III) $a \in E_2 \Rightarrow \mathrm{q}_1 = \mathrm{q}_1' \; e \; \mathrm{q}_2 \xrightarrow{\mathrm{a}} \mathrm{q}_2'$

Em (I) está-se perante o caso em que "a" pertence aos dois autómatos. Por outro lado, os casos (II) e (III) são as situações em que "a" apenas faz parte de um deles separadamente. Quando "a" está presente nos dois autómatos pode ser usado como elemento sincronizador.

Até aqui expressou-se como desenvolver modelos usado autómatos, mas sem existirem considerações de tempo. Antes disso, é necessário definir como o tempo é especificado, qualquer valor que tenha uma relação lógica não negativa possível de ser usada como constrangimento de tempo. Considerando um conjunto de relógios "X" e um constrangimento constante "C^+", e sendo "x" pertencente "X" e "c" pertencente "C^+", as relações possíveis serão $x \geq c$, $\mathrm{c} \geq \mathrm{x}$, $\mathrm{x} < \mathrm{c}$ e $\mathrm{c} < \mathrm{x}$. É importante ressalvar, também, que estes constrangimentos podem ser combinados com outras tarefas ou eventos.

Devido à capacidade de poderem ser reiniciados, os relógios não medem tempo, mas sim prazos, isto é possível devido à existência de um relógio global, que sincroniza todos os outros. Pode-se então definir um autómato temporizado de acordo com Equação 3:

$$A = (\mathrm{L}, \mathrm{L}_0, \mathrm{E}, \mathrm{X}, \mathrm{I}, \mathrm{T}.) \tag{3}$$

Sendo "L" o conjunto de lugares (*locations*), "L_0" ⊆ "L" o conjunto de lugares iniciais, "X" o conjunto finito de relógios (*clocks*), "I" o mapeamento que coloca etiquetas em cada lugar (L) com constrangimentos de tempo e T ⊆ L x E x 2^X I x L o conjunto de transições.

Uma transição (s, a, p, λ, s_2) onde "s" é o estado de partida e "s_2" o estado de chegada, "a" o evento impulsionador, "p" o constrangimento de tempo sobre "X", o λ ⊆ X dá a ordem para os relógios serem reiniciados. A sinemática utilizada pode ser apresentada por um autómato "A" através de um sistema de transições "S_A". Um estado de "$S_{A,}$" é o par (s, v) tal que "s" é um lugar de "A" e "v" um relógio interpretador de "X", tal que "v" satisfaz a condição I (s). O conjunto de estados de "A" é chamado de Q_A. O par (s, v) é estado inicial se "s" é um lugar inicial de "A", e se v(x)=0 para todos o relógios de "X". Existem dois tipos de transições possíveis:

- O tempo já passou: para um estado (s, v) e um valor real de incrementação $\sigma \geq 0$ tal que $(\mathrm{s}, \mathrm{v}) \xrightarrow{\sigma} (s, v + \sigma)$ para todos $0 \leq \sigma` \leq \sigma$ , e v + $\sigma`$ satisfaz a invariante I (s).
- O estado pode mudar devido à interação transição-lugar: para um estado (s, v) e uma transição (s, a, p, λ, s_2), a transição acontece quando "v" satisfaz a condição "p".

3.3.2 TCTL (*Timed Computation Tree Logic*)

TCTL (*Timed Computation Tree Logic*)[75] é a lógica temporal utilizada neste método, para especificar as propriedades, como já foi referido, esta é uma extensão CTL (*Computation Tree Logic*) proposta por Emerson e Clarke em 1982 [76], e que por sua vez é um subconjunto da CTL* (*Full Computation Tree Logic*) [77]. Esta lógica não considera o tempo de forma específica, mas sim tendo em atenção sequências de ações, ou seja, estados. Portanto, as propriedades em que o tempo é absoluto não são consideradas neste tipo de lógica. Na prática, existem formas de contornar este facto, utilizando por exemplo, as variáveis dos relógios.

Esta linguagem admite uma série de quantificadores atómicos (propriedades), os operadores booleanos , $\wedge$, $\vee$, $\Rightarrow$ e os operadores temporais.

Uma propriedade regra geral é composta por um quantificador de caminho, um quantificador de estado e uma característica que se quer verificar. O quantificador de estado especifica os estados. Por outro lado, um quantificador de caminho específica se é em todo o modelo ou partes deste.

3.3.3 UPPAAL: capacidades e descrição formal

O UPPAAL surgiu em 1995 a partir de uma parceria entre as universidades de Uppsala e Aalborg. É uma ferramenta que permite testar, a partir de Simulação gráfica ou através Verificação Formal, sistemas cujo tempo tem um papel crucial, e que o seu comportamento possa ser modelado por uma coleção de processos não determinados com uma estrutura de controlo finita e com comunicação, através de mensagens e variáveis partilhadas [53]. Esta ferramenta está em constante desenvolvimento, apresentando melhores resultados a cada ano, em termos de performance, estruturas de dados, capacidades de modelação e algoritmos implementados [78].

Em 2006, a versão base do UPPAAL atual, que aumentou a aplicabilidade e a sua performance, apresenta algumas características novas [70]. Esta versão permite, desde logo a criação de funções em linguagens de programação convencionais, nomeadamente C/C++ e *Java*, e as estruturas de controlo de fluxo da linguagem C são quase todas suportadas, isto permite reduzir estados intermédios nos modelos, pois assim, a sua evolução é conseguida diretamente. Existem apenas duas limitações, sendo uma delas o facto de as funções não se poderem invocar a si mesmas e tem de existir sempre retorno nas funções criadas. Esta última não é forçada pelo UPPAAL, mas se a função não apresentar retorno, consequentemente o modelo apresentará um ciclo infinito e a Verificação Formal não terminará. É ainda

permitido definir prioridades nos eventos a cumprir, estas são traduzidas pelas transições internas e pelos canais de sincronização [70]. Usa-se nas mais diversas áreas, desde sistemas de controlo em tempo real, protocolos de comunicação e outros sistemas, onde o tempo é crucial.

Um autómato na linguagem utilizada pelo UPPAAL não só suporta o formalismo TA, mas também algumas extensões ao mesmo, portanto, um autómato é composto um conjunto dos vários lugares dos diferentes *templates* que cooperam em simultâneo. Um lugar pode ser de diversos tipos [79,80]:

- Urgente (*urgent*): onde o tempo não deve passar quando o sistema está num destes lugares, conseguindo-se assim uma redução do número de relógios (*clocks)*, simplificando-se o modelo;
- *Committed:* onde o tempo também não e permitido que passe, contudo, estas têm a particularidade que se estiverem ativas, apenas as transições a partir destes estados podem acorrer. Caso exista mais que um lugar *committed* ativa ao mesmo tempo, será escolhida de forma aleatória uma das transições para ser transposta.

É necessário ainda definir o conceito de invariantes (*invariants),* que são condições que devem ser cumpridas quando o autómato está num determinado lugar. Estas podem ser relacionadas com variáveis ou com constrangimentos de tempo. Apenas os lugares iniciais e normais podem conter invariantes. O autómato manterá esse estado enquanto essa tarefa não for cumprida [79] e avançará para a transição seguinte quando esta for cumprida.

As transições, por sua vez, podem conter constrangimentos de tempo, reiniciar relógios e apresentar guardas (*guards*), este é um dos conceito mais importantes do UPPAAL, pois permite criar condições, para selecionar as transições pretendidas em determinado instante. Portanto, uma transição apenas ocorrerá quando os requisitos da sua guarda se verificarem. As transições também podem conter variáveis que apenas são validas naquela transição, e que podem ser usadas para influenciar outros parâmetros dentro dessa, sendo estas introduzidas no parâmetro *select*. Outra noção importante é a capacidade de se sincronizar *templates* a partir de mensagens, utilizando as sincronizações. Com este parâmetro é possível invocar ou ativar uma ou mais transições, utilizando um canal de sincronização previamente definido. A utilização da etiqueta "!" pode ser vista como um envio e a etiqueta "?" como uma receção. Existem três tipos de sincronização [79, 80]:

- Canal regular: quando um processo está num estado a partir do qual existe uma transição com a sincronização "c!", a única forma dessa transição ser ativada é se existir outro

processo noutra transição marcada com "c?", ou vice-versa. Se existirem vários pares "c!" e "c?", um deles é escolhido de uma forma não determinada;

- Canal Urgente: este tem um funcionamento semelhante ao anterior, mas não são permitidos atrasos, ou seja, não se podem utilizar guardas com relógios. Estas transições apenas podem interagir com outras que não exijam que passe tempo;
- Canal *Broadcast:* é um estado que tem uma transição marcada com "c!" e existem múltiplos processos com uma sincronização com "c?". Todas essas transições são executadas. No entanto, se não há processos com uma transição marcada com "c?", a transição marcada com "c!", realizar-se-á de qualquer maneira.

É ainda importante referir, que caso um lugar tenha mais que uma transição transponível ao mesmo tempo, apenas um deles será ativada, sendo a sua escolha feita de forma aleatória.

Existe ainda a capacidade de atualizar variáveis booleanas e inteiros, e até mesmo reais, mas estes últimos devem ser evitados, por sobrecarregarem os modelos. É possível, também, atualizar funções devolvidas em C++, que se revela bastante vantajoso para evitar estados desnecessários. Para tal, utiliza-se o parâmetro "*update*" das transições.

Tal como já foi referido, o UPPAAL permite realizar Verificação Formal por *Model Checking* dos modelos utilizando a linguagem TCTL para especificar as propriedades, pelo que, os operadores temporais têm um correspondente em UPPAAL, que se apresenta na Tabela 3.

Tabela 3 Quantificadores de TCTL e correspondência em UPPAAL.

TCTL	UPPAAL	Significado
A	A	Para todos os caminhos
E	E	Existe um caminho
G	[]	Todos os estados
F	<>	Algum estado
∧ , ∨	&&, ||	E, OU
¬, ⇒	not/ !, →	Negação/ Implica

Os tipos de propriedades que podem ser testados diretamente no UPPAAL, usando os seus editores, são bastante simples. A verificação de propriedades mais complexas pode necessitar ferramentas diferentes e mesmo, a adição de modelos de teste personalizados. Todas as propriedades são compostas

por um quantificador de caminho e um de estado, ou seja, primeiro define-se se é referente a um ou a todos os caminhos e, depois, se é aplicado a um estado ou a todos. Estas propriedades podem ser expressas através da linguagem do UPPAAL e podem ser classificadas por propriedades de acessibilidade, de segurança (*safety),* evolução *(liveness)* e *deadlock.*

As propriedades de acessibilidade são aquelas que permitem especificar se uma situação pode ser atingida. Considerando uma propriedade "p", estas são sempre expressas da forma "E<> p", ou seja, existe um caminho eventual, pelo qual o estado "p" acontecerá como esquematizado na Figura 28.

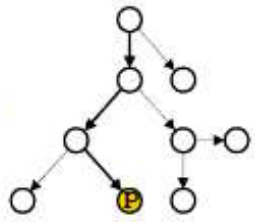

Figura 28 Demonstração da propriedade acessibilidade.

Por outro lado, as propriedades de segurança (*safety*) especificam que determinada condição acontece em todos os estados do caminho executado. Existem duas possibilidades, sendo a primeiro opção "E[] p", significando que existe um caminho global "p", isto é, existe um caminho executável, no qual "p" acontece em todos os estados, como demostrado na Figura 29.

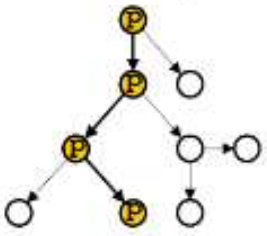

Figura 29 Demonstração da primeira propriedade de segurança.

A segunda opção é a basicamente a generalização da anterior, vindo "A[] p", significando todos os caminhos globais "p", ou seja, para todos os caminhos executáveis, "p" acontece em todos os estados, como está demonstrada na Figura 30.

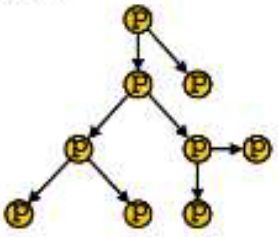

Figura 30 Demonstração da segunda propriedade de segurança.

Já as propriedades de evolução (*liveness*) são condições que garantem a ocorrência do acontecimento, num determinado momento. Existem novamente duas possibilidades, sendo a primeira "A<> p", significando que "p" acontece eventualmente para todos os caminhos, ou seja "p" acontece, pelo menos uma vez, em todos os caminhos, como demonstrado na Figura 31.

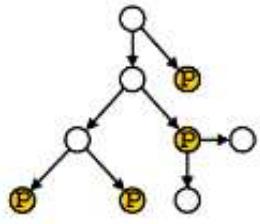

Figura 31 Demonstração da primeira propriedade de evolução.

Por outro lado, a segunda evolução funciona num princípio de causa efeito, ou seja "q –> p" (Em TCTL corresponderá AG (q ⇒ EF p)), significando que "q" leva sempre a "p". Qualquer caminho que começa com "q" eventualmente atingirá mais tarde um estado "p", como demonstrada na Figura 32.

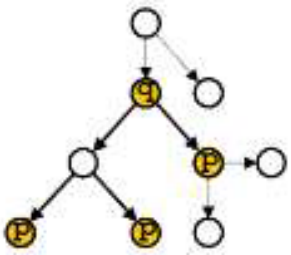

Figura 32 Demonstração da segunda propriedade de evolução.

E por fim, as condições de *deadlock*, ou seja, a não existência de *deadlock,* implica que não exista nenhum estado no sistema em que o autómato fica preso. Pode ser expressa de diversas formas, mas as duas mais comuns são:

- E <> *deadlock*, existe pelo menos um estado em que o sistema fica preso;
- A [] *not deadlock*, não existe *deadlock*, ou seja, não existe nenhum estado em que o sistema fica preso.

3.4 Técnicas de Simulação e Verificação Formal

Para simular um sistema mecatrónico existem quatro formas destintas, como esquematizado na Figura 33, sendo estas designadas de SiL (*Software-in-the-loop)*, HiL (*Hardware-in-the-loop), LT (Laboratory-testing*) e MiL (*Model-in-the-loop*).

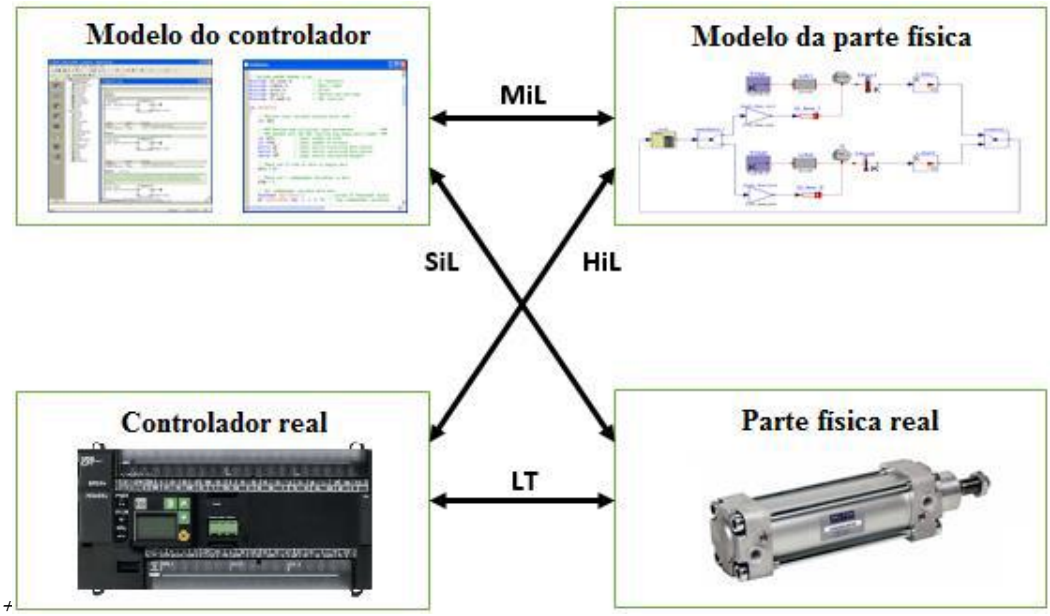

Figura 33 Diferentes formas de Simulação [81]

SiL carateriza-se por o programa do controlador ser simulado através de modelos, que depois são ligados à parte física real, sendo o comportamento modelado em malha fechada. Esta técnica combina a flexibilidade e baixo custo da Simulação, através de modelos com a maleabilidade de um emulador da parte física [82]. Vários equipamentos podem ser ligados aos modelos, mesmo que estes não estejam no mesmo laboratório em que se realiza a Simulação, permitindo uma gama alargada de testes. Admite desde logo simulações em ambientes controlados com grande flexibilidade, repetibilidade e elevada velocidade de processamento, conseguindo-se resultados num curto período de tempo. É possível, ainda, comparar os resultados entre diferentes estratégias, escolhendo-se posteriormente a melhor solução para se resolver o problema [83]. Esta técnica apresenta como principal desvantagem, em alguns simuladores, a representação do tempo não ser realista, podendo levar a discrepâncias nos resultados.

Por outro lado, a HiL [84] consiste em simular a parte física do sistema a partir de modelos e utilizar um controlador real. Este método é usado para analisar sistemas embebidos. Os sistemas HiL permitem a Simulação de várias situações e ambientes que seriam difíceis e/ou de elevados custos financeiros, utilizando protótipos reais para fazer estudos sobre o controlador. Esta metodologia deve incluir emulação eletrónica de sensores e atuadores. Esta emulação age como uma interface entre a Simulação da parte física do sistema e o controlador sobre teste. Os valores de cada um dos sensores são controlados pela Simulação da parte física do sistema, posteriormente, lida pelo controlador em teste, por sua vez, o controlador envia os sinais de *feedback*, ou seja, os valores para os atuadores. Consequentemente, a mudança nos sinais do controlador implica variações nas variáveis da Simulação [85, 86]. A metodologia

HiL permite, portanto, estudar as respostas dos controladores, tendo em conta as suas especificações e necessidades, aumentando a segurança sem grandes gastos.

LT (*Laboratory-testing*), por outro lado, é uma técnica que utiliza controladores e equipamentos reais em laboratório, conseguindo-se assim, uma análise do equipamento. Contudo, nem sempre esta é possível e/ou viável, devido à complexidade e aos gastos associados. Habitualmente, são realizados testes de comportamento específico em situações de normal funcionamento e testes de segurança. Desta forma, são obtidas informações muito próximas sobre o comportamento dos sistemas a operar em condições reais, portanto, é bastante útil nas fases finais do projeto. De acordo com os resultados, são realizados melhoramentos e corrigidos os problemas encontrados no protótipo desenvolvido.

Além do perigo para o operador e da possibilidade de destruição de equipamentos, podem dar-se situações de inexistência de equipamento para esta análise e o desenvolvimento de um protótipo pode, simplesmente, tornar-se demasiado dispendioso.

Todas estas técnicas de Simulação têm a particularidade de não testarem todo o espaço de comportamentos do controlador, tornando inviável a afirmação da sua eficácia a cem porcento.

Nesta dissertação, considera-se uma técnica de Simulação utilizando modelos (MiL), quer para o programa, quer para a parte física do sistema mecatrónico em malha fechada. Esta técnica é geralmente utilizada nas fases iniciais do processo de desenvolvimento de novos equipamentos. Não necessita de equipamentos específicos, logo são mais baratas que as anteriores. Quando desenvolvidas em ambiente adequado, além ser possível simular, também é viável a verificação por *Model Checking,* que permite garantir a análise de todo o espaço de comportamentos do controlador.

Os modelos procuram interagir entre si da mesma forma que os sistemas mecatrónicos interagem na realidade. Para que se consiga este efeito, é necessário um modelo responsável para gerir a ordem com que estes são executados e a forma como interagem entre si. Desde logo, existem dois grandes grupos de modelos, os que representam o comportamento do PLC e os que reagem, tal como acontece no processo, como se encontra esquematizado na Figura 34.

Figura 34 Ciclo de monotorização de um processo

A interação entre as duas partes do modelo é feita através de variáveis. As variáveis do processo a cada ciclo do PLC são atribuídas às variáveis internas do mesmo, e com estes dados correrá o seu código interno, que calculará as saídas. Esta informação é mais uma vez transmitida para o processo, através da atribuição dos valores das variáveis do controlador às suas correspondentes no processo.

Para realizar a modelação do comportamento do PLC é necessário utilizar diversos modelos para os diferentes comportamentos havendo consequentemente a necessidade de os interligar e fazer evoluir de forma correta, esta relação encontra-se descrita na Figura 35 [87]. Para fazer a referida evolução, utiliza-se sincronização por mensagem, existindo canais de sincronização desenvolvidos para o efeito. Como ilustrado na Figura 35, existe uma relação entre o modelo global e os outros modelos do controlador, nesta dissertação apenas considerou-se um PLC, mas qualquer outro tipo de controlador industrial estaria apto a ser considerado.

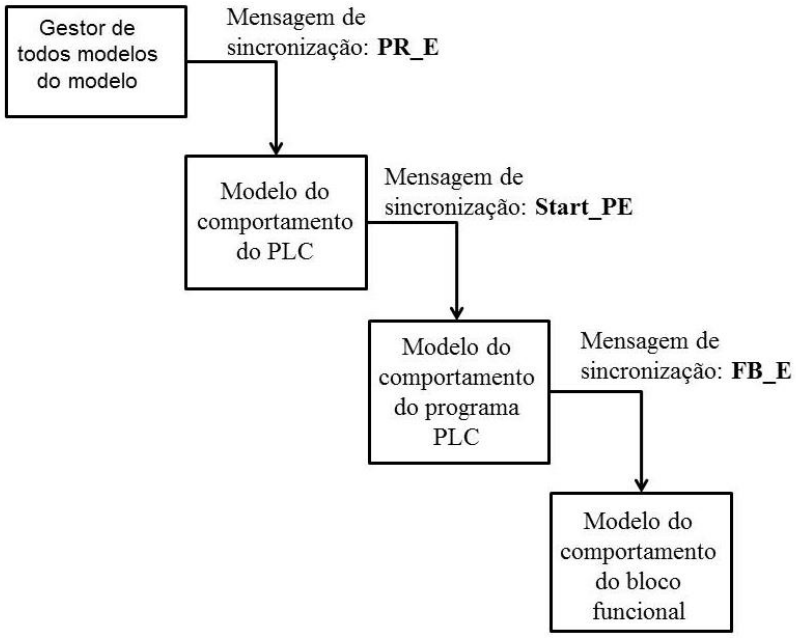

Figura 35 Representação esquemática da interação entre os modelos globais em TA usados nesta metodologia [adaptado de 87].

De facto, esta relação entre os módulos, torna possível que os valores das variáveis sejam obtidos nos momentos equivalentes aos correspondentes à dinâmica da execução do programa num PLC [87].

A Figura 36 ilustra a forma como este comportamento foi desenvolvido com as diferentes partes de cada módulo, consideradas na Figura 35.

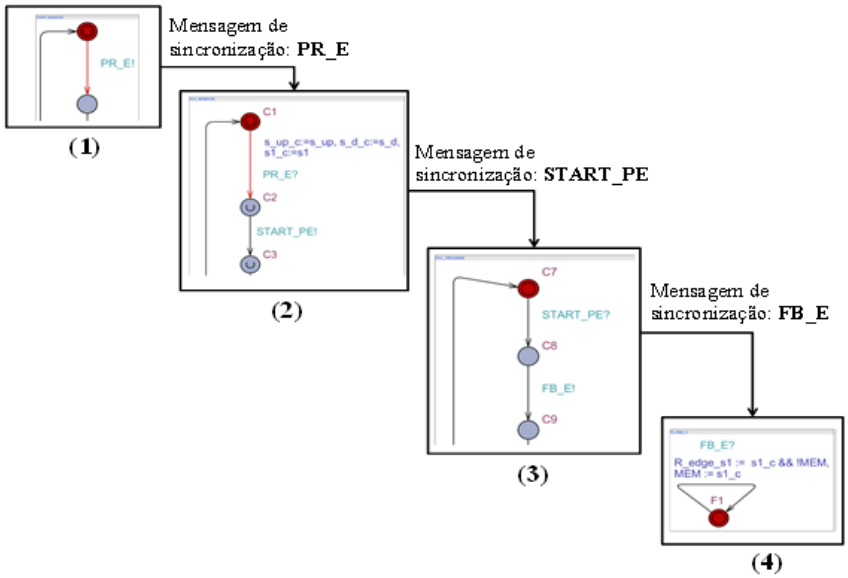

Figura 36 Ilustração da sincronização entre os modelos globais em TA, utilizados no UPPAAL [adaptado de 87].

No início, quando o modelo começa a sua evolução, o lugar inicial do módulo 1 (gestor de todos os módulos, Figura 36) começa a sua evolução e envia uma mensagem de sincronização para o módulo 2.

No módulo 2, os modelos do comportamento do controlador (Figura 36) e a mensagem recebida do módulo do gestor de eventos, permite a sua respetiva evolução. Este comportamento modelado encontra-se descrito em detalhe na referência [5].

O início da evolução do módulo correspondente ao programa do PLC, que apresenta vários passos, mas primeiro, são atualizados os blocos funcionais (módulo 4, Figura 36). Esta evolução irá ocorrer nesse preciso momento, e nunca mais durante a evolução do modelo, a menos que um novo ciclo do PLC volte a acontecer. Dentro do programa, os diferentes blocos funcionais têm de ser evoluídos em situações separadas, pois muitas das vezes estes têm implicação uns nos outros.

A RETER DESTE CAPÍTULO

Neste capítulo, inicialmente, são abordadas as técnicas de Simulação existentes para analisar controladores industriais, concluindo-se que MiL é vantajosa devido aos reduzidos custos e permitir, sobre o mesmo modelo, realizar Simulação e Verificação Formal por Model Checking. Continuamente, é apresentada a organização dos modelos do controlador e da parte física, e o que é necessário acrescentar para uma análise mias poderosa, nomeadamente os blocos funcionais da norma IEC 61 131-3. Por fim, são apresentados os formalismos utilizados para desenvolver os modelos (TA) e propriedades (TCTL) e a sua correspondência relativamente à ferramenta utilizada (UPPAAL).

Capítulo 4
CASO DE ESTUDO

Este capítulo tem como finalidade apresentar o caso de estudo utilizado para validar os modelos desenvolvidos. Numa primeira parte, é descrito o sistema e os seus componentes e apresentado a especificação de comando a implementar no controlador. Seguidamente, são expostos os modelos desenvolvidos para o controlador e para o processo. E por fim, serão apresentados os modelos dos blocos funcionais desenvolvidos que foram aplicados.

4. CASO DE ESTUDO

4.1 Descrição do sistema

O caso de estudo desta dissertação é uma barreira de um parque de estacionamento que apresenta um motor com dois sentidos de rotação: movimento de subida (M_UP) e um movimento de descida (M_D). Além disso, existe um conjunto de sensores: um para detetar que a barreira está em repouso (s_down); um para detetar que a barreira está subida (s_up) e dois sensores para detetar a presença de carro, um da parte de dentro (si) e outro para a parte de fora (so) da barreira. Esta encontra-se esquematizada na Figura 37, denota-se que o sensor "si" não se encontra representado.

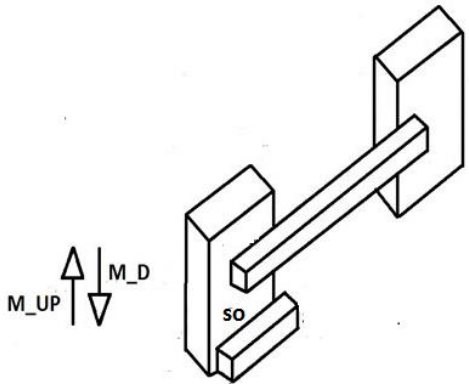

Figura 37 Esquematização da barreira aplicada ao caso de estudo.

O funcionamento que é necessário implementar no controlador é bastante simples. Consiste em controlar o levantamento da barreira de acordo com a presença de carros dentro ou fora do parque de estacionamento. Quando uma viatura é detetada a barreira deve abrir, e esta deve permanecer aberta até que o veículo transponha a barreira em segurança. Após isto, a barreira deve descer mas, se durante a descida for detetada a presença de outro carro, esta deve automaticamente iniciar o processo de subida, para reduzir o tempo de espera. O controlador, não deve levantar a barreira na presença de carros do lado de fora, se o parque se encontrar cheio, e deve nestas situações apenas abrir na presença de veículos no interior. Por questões de segurança, deve-se esperar sempre 30 segundos até dar a ordem para descer a barreira. Este comportamento deu origem a um código em SFC da Figura 38.

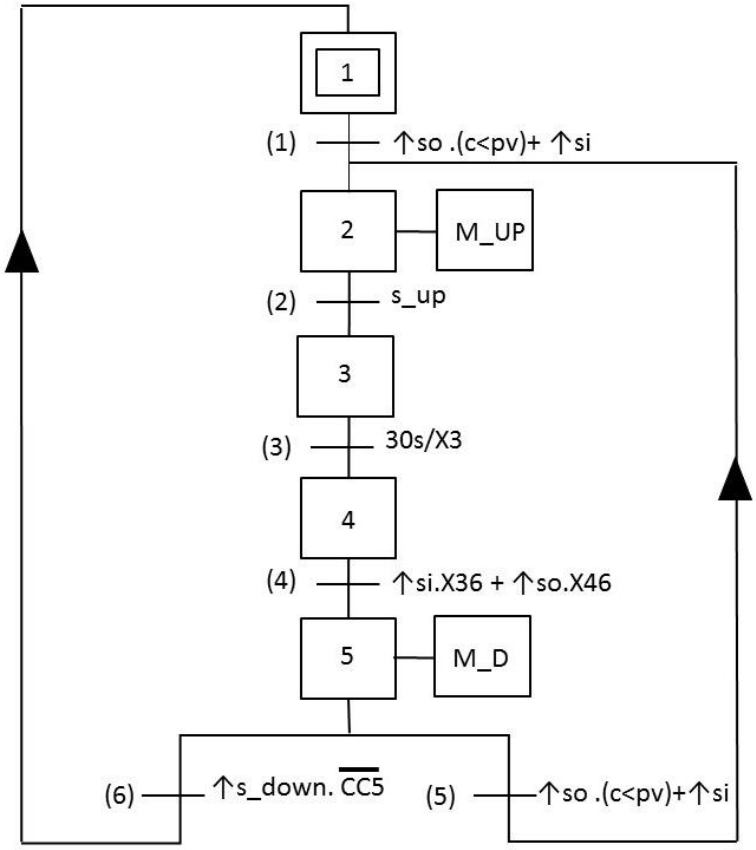

Figura 38 Especificação de Comando principal em SFC

Uma das características da linguagem SFC é a capacidade de permitir tarefas em paralelo, fazendo proveito desta aptidão, desenvolveu-se a especificação de comando principal e duas paralelas. Devido à existência de dois sensores, que detetam a presença de carros, desenvolveu-se um código para monitorizar a posição dos mesmos (Figura 39), ou seja, se a viatura vem de dentro para fora ou de fora para dentro do parque de estacionamento. E de acordo com estes dados somar ou subtrair ao número de carros "c" existentes no parque.

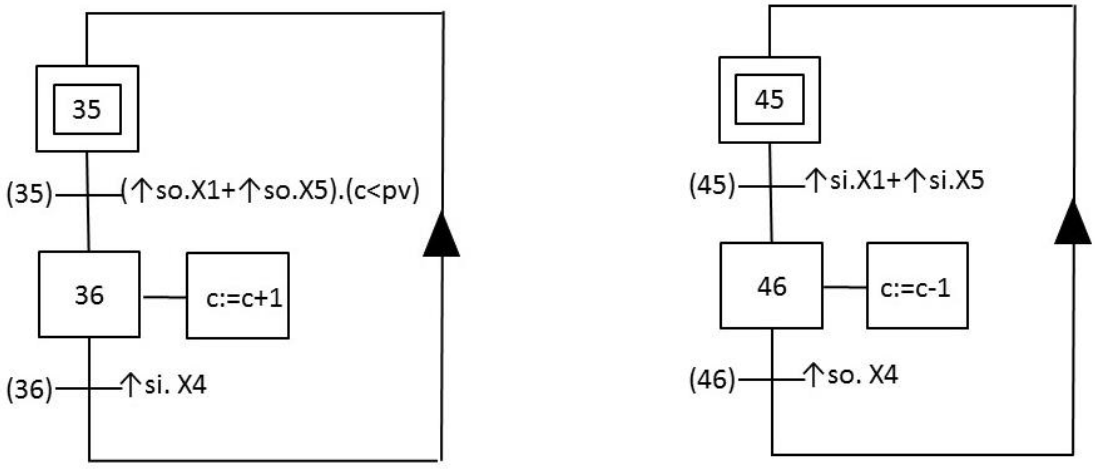

Figura 39 Especificação dos SFCs de observação

Estes SFCs paralelos foram desenvolvidos porque as instruções são sempre iguais, mas o que as ativa nem sempre é. Este método salvaguarda, também, a situação de estarem a entrar e a sair dois carros simultaneamente, sem existir qualquer conflito.

4.2 Modelação do controlador

O código desenvolvido irá corresponder às equações da Tabela 4, para que o UPPAAL consiga interpretá-las foram desenvolvidas algumas convenções. O flanco ascendente da variável "x" será representado por "R_x" e o flanco descendente por "F_x". A saída booleano dos blocos funcionais TON e TOF deve ser identificada com "T_x" e os contadores por "QU_x", caso seja, do contador de incrementação e "QD_x", caso seja, do contador decremental, para, assim, ser possível uma interpretação imediata do código implementado.

Tabela 4. Condições de transposição e correspondência em UPPAAL

Condições de transposição	Correspondência em UPPAAL
CC1= (↑ so (c < pv) + ↑ si) X1	CC1:=X1 && ((R_so && !QU_X36) || R_si),
CC2= s_up	CC2:=X2 && s_up_c,
CC3=30s/X3	CC3:=X3 && !T_X3 ,
CC4=(↑ si X36 + ↑ so X46) X4	CC4:=X4 &&((X36 && R_si) || (X46 && R_so)),
CC5= ((↑ so (c<pv)) + ↑ si) X5	CC5:=X5 &&((R_so && !QU_X36) || R_si),
CC6= ↑ s_down X5 $\overline{CC5}$	CC6:=X5 && R_s_down && !CC5,
CC35= (↑ so X1+↑ so X5) X35	CC35:=X35 &&((R_so && X1) ||(R_so && X5)),
CC36= ↑ si X4 X36	CC36:=X36 &&R_si && X4,
CC45= (↑ si X1+↑ si X4) X45	CC45:=X45 &&((R_si && X1) || (R_si && X5)),
CC46= ↑ so X4 X46	CC46:=X46 && R_so && X4

É importante salientar que o comportamento de contagem é modelado por um bloco funcional, tal como acontece nos programas de hoje em dia. Este bloco funcional tem a capacidade de contar e ativar um sinal booleano quando o limite for atingido. O mesmo raciocínio é aplicado nos *timers*, mas neste caso trata-se de um TOF, a transição só é validada quando deixar de existir o sinal deste FB. Este elemento, tal como o nome indica, atrasa o estado *off* de determinado elemento.

A modelação do controlador é composta pelo modelo do PLC de acordo com a referência [5] e os blocos funcionais, unidades fulcrais neste programa, pois sem elas seria impossível a implementação deste código (Figura 40).

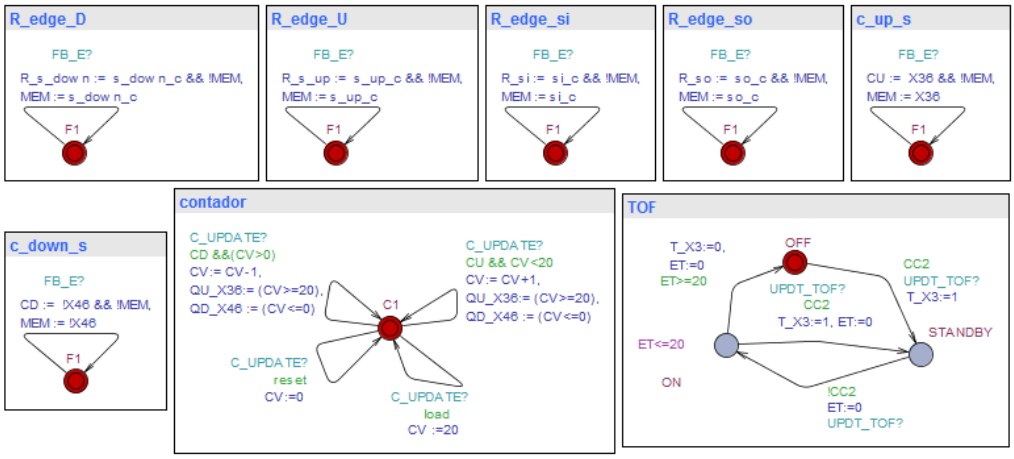

Figura 40 Blocos funcionais aplicados neste caso de estudo

A parte do controlador é composta por submodelos dos diferentes comportamentos que interagem entre si de acordo com o apresentado na Figura 36.

Existem 4 blocos funcionais responsáveis por gerar o sinal de flanco ascendente dos sensores si, so, s_up, s_down e dois que geram o mesmo sinal para os estados X36 e X46 (CU e CD), para que o modelo do contador apenas incremente ou decremente uma vez. Isto é necessário, senão o contador iria incrementar ou decrementar enquanto X36 e X46 estiverem ativos em cada ciclo do PLC. Poderiam ser optadas outras formas de sincronização dos modelos, mas neste caso utiliza-se mensagens, de acordo com uma ordem específica na Figura 35.

A atualização dos modelos dos blocos funcionais é feita através de canais de sincronização por mensagem do tipo *broadcast*, que permitem, tal como já referido, sincronizar vários modelos ao mesmo tempo. Contudo, este tipo de canais não pode apresentar guardas com constrangimento de tempo entre as transições pois, caso existam, o UPPAAL não permite testar propriedades de *deadlock.* Este facto exige atenção especial no desenvolvimento dos *timers.*

4.3 Modelação do processo

Por outro lado, os modelos da parte física (Figura 41) são compostos pelos modelos dos sensores que detetam a presença de carros dentro e fora do parque de estacionamento, por um modelo de uma

barreira e dois modelos para modelar o comportamento do motor, que por sua vez controla o movimento da barreira, um para o movimento de subida e outro para o movimento de descida.

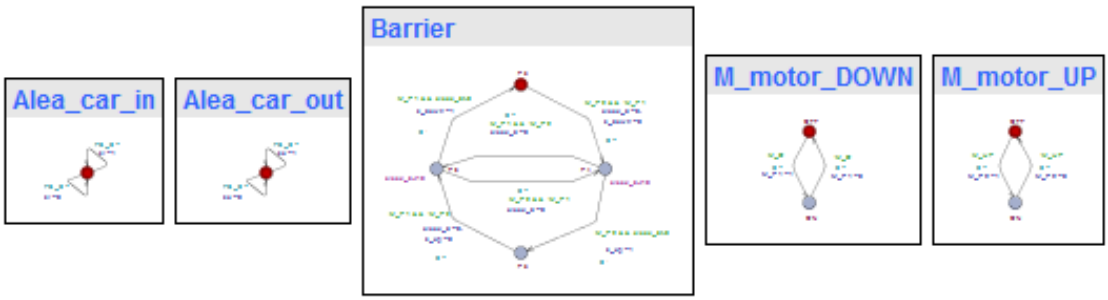

Figura 41 Modelos da parte física

Os modelos dos sensores de presença do carro são basicamente variáveis aleatórias, que trocam o seu valor a cada ciclo do PLC, ao mesmo tempo que são atualizados os flancos. Tal como acontece na realidade, o condutor pode decidir sobre a barreira do parque de estacionamento, após ter acionado o sensor, não entrar, entrar antes de a barreira estar completamente aberta, entre outras situações. Sendo, portanto, correto considerar completamente aleatório o seu comportamento, pois se controlador conseguir responder corretamente a este tipo de situações, também responderá afirmativamente numa situação real. O modelo da barreira apresenta um comportamento bastante simples que, tal como qualquer cilindro, apresenta 4 estados, correspondentes a barreira recuada, subida, a subir e a descer, de acordo a Figura 11. Este modelo permite simular a situação de a barreira estar a recuar e ter de subir, porque apareceu um carro. Esta é a disposição em que podem surgir maiores dificuldades, tornando-se importante a modelação para permitir testes sobre está organização.

4.4 Modelação dos blocos funcionais

Os blocos funcionais relevantes para este exercício são o flanco ascendente, flanco descendente, contador incremental e decremental e o TOF, como já foi referido. Mas, também, já foram desenvolvidos modelos para o CTD, CTU e TON (Anexo I). Nesta abordagem procurou-se dividir os modelos por *templates* e torna-nos o mais reutilizáveis possível, para agilizar o processo de modelação de um sistema mecatrónico.

4.4.1 Flanco ascendente e descendente

O modelo do flanco ascendente deve ser capaz de, quando uma variável mudar do estado booleano zero para um, enviar um sinal também booleano, que informa o PLC que esta mudou de estado. Este

sinal apenas deve estar ativo durante um ciclo interno e voltar a zero em seguida. Este comportamento está representado na Figura 42. Para que tal seja possível é necessário que o modelo seja atualizado em todos os ciclos.

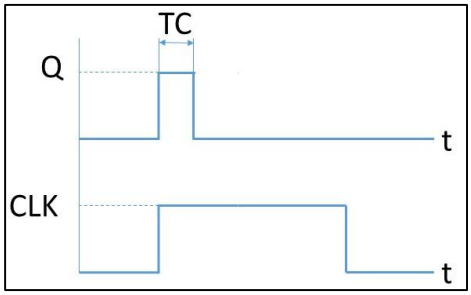

Figura 42 Representação do comportamento do flanco ascendente.

De notar que, quanto menor foi o tempo de ciclo (TC), maior será a precisão do equipamento e este varia de PLC para PLC. O modelo desenvolvido apresenta dois parâmetros, um que é a variável que pretende gerar o flanco (CLK) e outro que é o sinal booleano (Q), que apenas se ativa quando se registar a mudança de estado de zero para um em CLK. Estes elementos estão parametrizados no modelo, para assim se obter um comportamento semelhante ao dos blocos funcionais, como se pode ver na Figura 43.

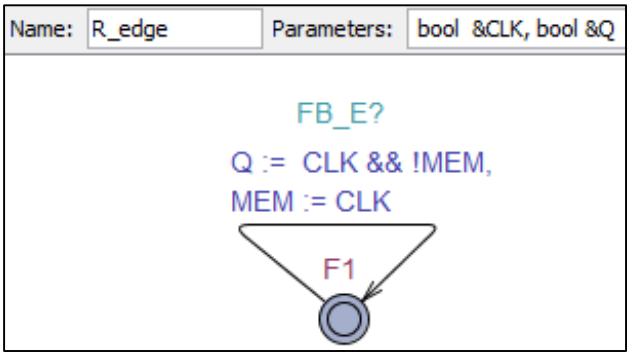

Figura 43 Modelo do flanco ascendente.

Assim, consegue-se reduzir os erros e o tempo necessário, pois o modelo é extremamente reutilizável. Com o mesmo modelo, é possível, ainda, implementar diversos flancos ascendentes necessários para este caso de estudo, como se pode ver na esquematização da Figura 40. A resposta deste modelo mostrou-se excelente, pois mesmo quando sujeita a uma variável aleatória, respondeu sempre corretamente.

Neste caso de estudo, foram utilizados quatro elementos deste tipo que nasceram do bloco parametrizado da Figura 43, de acordo com o apresentado na Figura 44.

$$R\_edge\_D = R\_edge(s\_down\_c, R\_s\_down)$$
$$R\_edge\_U = R\_edge(s\_up\_c, R\_s\_up)$$
$$R\_edge\_si = R\_edge(si\_c, R\_si)$$
$$R\_edge\_so = R\_edge(so\_c, R\_so)$$

Figura 44 Parametrização dos flancos ascendentes necessários.

Basicamente, para cada sinal de sensor foi gerado um flanco ascendente que é utilizado para validar as condições de transposição. Esta parametrização deu origem aos elementos da Figura 45, onde representa-se cada FB com os seus respetivos parâmetros. É importante salientar que os parâmetros utilizados como *input* são sempre variáveis internas do programa do PLC, e as suas saídas apenas são validas dentro do mesmo ambiente, para assim se conseguir simular a dinâmica de um programa realisticamente.

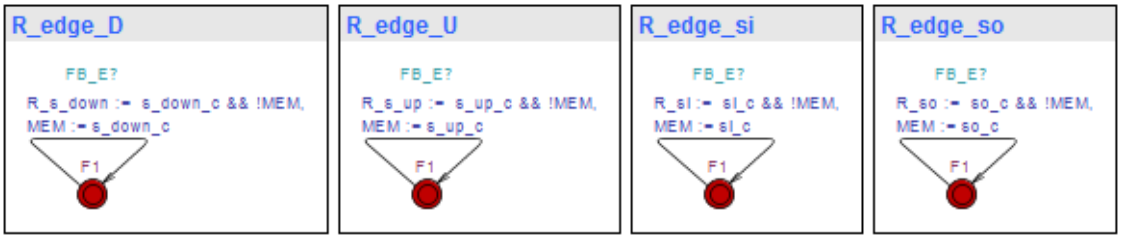

Figura 45 Modelos dos flancos ascendentes com parâmetros aplicados utilizados neste caso de estudo

Na Figura 45, constata-se que o elemento funciona com os parâmetros aplicados, é como se fossem "substituídos" para criar um novo elemento totalmente independente do anterior.

Por outro lado, o flanco descendente foi pensado para acionar um sinal também com a duração de um ciclo do PLC, quando uma determinada variável muda do estado lógico um para zero, como representado na esquematização da Figura 46.

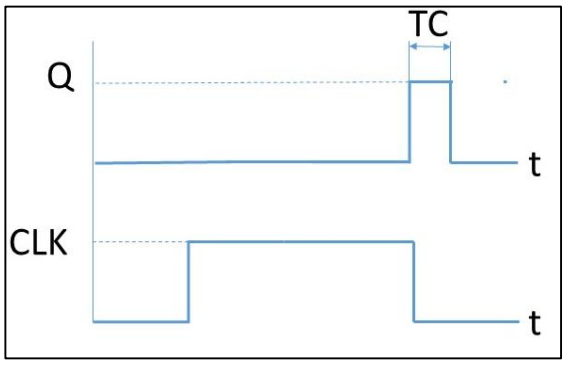

Figura 46 Esquematização do comportamento do flanco descendente

Tal como o anterior, quanto menor for o tempo TC mais rápida será a resposta do controlador. Foi desenvolvido, mais uma vez, um modelo parametrizado para permitir reutilização máxima do modelo, apesar de neste caso de estudo apenas ser necessário um para servir de sinal do contador decremental,

apresente os mesmos *inputs* e *outputs* do modelo do flanco ascendente, como se pode ver na Figura 47.

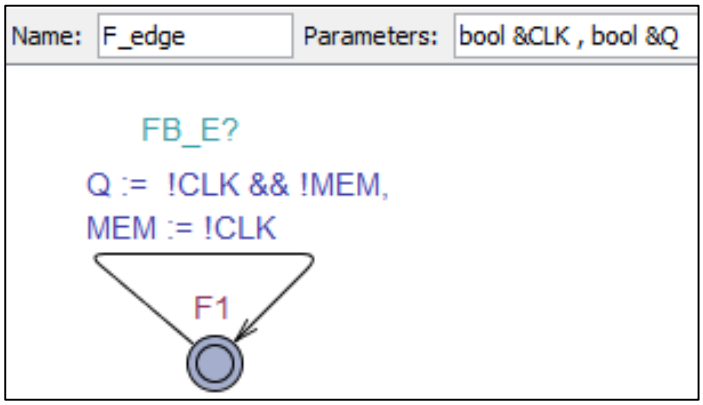

Figura 47 Modelo do flanco descendente.

Estes modelos são extremamente próximos do que acontece na realidade, porque se determinada variável realizar duas vezes a transição lógica de um para zero no mesmo ciclo do PLC, apenas uma delas é considerada. Deve, ainda, ter em atenção, que todos eles utilizam o mesmo canal de sincronização, sem qualquer efeito prejudicial.

4.4.2 TOF

A representação de tempo de forma realista tem sido uma preocupação para várias equipas de projeto, como referido anteriormente, mas podem existir situações em que o modelo do TON não seja suficiente, por isso foi preocupação desta dissertação desenvolver um modelo do TOF também. Este modelo foi, tal como todos os outros, parametrizado, tendo como *inputs* o intervalo de tempo (PT) e "IN", que representa o sinal que vai ser atrasado. Tem apenas uma saída booleana, que se manterá ativa enquanto o intervalo de tempo "PT" não tiver sido atingido. O modelo é composto por três estados: um em que está desligado (OFF), o estado seguinte é ativo quando o sinal que pretende atrasar o flanco descendente passar a ativo (*Standby*). Quando deixar de se registar este sinal o modelo atinge o estado ON, onde conta o tempo, e se antes do intervalo "PT" ser atingido, se registar novamente o sinal de "IN", este volta para o lugar *Standby*. Por outro lado, se o tempo for respeitado sem se registar este sinal, o modelo volta para o estado OFF. É importante salientar que isto é conseguido através de uma invariante que fará avançar o modelo para OFF, sem necessidade de sinal de sincronização. Mas, caso se registe "IN", e apenas nesse caso, a transição concorrente será validada e o bloco funcional volta para *Standby*, de acordo com o modelo da Figura 48.

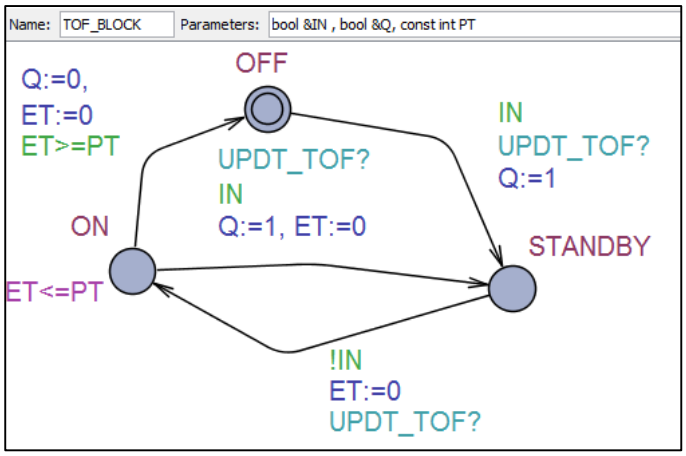

Figura 48 Modelo do TOF em TA

Neste caso, era necessário atrasar em 30 segundos após a barreira estar completamente aberta, para se garantir que os carros transpõem a mesma em segurança. Então, no programa utiliza-se um X3 que espera que este tempo seja transposto. Para se modelar este comportamento utilizado um TOF como parâmetro de entrada, a condição de transposição anterior X3, ou seja CC2, este funciona como o elemento acionador da contagem de tempo. A condição de transposição CC3 não é validada enquanto o TOF não ficar desativado. Esta situação deu origem ao seguinte parâmetros para o modelo TOF TOF_BLOCK(CC2, T_X3 , 30), onde CC2 é o *input*, T_X3 o *output* e o tempo de espera 30 unidades de tempo. Esta configuração originou o modelo da Figura 49.

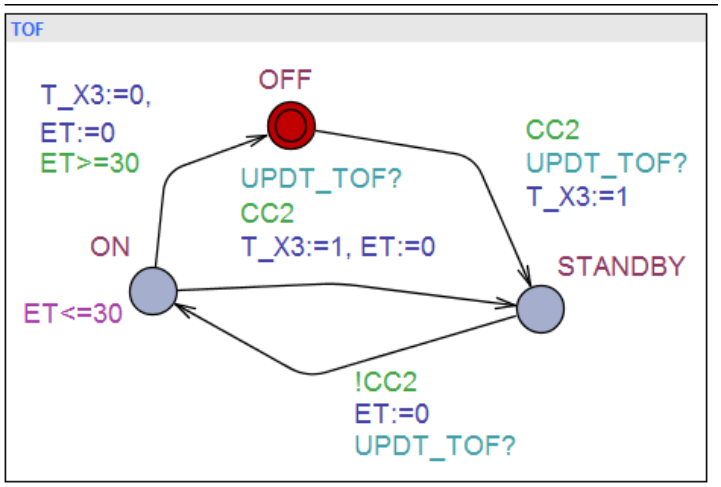

Figura 49 Modelo do TOF parametrizado aplicado ao caso de estudo

Este modelo apenas pode apresentar intervalos de tempo superiores ao tempo de ciclo do PLC, tal como acontece na realidade. Este modelo apresentou excelentes resultados e respondeu sempre corretamente e como foi parametrizado pode ser reutilizado em outros modelos, quando seja necessário modelar tempos.

4.4.3 Contador decremental e incremental

Este modelo combina a capacidade de somar e subtrair sobre a mesma variável que é o pretendido para manter um registo atualizado do número de carros existentes dentro do parque de estacionamento. Para se desenvolver este modelo optou-se por utilizar uma técnica em que cada tarefa que o modelo tem de realizar corresponde a uma transição. Desenvolveu-se, então, o modelo do referido bloco funcional de acordo com a Figura 50.

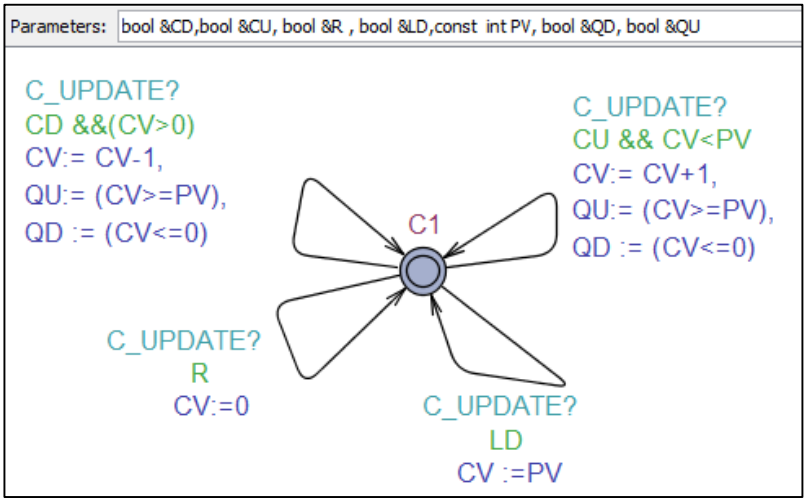

Figura 50 Modelo do contador decremental e incremental

Este modelo à primeira vista pode parecer um pouco complexo, mas na verdade é bastante simples. Como representado na Figura 27, o bloco funcional é composto por quatro condições, que correspondem a quatro transições; uma para reiniciar a contagem; outra para carregar um valor para a variável interna do modelo; uma transição responsável por realizar a contagem decremental; e por fim outra transição para incrementar. De acordo com os *inputs*, o modelo escolhe qual a transição a ativar em cada situação. Este efeito é conseguido utilizando guardas que correspondem as diferentes condições presentes no FB (ver Figura 27). Saliente-se ainda, tal como já referido, o processo de contagem prossupõem a utilização de um flanco ascendente para gerar o sinal CU (Figura 27), e um FB flanco descente para gerar o sinal CD. Este método permite que apenas se decremente ou incremente uma unidade por cada evento (ver Figura 51).

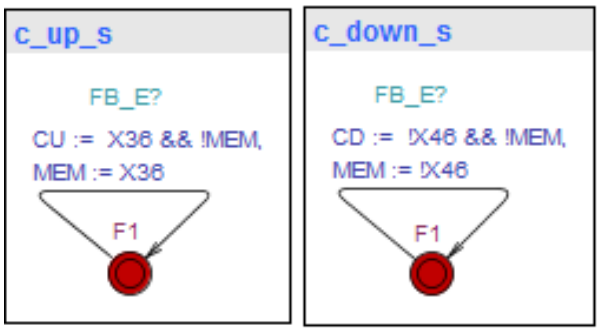

Figura 51 Modelos geradores dos sinais CU e CD.

Estes sinais são gerados de acordo com os SFCs paralelos da Figura 39 e, portanto, quando o X36 estiver ativo significa que o carro está no processo de entrada do parque de estacionamento. Por outro lado, se estiver ativo o X46, significa que um carro está a sair. Se ambos estiverem ativos, representa a situação em que estão a entrar e a sair dois carros simultaneamente. De acordo com isto, os flancos ascendente e descendente geram os sinais para se realizar o processo de contagem.

Atualizar os sinais booleanos que informam que os limites do FB foram atingidos, tem de ser atualizados sempre que se incrementa ou decrementa. Supondo que o sinal do limite mínimo estava ativo (QD) e foi somada uma unidade, este sinal tem de ser desativado. Por outro lado, se após se incrementar o limite máximo for atingido tem de ser atualizado o sinal "QU". Para se conseguir este efeito foi necessário colocar estes booleanos nas duas transições (QU /QD), como se pode ver na Figura 50. Por último, este modelo foi parametrizado e pensado para ser utilizado da mesma forma que os blocos funcionais funcionam, ou seja, não é necessário fazer qualquer alteração no seu interior. Considerou-se por comunidade a variável "CV", como sendo interna ao modelo, porque o modelo já apresenta bastantes parâmetros e o UPPAAL permite ver os valores de "CV" a qualquer momento.

Com a modelação destes blocos funcionais, desde logo, é possível testar um grupo muito superior de códigos e comportamentos. Todos eles se mostraram vantajosos e importantes para a modelação deste caso de estudo. É importante reter que todos os blocos funcionais modelados estão de acordo a norma IEC 61 131-3, e que poderão ser utilizados futuramente em Simulação e Verificação Formal de sistemas industriais utilizando o UPPAAL

A RETER DESTE CAPÍTULO

Neste capítulo utilizou-se uma barreira inteligente de um parque de estacionamento, como caso de estudo, cujo seu controlador é um PLC. Desenvolveu-se um programa em SFC dividido em três partes, uma com o código principal e outras duas para monitorizar se os carros estão a fazer o movimento de dentro para fora da barreira ou de fora para dentro, de acordo com a ordem de ativação dos sensores.

Para modelar este sistema mecatrónico utilizou-se, na parte física, um modelo para a barreira, com os sensores incorporados e para o motor dois modelos: um para o movimento de subida da barreira e outro para o movimento de descida da barreira. Neste exercício utilizaram-se quatro tipos de blocos funcionais, um flanco ascendente, um flanco descente, um TOF e um contador decremental e incremental.

Capítulo 5
APRESENTAÇÃO E DISCUSSÃO DOS RESULTADOS

Neste capítulo pretende-se apresentar os resultados obtidos de Simulação e Verificação Formal, quer no ponto de vista geral do sistema, quer do ponto de vista da resposta dada pelos blocos funcionais individualmente.

5. RESULTADOS OBTIDOS

5.1 Resultados de Simulação

A primeira análise que deve ser feita aos modelos, do ponto de vista de Simulação, deve se concentrar em procurar possíveis erros de digitação e gralhas.

Após estas correções, o passo seguinte é garantir que as ordens do controlador têm consequência nos atuadores, neste caso, trata-se do modelo do motor, pois este é responsável por controlar movimento da barreira. Portanto, tal como especificado no SFC da Figura 38, quando o "X2" estiver ativo será enviada a ordem "M_UP" e por outro lado quando "X5" estiver, corresponderá a ordem de recuo "M_D". Portanto, quando o "X2" estiver ativo, o modelo "M_motor_UP" deve estar no estado ON e quando "X5" estiver ativo o modelo "M_motor_DOWN" deve estar no estado ON, tal como na Figura 52.

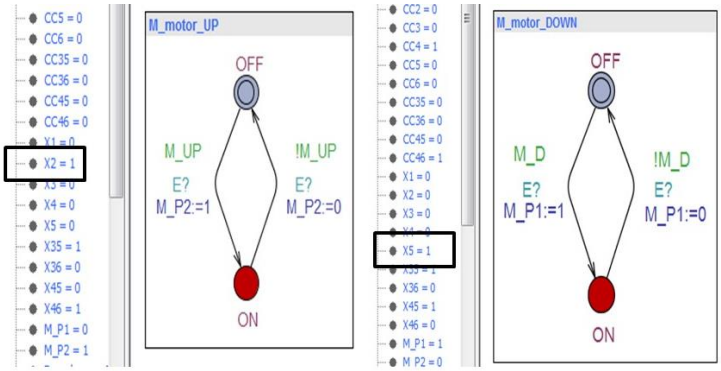

Figura 52 Consequência das ordens nos modelos do motor.

Verificou-se que este comportamento acontecia sempre, tendo, portanto, um funcionamento correto (Figura 52).Por outro lado, quando deixam de estar ativas as ordens, os modelos devem regressar ao estado OFF.

Estes, por sua vez, são responsáveis por controlar o movimento da barreira, portanto, quando o "M_motor_UP" estiver no lugar ON no modelo da barreira deve corresponder o lugar P4, correspondente à situação de subida.

Contrariamente, quando se dá ordem de descida, o modelo do "M_motor_DOWN", apresenta o lugar ON e a barreira, o lugar P6, que corresponde à descida da mesma para a posição de repouso (Figura 53). Em todas as outras situações, os modelos dos motores apresentam-se desativados.

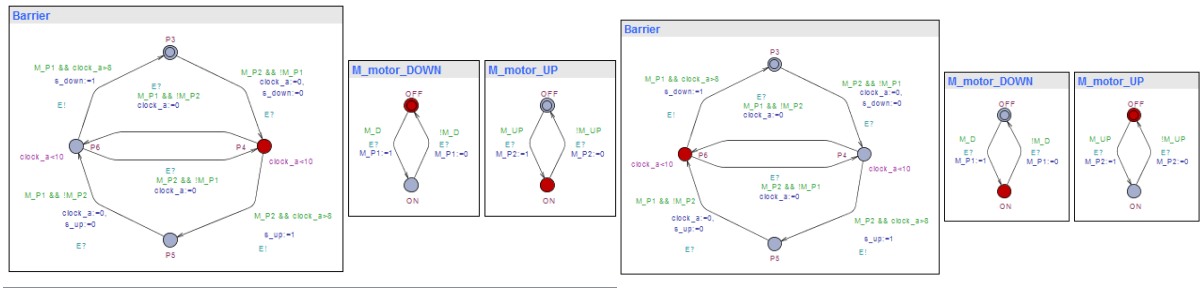

Figura 53 Correspondência dos lugares do motor com as da barreira.

Após verificar que os movimentos particulares apresentam o correto funcionamento, falta agora olhar para os modelos de uma forma mais global, para verificar se, realmente, correspondem ao especificado. Conclui-se que os modelos reagem como o esperado e cada variável de atividade de etapa corresponde a um comportamento de acordo com o SFC da Figura 38.

Falta ainda abordar a questão da gestão do número de carros presente no parque, para se controlar se o carro está a entrar ou sair é necessário observar com atenção a combinação entre os sensores, que posteriormente atualizará o número de carros no parque de estacionamento.

Denota-se, que o controlo da passagem é feito através da monotorização da combinação dos sensores. Quando o carro está a entrar, vai ativar primeiro o sensor fora da barreira (Alea_car_out) e só depois o dentro da barreira (Alea_car_in). No caso de estar a sair, o processo é o inverso (ver Figura 39). Contudo, para seguir o representado na IEC 61 311-3, o processo de contagem é díspar entre as duas situações. No caso de ser uma entrada, será incrementada uma unidade, quando o estado X36 é atingido, mas se estiver a sair será decrementada apenas quando o carro estiver acionado o segundo sensor, pois só assim ocorrerá o flanco descendente de X46.

Falta, agora, abordar a análise do ponto de vista de Simulação dos blocos funcionais utilizados. Dando especial atenção ao flanco ascendente, este vai reagir de acordo com os apresentados na Tabela 1, onde são apresentadas as reações deste elemento, de acordo com todas as situações. Importa, ainda, referir o sinal booleano de determinado flanco ascendente só se mantém ativo durante um ciclo interno do PLC, de acordo com o pretendido na Figura 42.

Por outro lado, o flanco descendente reagirá de uma forma diferente, de acordo com a Tabela 2, onde se apresentam todas as combinações de variáveis e como estas influenciam o comportamento deste elemento.

Por fim, o TOF, que é uma forma alternativa de modelar tempos de forma realista sem usar o TON. Para este elemento é necessário uma atenção mais cuidada, porque apresenta mais do que um lugar e estes têm influência uns nos outros. Este elemento tem, como *input*, CC2, que é a transição

anterior ao X3 e é validada quando a barreira ativa o sensor, informando que a posição completamente aberta foi atingida, consequentemente o TOF atinge o estado "Standby".

Segue-se o flanco descendente do CC2, que fará com que o TOF atinja o estado "ON", onde é iniciada a contagem de intervalo de tempo de 30 UT (*time units*). Após esse intervalo de tempo ser atingido, o TOF atinge o estado OFF e só volta estado "Standby" quando se voltar a registar o CC2.

5.2 Resultados da Verificação Formal

O processo de Verificação Formal neste exercício complementa duas partes, uma que se concentra na verificação do equipamento e a segunda parte, que concentra-se em garantir que os modelos dos blocos funcionais estão de acordo com o especificado na norma IEC 61 131-3.

O primeiro teste a ser feito é contra *deadlock*, que provou-se desde logo favorável, como demonstra a Figura 55, onde se constata uma resposta afirmativa do UPPALL.

De acordo com o especificado no subcapítulo 4.1, quando a barreira está em repouso e é detetada a presença de carro junto a esta no interior ou exterior, e o parque não está cheio, implica a abertura da mesma. Este comportamento deu origem à propriedade da Tabela 5.

Tabela 5 Propriedade desenvolvida para testar a abertura da barreira quando esta está em repouso.

Propriedade	Conversão para TCTL	Conversão para UPPAAL
A barreira está em repouso (X1) e é acionado o sensor interior ou exterior é c<20 implica sempre a abertura desta	AG((X1 ∧ ↑si) ∨ (X1 ∧ ↑so ∧ c<20) ⇒ EF (Barreira aberta))	(X1 && R_si) || (X1 && R_so && !QU_X36) –> Barrier.P5

Este é comportamento é o mais simples a ser aplicado e o verificador do UPPAAL mostrou que, de fato, os modelos desenvolvidos respeitavam está propriedade. É importante, ainda, garantir que quando o limite máximo do parque for atingido, a barreira, quando acionada pelo lado exterior, não levante, para que não entre nenhum veículo. Este comportamento deu origem à propriedade da Tabela 6.

Tabela 6 Propriedade da não abertura da barreira quando o parque está cheio.

Propriedade	Conversão para TCTL	Conversão para UPPAAL
Com o limite do parque for atingido a barreira não abre quando é acionado o sensor do lado exterior em repouso ou a descer.	AG((X1∧(c>20)∧↑so) ∨ (X5∧(c>20)∧↑so) ⇒ EF (barreira nunca levanta))	(X1&& R_so&&QU_X36) || (X5&& R_so&&QU_X36) –> !Barrier.P5

Quando o controlador está no X1, encontra-se em repouso, e em X5 em descida. Foi considerado como lotação do parque 20 veículos e quando este limite for atingindo, o bloco funcional, contador incremental e decremental, mantém ativo um sinal booleano, que é representado pela variável QU_X36. Estas duas propriedades foram validadas no verificador do UPPAAL, como demonstra a Figura 55.

Falta agora testar o comportamento modelado no processo de descida, neste caso, existem dois comportamentos possíveis. Aqui, se não se registar sinal dos sensores "si" ou "so" a barreira regressa à posição de repouso. Este procedimento deu origem à propriedade da Tabela 7.

Tabela 7 Propriedade da barreira a descer e não é detetado nenhum carro junto desta.

Propriedade	Conversão para TCTL	Conversão para UPPAAL
Barreira a descer (Barrier.P6) é não são acionados os sensores "so e si" durante este intervalo de tempo desta tarefa.	AG (((Barreira a descer ∧ (Ordem de recuo) ∧ (Tempo de recuo atingido)) ⇒ (EF (barreira em repouso))	(M_D&&(Barrier.clock_a>=10) && Barrier.P6) –> Barrier.P3

Se os sensores "si" e "so" não forem acionados, a ordem M_D vai se manter durante o processo de descida da barreira, portanto, é necessário provar este facto, o que deu origem à seguinte propriedade, descrita na Tabela 8.

Tabela 8 Propriedade que prova que se não for acionado o sensor "si" ou "so" a ordem "M_UP" mantém-se.

Propriedade	Conversão para TCTL	Conversão para UPPAAL
Barreira a descer (Barrier.P6) é não são acionados os sensores "so e si" implica a ordem de recuo mantém-se.	AG (((Barreira a descer ∧ $\uparrow\overline{so}$ ∧ $\uparrow\overline{si}$ ⇒ (EF M_D)	(Barrier.P6 && !R_so && !R_si) –> M_D

Estas duas propriedades também se verificaram no modelo desenvolvido (Figura 55), conseguindo-se, assim, verificar que se não existir acionamento dos sensores "si" ou "so" durante a descida da barreira, esta entra em repouso.

Falta provar um comportamento que o controlador deve apresentar, que será a passagem automática do estado de recuo para avanço, caso seja registado sinal nos sensores que detetam a presença de carro junto a este, desenvolvendo-se consequentemente a propriedade da Tabela 9.

Tabela 9 Propriedade da barreira a descer e é acionado um dos sensores "si" ou "so" levando esta a abrir.

Propriedade	Conversão para TCTL	Conversão para UPPAAL
Barreira a descer (Barrier.P6) é são acionados os sensores "so" e/ou "si" implicara a abertura da barreira.	AG ((Barreira a descer) ∧(↑si ∨ ↑so)⇒ (EF Barreira aberta))	Barrier.P6 && ((R_si) || (R_so && !QU_X36)) –> Barrier.P5

De facto, esta propriedade não se verifica, pois para se atingir o X5, é acionado um dos referidos sensores (si ou so). Existe a possibilidade da barreira estar a descer e esta combinação se verificar e o sistema não atingir o lugar correspondente à barreira aberta, sendo então necessário excluir a transição que leva a situação referida (CC4). Obteve-se a propriedade da Figura 54 onde é excluído o caso referido.

```
Barrier.P6 &&!CC4 (( R_si) || ( R_so && !QU_X36) ) –> Berrier.P5
```

Figura 54 Propriedade da tabela 9 corrigida.

Estas propriedades anteriormente referidas foram todas verificadas, como se pode ver na Figura 55.

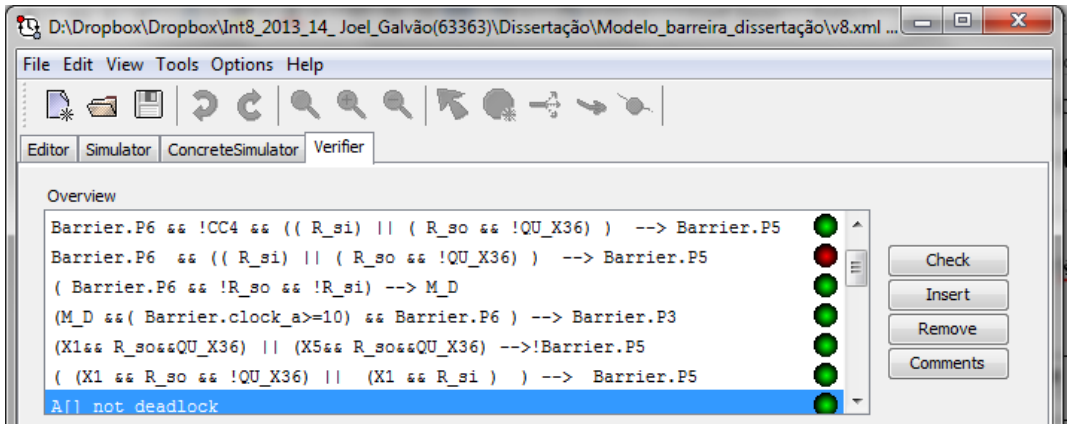

Figura 55 Propriedades testadas aplicadas ao modelo

A segunda parte deste trabalho concentra-se no desenvolvimento das propriedades necessárias para verificar o comportamento dos blocos funcionais.

Começando com as propriedades desenvolvidas para os flancos ascendentes presentes no modelo. Para estes elementos é necessário, que a propriedade garanta, que quando o sensor apresente a transição de zero para um, então a saída booleana Q do bloco é um. Portanto temos a propriedade da Tabela 10

Tabela 10 Propriedade dos flancos ascendentes aplicado no sensor si.

Propriedade	Conversão para TCTL	Conversão para UPPAAL
Antes da atualização do flanco ascendente do sensor "si", a sua variável do programa é um (si_c), a memória do bloco é zero e o booleano Q está desativado. Implicará sempre, (depois de atualizar), que o booleano Q toma o valor lógico de um.	AG ((Flanco sem atualizar) ∧ (si) ∧ memoria do bloco igual a zero)⇒ (EF(Q ∧ memória do bloco igual a um)))	(PLC_PROGRAM.C8 && si_c &&!R_edge_si.MEM)–> (PLC_PROGRAM.C9&&R_si && R_edge_si.MEM)

Esta propriedade segue o mesmo princípio nos outros cinco flancos ascendentes e deu origem às propriedades da Figura 56.

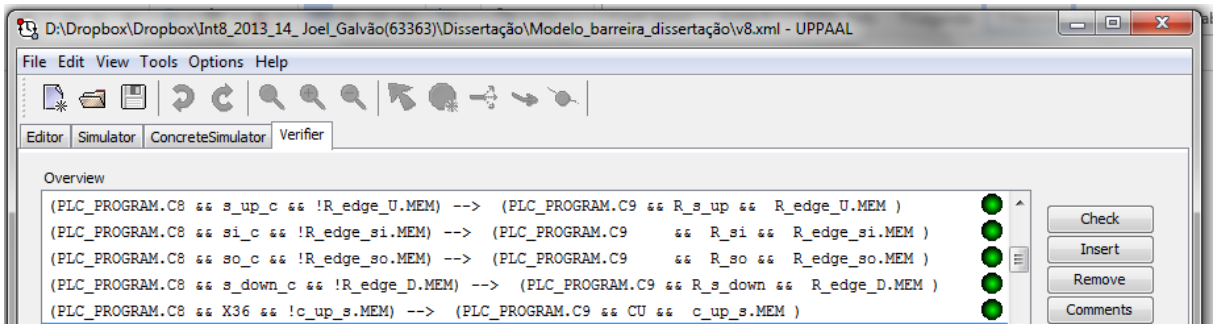

Figura 56 Propriedades dos flancos ascendentes aplicados a este caso de estudo.

O "PLC_PROGRAM.C8" é o lugar antes da atualização destes elementos, portanto ainda não tem em consideração as alterações realizadas pelo processo. Por outro lado, o lugar "PLC_PROGRAM.C9" é exatamente a seguir da atualização dos blocos, conseguindo-se assim, saber se o elemento está desatualizado ou não.

Passando para os contadores, aqui é importante garantir que quando o limite máximo predefinido for atingido seja ativado o sinal QU do contador, que originou a propriedade da Figura 57.

contador.CV>20 –> QU_X36

Figura 57 Propriedade de acionamento do sinal QU.

Falta, ainda, analisar o FB TOF, que basicamente apresenta dois comportamentos possíveis, quando está a contar tempo e a barreira se encontra aberta; e quanto o tempo já passou, que leva a barreira a ir para o estado onde está a descer. Estas propriedades encontram-se especificadas na Tabela 11.

Tabela 11 Propriedades aplicados ao Bloco Funcional TOF.

Propriedade	Conversão para TCTL	Conversão para UPPAAL
O TOF está no estado ON e o relógio ET ainda não atingiu o limite PT (30UT) mantendo-se a barreira aberta	AG((TOF.ON) ∧ (ET<30)⇒ (EF Barreira aberta))	(TOF.ON&&TOF.ET<30)–> Barrier.P5
O TOF está no estado ON e o relógio ET 'atingiu o limite PT (30UT) o que levará a barreira a recuar	AG ((TOF.ON) ∧ (ET>=30)⇒ (EF Barreira a fechar))	(TOF.ON&&TOF.ET>=30)–> Barrier.P6

Estas propriedades foram verificadas com êxito, e assim se conseguiu provar que os modelos dos blocos funcionais funcionam de acordo com o especificado na IEC 61 131-3.

A RETER DESTE CAPÍTULO

Neste capítulo demonstrou-se, através do caso de estudo, que a abordagem apresentada é bastante útil e fácil de ser utilizada. Provou-se ainda que todos os blocos funcionais modelados apresentam o comportamento esperado e são fáceis de implementar.

Capítulo 6
CONCLUSÃO

Neste capítulo são apresentadas as conclusões gerais de toda a dissertação desenvolvida, assim como as perspetivas do que pode ser desenvolvido nas etapas seguintes.

6. CONCLUSÕES

No processo de desenvolvimento de novos produtos, muitas vezes a indústria opta por uma metodologia mecatrónica de desenvolvimento do produto, que apresenta grandes vantagens, mas também novos desafios em termos de desenvolvimento. Este facto tem levado à procura de novas formas para agilizar este processo e, ao mesmo tempo, torná-lo mais seguro. Nesta dissertação, apresenta-se uma abordagem de análise de especificações de comando de sistemas mecatrónicos, através de Simulação por MiL e Verificação Formal por *Model Checking*. Sendo o principal objetivo resolver a lacuna existente nos modelos das POU dos programas de PLCs, mais precisamente, no que diz respeito aos blocos funcionais da norma IEC 61 131-3. Ao mesmo tempo, tem-se também como objetivo, garantir que os modelos desenvolvidos são reutilizáveis, para poderem ser utilizados em futuras análises.

Existem diversos tipos de controladores industriais, mas os mais utilizados, em quase todos os sistemas de controlo na indústria, são os PLCs. Foram desenvolvidas duas normas concorrentes, que podem ser utilizadas para desenvolver programas para estes autómatos, nomeadamente IEC 61 131 e IEC 61 449. Apesar do conceito de programação distribuída da IEC 61 449 ser bastante pertinente, a sua aplicação prática é muito reduzida. Por outro lado, a IEC 61 131 está completamente testada e implementada industrialmente e já existem engenheiros treinados com as suas linguagens. Realizar a troca de sistemas traria, portanto, custos muito elevados a nível de equipamentos e treino dos programadores. Pelas razões apresentadas é considerada a norma IEC 61 131-3 como referência para esta dissertação.

Diversas equipas de trabalho apresentam modelos para o bloco funcional TON, mas os restantes FBs propostos pela norma IEC 61 131-3, apenas são alvo de estudo pelas referências [41,55]. Contudo, não apresentam quais as considerações a ter para a sua conversão para o formalismo por eles utilizado e o método aplicado não dá a atenção necessária à parte física, pois é muito centrado na forma como é escrito o programa.

Foi considerado relevante desenvolver modelos para os blocos funcionais TON, TOF, flanco ascendente, flanco descendente, contador incremental, contador decremental e contador incremental e decremental. Todos os referidos blocos funcionais foram efetivamente convertidos para modelos na totalidade em TA.

Todos os modelos dos blocos funcionais foram parametrizados para agilizar o processo de modelação em futuros trabalhos, e foi possível simular e verificar formalmente o comportamento da especificação de comando do caso de estudo e blocos funcionais utilizados.

Futuramente, a investigação seguirá para o desenvolvimento de modelos de outros tipos de controladores industriais que não os PLCs. Existe, também, a necessidade de modelar a comunicação entre diferentes plataformas, para que seja possível aplicar esta metodologia a redes industriais complexas com diversos atuadores e unidades de controlo a interagir entre si.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] W. Bolton, "Introdução à mecatrônica," in *Mecatrônica: uma abordagem multidisciplinar*, 4ª ed São Paulo,Brasil:Bookman, 2010,pp. 11–13.

[2] J. Gausemeier and S. Moehringer, "VDI 2206-A new guideline for the design of mechatronic systems," apresentado no *IFAC Conference on Mechatronic Systems,*Berkeley, 2002.

[3] Y. Zhang, Y. Dong, H. Hong, and F. Zhang, "Code Formal Verification of Operation System," *I.J.Computer Network and Information Security*, vol. 2, pp. 10–18, December 2010.

[4] J. Campos and J. Machado, "A Specification Patterns System for Discrete Event Systems Analysis," *Int. J. Adv. Robot. Syst.*, vol. 10, 2013.

[5] J. M. Machado, "Influence de la prise en compte d'un modèle de processus en vérification formelle des Systèmes à Evénements Discrets,"Tese de Doutoramento, Universidade do Minho, Guimarães,2006.

[6] Programmable controllers - Part 3: Programming languages, IEC 61 131-3, 2003.

[7] K.-H. John and M. Tiegelkamp, *IEC 61 131-3: programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids*. 2nd ed. New York: Springer, 2010.

[8] B. Meenakshi, "Formal verification," *Resonance*, vol. 10, no. 5, pp. 26–38, 2005.

[9] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, December 1999.

[10] L. Baresi, S. Carmeli, A. Monti, and M. Pezzè, "PLC programming languages: A formal approach," *Proc. Autom.*, vol. 98, 1998.

[11] "MathWorks - MATLAB and Simulink for Technical Computing." [Online]. Disponivel em: http://www.mathworks.com/. [Acedido em: 8 de Abril de 2015].

[12] A. M. A. Al-Ahmari and K. Ridgway, "An integrated modelling method to support manufacturing systems analysis and design," *Comput. Ind.*, vol. 38, no. 3, pp. 225–238, 1999.

[13] B. K. Choi, K. H. Han, and T. Y. Park, "Object-oriented graphical modeling of FMSs," *Int. J. Flex. Manuf. Syst.*, vol. 8, no. 2, pp. 159–182, 1996.

[14] C. M. Park, S. M. Bajimaya, S. C. Park, G. N. Wang, J. G. Kwak, K. H. Han, and M. Chang, "Development of virtual simulator for visual validation of PLC program," in *Computational Intelligence for Modelling, Control and Automation, 2006 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, 2006, p. 32.

[15] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland, "Finite state automata and simple recurrent networks," *Neural Comput.*, vol. 1, no. 3, pp. 372–381, 1989.

[16] M. Barth and A. Fay, "Automated generation of simulation models for control code tests," *Control Eng. Pract.*, vol. 21, no. 2, pp. 218–230, 2013.

[17] Arena Simulation, "Discrete Event Simulation Software - Discrete Event Modeling." [Online]. Disponível em: https://www.arenasimulation.com/what-is-simulation/discrete-event-simulation-software. [Acedido em: 27 de Junho de 2015].

[18] Applied Materials, "AutoMod®." [Online]. Disponível em: http://www.appliedmaterials.com/global-services/automation-software/automod. [Acedido em : 27 Junho de 2015].

[19] S. C. Park, M. Ko, and M. Chang, "A reverse engineering approach to generate a virtual plant model for PLC simulation," *Int. J. Adv. Manuf. Technol.*, vol. 69, no. 9–12, pp. 2459–2469, 2013.

[20] L.-J. Koo, C. M. Park, C. H. Lee, S. Park, and G.-N. Wang, "Simulation framework for the verification of PLC programs in automobile industries," *Int. J. Prod. Res.*, vol. 49, no. 16, pp. 4925–4943, 2011.

[21] M.-S. Ko, D. Chang, G.-N. Wang, and S. C. Park, "The Template Model Approach For PLC Simulation In An Automotive Industry.," in *ECMS*, 2012, pp. 306–312.

[22] B. P. Zeigler, "DEVS representation of dynamical systems: Event-based intelligent control," *Proc. IEEE*, vol. 77, no. 1, pp. 72–80, 1989.

[23] I. Moon, G. J. Powers, J. R. Burch, and E. M. Clarke, "Automatic verification of sequential control systems using temporal logic," *AIChE J.*, vol. 38, no. 1, pp. 67–75, 1992.

[24] R. J. K. Jacob, "A state transition diagram language for visual programming," *IEEE Comput.*, vol. 18, no. 8, pp. 51–59, 1985.

[25] T. Hafer and W. Thomas, "Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree," in *Automata, Languages and Programming*, Springer, 1987, pp. 269–279.

[26] E. Carpanzano, L. Ferrucci, D. Mandrioli, M. Mazzolini, A. Morzenti, and M. Rossi, "Automated formal verification for flexible manufacturing systems," *J. Intell. Manuf.*, vol. 25, no. 5, pp. 1181–1195, 2014.

[27] A. Morzenti and P. San Pietro, "Object-oriented logical specification of time-critical systems," *ACM Trans. Softw. Eng. Methodol.*, vol. 3, no. 1, pp. 56–98, 1994.

[28] M. Pradella, "A User's Guide to Zot," *Log. Comput. Sci.*, 2009.

[29] I. E. Commission and others, “International Standard IEC 61 499, Function Blocks, Part 1-Part 4, IEC Jan. 2005.”

[30] V. Vyatkin and H.-M. Hanisch, “Verification of distributed control systems in intelligent manufacturing,” *J. Intell. Manuf.*, vol. 14, no. 1, pp. 123–136, 2003.

[31] S. Patil, S. Bhadra, and V. Vyatkin, “Closed-loop formal verification framework with non-determinism, configurable by meta-modelling,” in *IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society*, 2011, pp. 3770–3775.

[32] S. Patil, V. Vyatkin, and M. Sorouri, “Formal verification of intelligent mechatronic systems with decentralized control logic,” in *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*, 2012, pp. 1–7.

[33] R. S. Sreenivas and B. H. Krogh, “On condition/event systems with discrete state realizations,” *Discret. Event Dyn. Syst.*, vol. 1, no. 2, pp. 209–236, 1991.

[34] J. Cardoso and R. Valette, *Redes de Petri*. Editora da UFSC, 1997.

[35] V. Vyatkin, P. Starke, and H.-M. Hanisch, “ViVe and SESA Model Checkers,” 2007. [Online]. Disponível em: http://homepages.engineering.auckland.ac.nz/~vyatkin/tools/modelchekers.html. [Acedido em: 19 de Março de 2015].

[36] J. J. B. Machado, B. Denis, J.-J. Lesage, J.-M. Faure, J. Fereira, and others, “Logic controllers dependability verification using a plant model,” in *Proceedings of the 3rd IFAC Workshop on Discrete-Event System Design, DESDes’ 06, Rydzyna (Poland), 26-28 September 2006*, 2006.

[37] P. Wolper, “Temporal logic can be more expressive,” *Inf. Control*, vol. 56, no. 1, pp. 72–99, 1983.

[38] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, “NuSMV: A new symbolic model verifier,” in *Computer Aided Verification*, 1999, pp. 495–499.

[39] O. Pavlovic and H.-D. Ehrich, “Model Checking PLC Software Written in Function Block Diagram,” *2010 Third Int. Conf. Softw. Testing, Verif. Valid.*, pp. 439–448, 2010.

[40] B. F. Adiego, D. Darvas, E. B. Vinuela, J.-C. Tournier, V. M. G. Suárez, and J. O. Blech, “Modelling and Formal Verification of Timing Aspects in Large PLC Programs,” in *Proc. of IFAC World Congress*, 2014.

[41] R. Wang, Y. Guan, L. Liming, X. Li, and J. Zhang, “Component-based formal modeling of PLC systems,” *J. Appl. Math.*, vol. 2013, 2013.

[42] A. Basu, M. Bozga, and J. Sifakis, “Modeling heterogeneous real-time components in BIP,” in *Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on*, 2006, pp. 3–12.

[43] S. Bensalem, M. Bozga, T.-H. Nguyen, and J. Sifakis, "D-finder: A tool for compositional deadlock detection and verification," in *Computer Aided Verification*, 2009, pp. 614–619.

[44] M. Zhou, H. Wan, R. Wang, X. Song, C. Su, M. Gu, and J. Sun, "Formal component-based modeling and synthesis for PLC systems," *Comput. Ind.*, vol. 64, no. 8, pp. 1022–1034, 2013.

[45] R. ; Alur and D. Dill, "Automata for modeling real-time systems," *Proc. seventeenth Int. Colloq. Autom. Lang. Program.*, pp. 322–335, 1990.

[46] L. P. Assis Barbosa, K. C. Gorgônio, L. D. da Silva, A. M. N. de Lima, and A. Perkusich, "On the automatic generation of timed automata models from Function Block Diagrams for Safety Instrumented Systems," in *12th IEEE Conference on Emerging Technologies and Factory Automation - EFTA*, 2007, pp. 406–412.

[47] "Extensible Markup Language (XML) 1.1 (Second Edition)." [Online]. Disponível em: http://www.w3.org/TR/2006/REC-xml11-20060816/. [Acedido em: 4 Junho 2015].

[48] K. G. Larsen, M. Mikucionis, and B. Nielsen, "Uppaal tron user manual," *CISS, BRICS, Aalborg Univ. Aalborg, Denmark*, 2009.

[49] D. Soliman, K. Thramboulidis, and G. Frey, "Transformation of Function Block Diagrams to UPPAAL timed automata for the verification of safety applications," *Annu. Rev. Control*, vol. 36, no. 2, pp. 338–345, Dec. 2012.

[50] "PLCopen for efficiency in automation." [Online]. Disponível em: http://www.plcopen.org/. [Acedido em: 04 Junho de 2015].

[51] N. Bauer, S. Engell, R. Huuck, S. Lohmann, B. Lukoschus, M. Remelhe, and O. Stursberg, "Verification of PLC programs given as sequential function charts," in *Integration of software specification techniques for applications in Engineering*, Ehrig, H.,., Springer, 2004, pp. 517–540.

[52] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking in dense real-time," *Inf. Comput.*, vol. 104, no. 1, pp. 2–34, 1993.

[53] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a Nutshell," *Int. J. Softw. Tools Technol. Transf.*, vol. 1, no. 1, pp. 134–152, 1997.

[54] M. Perin and J.-M. Faure, "Building meaningful timed models of closed-loop DES for verification purposes," *Control Eng. Pract.*, vol. 21, no. 11, pp. 1620–1639, 2013.

[55] E. P. Enoiu, D. Sundmark, and P. Pettersson, "Model-based test suite generation for function block diagrams using the uppaal model checker," in *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*, 2013, pp. 158–167.

[56] A. R. de Souza, A. C. Paixão, D. D. Uzêda, M. A. Dias, S. Duarte, and H. S. de Amorim, "The Arduino board: a low cost option for physics experiments assisted by PC," *Rev. Bras. Ensino Fi̇sica*, vol. 33, no. 1, pp. 1–5, 2011.

[57] D. M. Considine and G. D. Considine, *Standard handbook of industrial automation*. Chapman & Hall, 1986.

[58] L. A. Bryan and E. A. Bryan, "Introduction to Programmable Controllers," in *Programmable controllers: theory and implementation*, 2002, pp. 4–32.

[59] W. Bolton, *Programmable logic controllers*. Newnes, 2009.

[60] J. N. Pires, *Automação Industrial*, vol. 2. ETEP, 2003.

[61] L. A. Bryan and E. A. Bryan, "Introduction to IEC 1131 Standard and Programming Language," in *Programmable controllers: theory and implementation*, 2002, pp. 374–439.

[62] Bosch Rexroth. The Drive & Control Company, "Understanding the IEC 61 131-3 Programming Languages," *Control Eng.*, p. 6, 2009.

[63] R. N. Nagel and R. Dove, *21st century manufacturing enterprise strategy: An Industry-Led View*. DIANE Publishing, 1991.

[64] V. Vyatkin, "The IEC 61 499 standard and its semantics - Bridging the Gap Between PLC Programming Languages and Distributed Systems," *Ind. Electron. Mag. IEEE*, vol. 3, no. 4, pp. 40–48, 2009.

[65] A. Zoitl and V. Vyatkin, "IEC 61 499 Architecture for Distributed Automation: the 'Glass Half Full'View," *IEEE Ind. Electron. Mag.*, vol. 3, no. 4, pp. 7–23, 2009.

[66] P. Vvba, P. Tichý, P. Mařík, V. Hall, K.H. Staron, R.J. Maturana, F.P. Kadera, "Rockwell Automation Holonic and Multiagent Control Systems Compendium," in *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 2011, vol. 41, no. 1, pp. 14–30.

[67] G. Ćengić and K. Åkesson, "On formal analysis of IEC 61 499 applications, Part A: Modeling," *Ind. Informatics, IEEE Trans.*, vol. 6, no. 2, pp. 136–144, 2010.

[68] K. Thramboulidis, "IEC 61 499 vs. 61131: A Comparison Based on Misperceptions," Patras, Greece, 2013.

[69] A. Zoitl, T. Strasser, C. Sunder, and T. Baier, "Is IEC 61 499 in Harmony with IEC 61 131-3?," *Ind. Electron. Mag. IEEE*, vol. 3, no. 4, pp. 49–55, 2009.

[70] G. Behrmann, A. David, K. G. Larsen, J. Hakansson, P. Petterson, W. Yi, and M. Hendriks, "UPPAAL 4.0," in *Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on*, 2006, pp. 125–126.

[71] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge University Press, 2004.

[72] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2. pp. 183–235, 1994.

[73] R. Alur, "Timed automata," in *Computer Aided Verification*, 1999, pp. 8–22.

[74] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," in *Lectures on Concurrency and Petri Nets*, Springer, 2004, pp. 87–124.

[75] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, *Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems*. Springer, 1993.

[76] E. A. Emerson and E. M. Clarke, "Using branching time temporal logic to synthesize synchronization skeletons," *Sci. Comput. Program.*, vol. 2, no. 3, pp. 241–266, 1982.

[77] M. Reynolds, "An axiomatization of full computation tree logic," *J. Symb. Log.*, vol. 66, no. 03, pp. 1011–1057, 2001.

[78] G. Behrmann, J. Bengtsson, A. David, K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal implementation secrets," in *Formal Techniques in Real-Time and Fault-Tolerant Systems*, 2002, pp. 3–22.

[79] G. Behrmann, A. David, and K. Larsen, "A Tutorial on Uppaal," in *Formal Methods for the Design of Real-Time Systems*, vol. 3185, Springer Berlin Heidelberg, 2004, pp. 200–236.

[80] J. S. Carvalho and S. M. de Sousa, "Tutorial de Uppaal," in *Universidade da Beira Interior-Publicação Interna*, 2009, pp. 1–15.

[81] N. M. E. Canadas, "Modelação da parte física de sistemas mecatrónicos e estudo da sua influência em Simulação MiL (Model-in-the-loop),"Dissertação de Mestrado, Universidade do Minho, 2013.

[82] S. Demers, P. Gopalakrishnan, and L. Kant, "A generic solution to software-in-the-loop," in *Military Communications Conference, 2007. MILCOM 2007. IEEE*, 2007, pp. 1–6.

[83] O. Meister, N. Frietsch, J. Seibold, and G. F. Trommer, "Software-in-the-Loop Simulation for Small Autonomous VTOL UAV with Teaming Capability," *Inst. Syst. Optim. Ger.*, 2008.

[84] D. Chioran and J. M. Machado, "Design of a Mechatronic System for Application of Hardware-in-the-loop Simulation Technique," 2011.

[85] ADI | Solutions in Real Time, "What is Hardware-in-the-Loop Simulation?" [Online]. Disponível em: http://www.adi.com/technology/tech-apps/what-is-hardware-in-the-loop-simulation/. [Acedido em: 1 Setembro de 2015].

[86] OPAL RT Developers, “About Hardware-In-the-Loop Simulation.” [Online]. Disponível em: http://www.opal-rt.com/about-hardware-loop-simulation. [Acedido em: 1 Setembro de 2015].

[87] J. Machado, J. Galvão, and A. Fernandes, “Function blocks and Formal verification: systematic modelling approach,” in *23rd ABCM International Congress of Mechanical Engineerin*, Rio de Janeiro, 2015.

Anexo I
MODELOS DOS BLOCOS FUNCIONAL ADICIONAIS

Este capítulo pretende apresentar modelos dos blocos funcionais TON, contador incremental e contador decremental também desenvolvidos utilizando exercício simples que permite testar todas as funcionalidades que estes FBs devem apresentar.

ANEXO I – MODELOS DOS BLOCOS FUNCIONAL ADICIONAIS

Estes blocos funcionais são também importantes de ser convertidos para Autómatos Finitos Temporizados para posteriormente serem utilizados em futuros trabalhos de Simulação e Verificação Formal de sistemas mecatrónicos, pois são também necessários em algumas situações. Para testar estes elementos foi desenvolvido um SFC (Figura 58), com uma sequência de cilindros que começa por mandar avançar o cilindro "B", seguindo-se o avanço e recuo do cilindro "A" cinco vezes consecutivas. Por fim, o cilindro "B" recua seguindo-se o avança e recuo quatro vezes do "C". Colocou-se, também, um atraso de 50 unidades de tempo no X5.

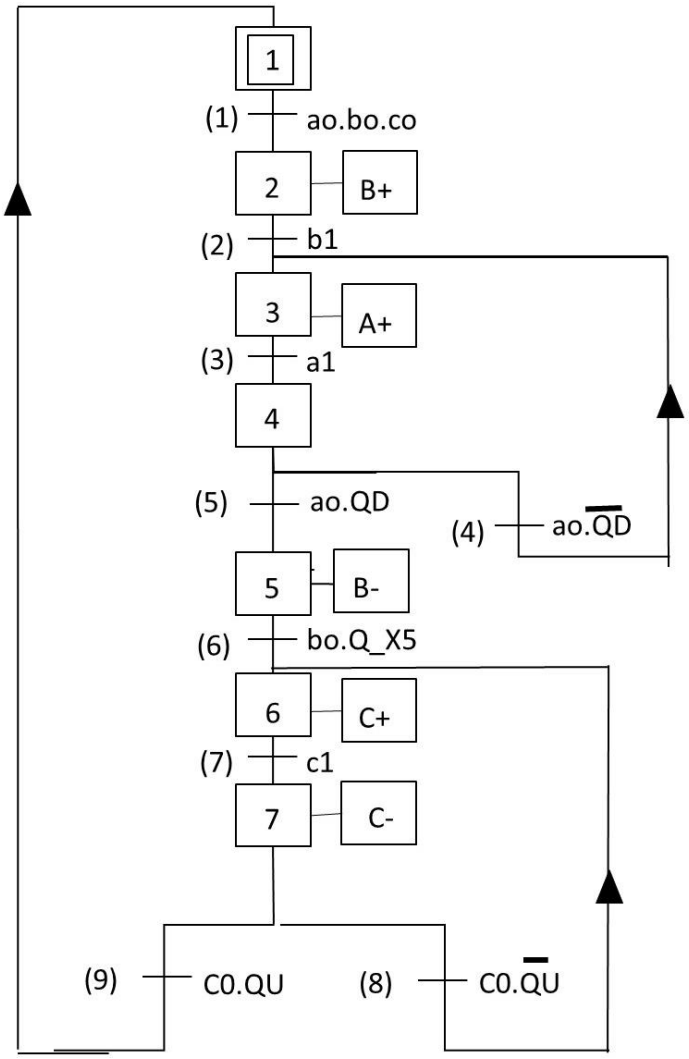

Figura 58 SFC com sequência pneumática simples

Este SFC deram origem às seguintes equações lógicas implementadas no UPPAAL da tabela 11, já convertidos para UPPAAL, estas já partem do princípio que serão utilizados os blocos CTD e CTU para contar os ciclos e que enviam uma variável booleana quando o limites destes elementos for atingindo,

informando o controlador que os ciclos dos cilindros já foram efetuados Para gerar o atraso de 50 UT utilizando um TON. Como parâmetro de entrada deste elemento foi colocado o X5 e na CC5 o sinal de saída deste FB para que apenas quando o referido tempo tiver passado o SFC passe para a ação seguinte.

Tabela 12 SFC implementado no UPPAAL consequência da figura 66

Condições de transposição	Ações
CC1:=X1 && a0_c && b0_c && c0_c,	X1:=CC9 || X1 && !CC1,
CC2:=X2 && b1_c,	X2:=CC1 || X2 && !CC2,
CC3:=X3 && a1_c,	X3:=CC2 || X3 && !CC3 || CC4,
CC4:=X4 && a0_c && !QD ,	X4:=CC3 || X4 && !(CC4 || CC5) ,
CC5:=X4 && a0_c && QD,	X5:=CC5 || X5 && !CC6,
CC6:= X5 && b0_c && Q_X5,	X6:=CC6 || X6 && !CC7 || CC8,
CC7:= X6 && c1_c,	X7:=CC7 || X7 && !(CC8 || CC9)
CC8:= X7 && c0_c && !QU,	
CC9:= X7 && c0_c && QU	

O modelo desenvolvido é composto por diversos submodelos dos diversos subsistemas deste sistema, é composto pelos tradicionais dois modelos para representar o comportamento do PLC, de acordo com a referência [5]. Na parte física consideraram-se três modelos, um para cada cilindro e par de sensores, de acordo também com o já abordado no capítulo 3.1 na Figura 11.

Para contar, foi desenvolvido, o modelo do bloco funcional contador incremental para contar os ciclos do cilindro "A". Este elemento foi parametrizado como todos os outros modelos, apresentado de acordo com a Figura 59.

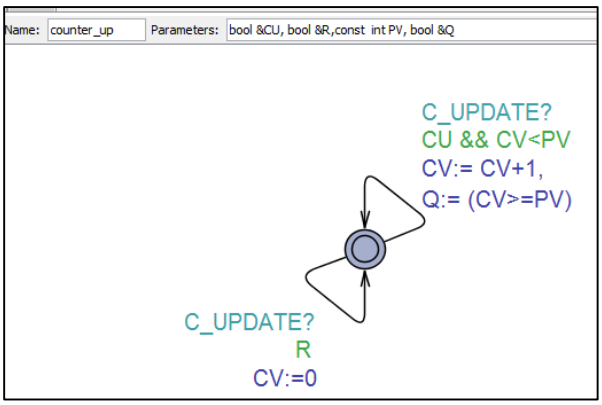

Figura 59 Modelo parametrizado do CTU

Este elemento apresenta, como *inputs,* o sinal para incrementar CU, que é gerado por um bloco funcional flanco ascendente, tal com acontece no CTDU, apresentado anteriormente na Figura 51. Um inteiro, que define o número de vezes que se quer incrementar e um sinal, que ordena o reinício do contador (R) e por fim, um booleano Q, que informa que o limite foi atingido. Este aplicado ao caso de estudo deu origem ao elemento da Figura 60.

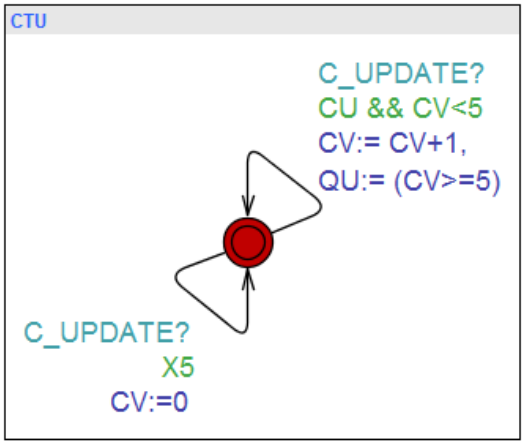

Figura 60 Modelo do contador incremental com parâmetros aplicados.

Este FB é bastante mais simples que o CTDU, porque não se está a efetuar processos de contagem paralelos sobre a mesma variável. Portanto, quando a recebido o sinal CU do flanco ascendente (dinâmica igual a apresentada no CTDU) e o limite PV não foi atingido, neste exercício é 5, ou seja CV não atingiu o valor de 5, incrementa uma unidade sobre a referida variável. Quando CV é igual a PV é necessário reiniciar o modelo, neste caso este é feito no X5.

Para contar os ciclos do cilindro "C", optou-se por utilizar um contador decremental. Neste caso, o sinal para se decrementar é dado por um flanco descendente (processo igual ao utilizado no CTDU para geral o CD da Figura 51) e por sua vez é contado até se atingir um limite mínimo iniciando-se num limite superior predefinido pelo programador. Este deu origem ao modelo da Figura 61.

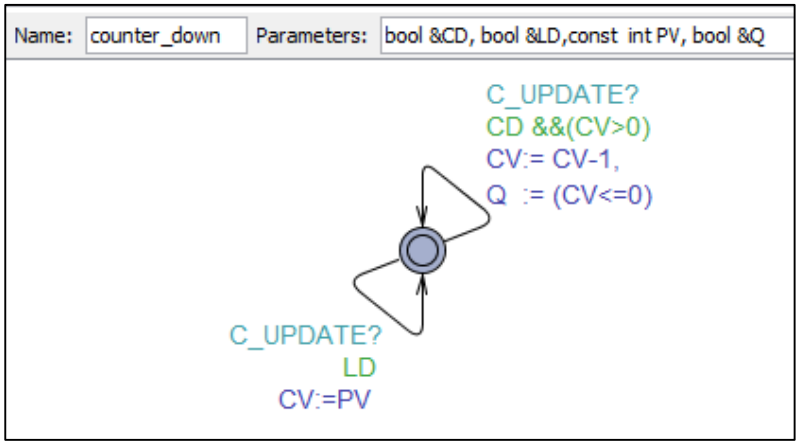

Figura 61 Modelo do CTU parametrizado

Neste caso, foi considerado, tal como no CTDU, o limite mínimo de contagem como sendo zero e, portanto, este só para de decrementar quando CV atingir este número. Para se voltar a decrementar, é necessário reiniciar o valor de PV. Este modelo tem a vantagem de poder definir valores iniciais diferentes durante a execução, o que pode ser vantajoso, em algumas situações, para se contar a partir de valores diferentes, usando o mesmo modelo. Aplicando este modelo aos seus parâmetros obteve-se o modelo da Figura 62.

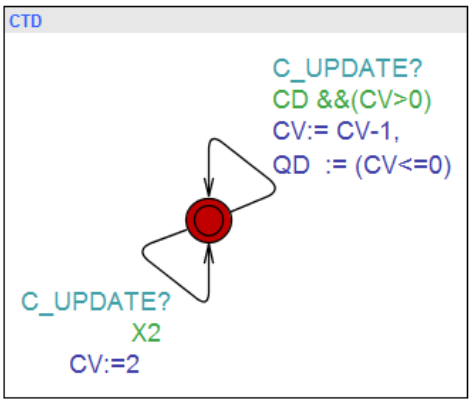

Figura 62 Modelo do CTD com os parâmetros aplicados

Neste modelo o processo de reiniciação consiste em voltar a atribuir a PV o seu valor inicial, que neste caso, será realizado no X2. Falta, ainda, referir que os dois modelos são atualizados através da mesma variável sem qualquer desvantagem.

O bloco funcional TON foi bastantemente trabalhado por diversas equipas, como já foi apresentado, mas também, é necessário desenvolver um modelo para ser posteriormente utilizado em outros trabalhos, com o objetivo de simular e verificar formalmente sistemas mecatrónicos. Este trabalho deu origem ao modelo Figura 63.

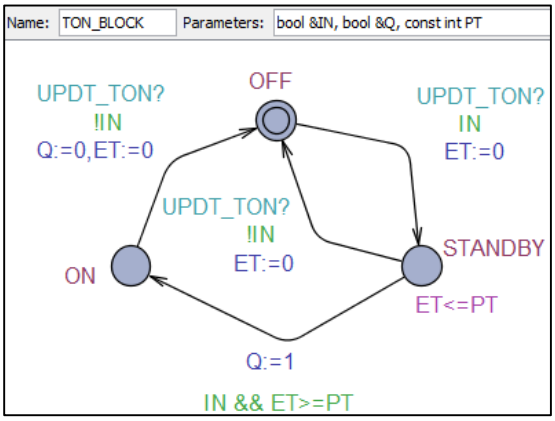

Figura 63 Modelo do TON desenvolvido

Este modelo é bastante semelhante ao apresentado na referência [54] e já se apresenta amplamente testado e aceite pela comunidade académica. O trabalho neste elemento concentrou-se, sobretudo, em

garantir que todos os comportamentos estão, de facto, de acordo com o especificado da norma IEC 61 131-3 e por fim parametriza-lo, para que não sejam necessárias alterações quando este bloco funcional seja preciso em alguma futura análise. Evitando-se assim erros e reduzindo o tempo a desenvolver os modelos para futuros trabalhos. Para utilizar este elemento, é necessário definir qual é o seu IN, ou seja, o sinal que terá o seu estado atrasado, o intervalo de tempo PT (tempo de atraso que o IN será sujeito) e por fim, definir o booleano Q. Aplicado a este caso de estudo, obtém-se o modelo com os parâmetros aplicados da Figura 64.

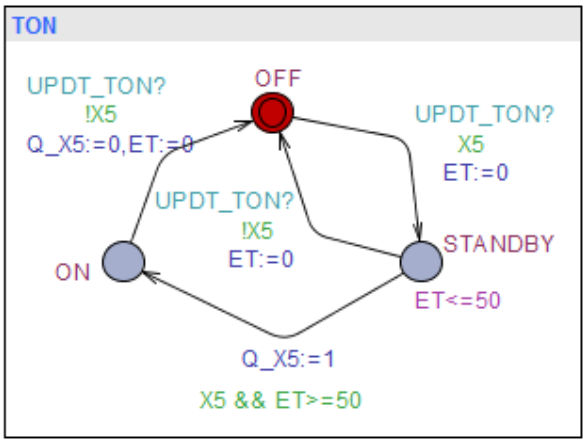

Figura 64 Modelo do TON com parâmetros aplicados.

A RETER DESTE CAPÍTULO

Neste anexo apresenta-se os modelos dos blocos funcionais TON, Contador Decremental e Contador Incremental criados em TA, que poderão ser necessários futuramente em Simulação e Verificação Forma de Sistemas Mecatrónicos.