FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO


# Atualização do FEUPAutom e criação de software simulador / SCADA genérico

**Bruno Miguel Fernandes Augusto**

PARA APRECIAÇÃO POR JÚRI

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Professor Doutor Armando Jorge Miranda de Sousa

29 de Junho de 2015

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

**MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES** | **2014/2015**

A Dissertação intitulada

"Atualização do FEUPAutom e Criação de Software Simulador / SCADA Genérico"

foi aprovada em provas realizadas em 17-07-2015

**o júri**

Presidente Professor Doutor Mário Jorge Rodrigues de Sousa
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor José Luís Magalhães Lima
Professor Adjunto do Departamento de Eletrotécnia da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Bragança

Professor Doutor Armando Jorge Miranda de Sousa
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

Bruno Augusto

Autor - Bruno Miguel Fernandes Augusto

Faculdade de Engenharia da Universidade do Porto

# Resumo

A simulação é, hoje em dia, usada em muitas aplicações com grandes benefícios, quer para ensino, quer em aplicações industriais. No ensino existem ainda poucas ferramentas que possibilitem a rápida e ágil simulação de sistemas simples e respetivo controlo na lógica dos sistemas orientados a eventos discretos, no espetro dos Autómatos Programáveis. Esta é a área de intervenção da Unidade Curricular (UC) 'Sistemas e Automação' do MIEEC da FEUP.

Nesta Dissertação desenvolveu-se a ferramenta FEUPSim, um simulador e SCADA, que comunica com o softPLC FEUPAutom também ele atualizado no âmbito deste trabalho. Foi utilizado o FreePascal/Lazarus como ferramenta de programação livre e aberta com o propósito de obter aplicações *cross-platform*. Foi utilizado ainda o protocolo de comunicações ModBusTCP para tirar proveito de comunicações normalizadas permitindo que as ferramentas desenvolvidas comuniquem com outros equipamentos e aplicações.

O FEUPSim é uma ferramenta que permite ao seu utilizador criar entidades tal como figuras geométricas, imagens, sensores, etc., onde o aspeto e o movimento ao longo do tempo dessas entidades depende de entradas e saídas do FEUPAutom. Como estratégia para manter facilidade de utilização, o sistema foi mantido em 2D apesar de poder futuramente ser alargado a 3D. Como SCADA, o FEUPSim pode ser ligado a equipamentos reais ou outras aplicações e permite a visualização interessante (incluindo animações) de um sistema remoto. Existe ainda a possibilidade do registo temporal dos detalhes relevantes (histórico do sistema). As facilidades de manipulação de projetos do FEUPSim e do FEUPAutom foram criadas/atualizadas por forma a tirar proveito da gravação de ficheiros XML, sem que se perdesse compatibilidade com projetos antigos.

O FEUPSim foi testado com sucesso através de diversos casos de estudo: um semáforo, um elevador e um limpa para-brisas. A aplicação foi também testada pelos estudantes da UC mencionada e validada na forma de inquéritos, que revelaram um elevado interesse e aceitação relativamente ao trabalho desenvolvido. Espera-se assim que os professores e os próprios estudantes de futuras ocorrências da UC possam criar as suas próprias simulações.

No fim desta dissertação são tiradas conclusões dos benefícios que tanto a atualização do FEUPAutom como a criação do simulador FEUPSim trazem à UC. As duas novas aplicações foram extensivamente testadas em Windows, tendo sido ainda realizados alguns testes com sucesso em Linux e espera-se brevemente fazer testes sob a plataforma Mac OS X. As ferramentas desenvolvidas serão disponibilizadas sem custos à comunidade.

O trabalho futuro inclui integrar as duas aplicações para estudo aprofundado da interação tempo real, estudar o alargamento para outros protocolos de comunicação, testes em ambientes reais e eventualmente automatizar a validação de programas de controlo.

**Palavras chave:** Simulação, SCADA, PLC, SoftPLC, Automação, Ensino, Eventos Discretos

# Abstract

The simulation is nowadays used in many applications with great benefits both for teaching and industrial applications. In education there are few tools that enable the fast and agile simulation of simple systems and respective control of the systems oriented to discrete event, in the spectrum of PLCs. This is the area of intervention of the course 'Systems and Automation' from the program MIEEC at FEUP.

In this Dissertation it was developed the FEUPSim tool, a simulator and SCADA, which communicates with the SoftPLC FEUPAutom who also was updated in this work. It was used the FreePascal/Lazarus as free and open programming tool in order to obtain cross-platform applications. It was also used ModbusTCP communications protocol to take advantage of enabling communications standard developed tools to communicate with other devices and applications.

FEUPSim is a tool that allows the user to create entities such as geometric shapes, images, sensors, etc., where the look and movement over time of these entities depends on inputs and outputs of FEUPAutom. As a strategy to maintain ease of use, the system was kept 2D although it may be extended in the future to 3D. As SCADA the FEUPSim can be linked to real equipment or other applications and allows interesting viewing (including animation) of a remote system. There is also the possibility of temporal record of relevant details (history system). The handling features of FEUPSim and FEUPAutom projects were created/updated in order to take advantage of the XML file recording, without losing retro-compatibility with older projects.

FEUPSim was tested with success through several case-studies: a traffic light, a lift, a clean wind-shield. The application was also tested by the students of course mentioned and validated in the form of surveys, which have shown the high interest and acceptance in relation to the work developed. It is expected that teachers and the students themselves for future occurrences of the course can create their own simulations.

At the end of this Dissertation, conclusions are drawn to both the update of FEUPAutom and the creation of FEUPSim simulator bring to course. The two new applications were extensively tested on Windows and there were also some successfully tests on Linux, and is expected to make tests under Mac OS X in the near future. The platform tools will be made available free of charge to the community.

Future work includes integrating the two applications for in-depth study of real-time interaction, consider extending to other communication protocols, tests in real environments and eventually automate the validation of monitoring evaluations.

**Keywords:** Simulation, SCADA, PLC, SoftPLC, Automation, Education, Discrete Events

# Agradecimentos

Em primeiro lugar quero agradeço aos meus pais por todo o apoio ao longo destes 23 anos, onde foram o pilar de toda a minha vida pessoal e académica. Foram muitas as horas que perderam comigo sem nunca exigir menos que máximo e, graças a eles, encontro-me aqui nesta última etapa antes de partir para o mundo laboral.

À minha namorada Ana Miranda (aka Licas) quero agradecer por todo o apoio que me deu ao longo destes não tão curtos 5 meses de dissertação, pela paciência de me ensinar a diferença entre o tem e o têm, por todas as horas ao meu lado na sala i105 onde passávamos, por vezes, tardes inteiras sem nunca falar (pois cada um estava a concentrado no seu trabalho) e, finalmente, pelos passeios de 10 mins que fazíamos quando eu começava a stressar demasiado porque algo não funcionava como eu tinha projetado.

Quero ainda agradecer ao grande Xóvitor por vir sempre à sexta-feira com histórias novas para contar, por toda a amizade demonstrada ao longo do curso onde fizemos um site de venda de tabletes, um *mini-segway* bêbado ou até mesmo um trabalho de 20 valores sobre o comboio mais rápido do mundo!!! Ao Miguel Fernandes (Sr. mete-se em tudo) por ser um stressado maior do que eu com o seu próprio trabalho, fazendo com que os meus problemas parecessem menores ao tentar ajudá-lo. Ao Maqtista por chegar todos os dias à i105 com boa disposição e sempre com uma música nova para pôr na cabeça de todos os presentes. E claro a todos os meus amigos que me ajudaram a chegar até aqui!

Por fim, gostaria de agradecer ao meu incansável orientador, o Doutor Armando Sousa por ter acreditado e confiado em mim para a realização desta dissertação e mesmo quando o seu tempo livre tendia para o negativo, arranjar sempre um bocadinho para me ajudar e ter as reuniões semanais que me mantiveram no ritmo certo!

Bruno Augusto

*"What we do for ourselves dies with us.*
*What we do for others and the world remains and is immortal."*

Albert Pine

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Abreviaturas e Símbolos

| | |
|---|---|
| API | Application Program Interface (Interface de Programação de Aplicações) |
| DWS | Delphi Web Script |
| FE | Falling Edge |
| FEUP | Faculdade de Engenharia da Universidade do Porto |
| GRAFCET | Graphe Fonctionnel de Commande, Étapes Transitions |
| HMI | Human Machine Interface (Interface Homem-Máquina) |
| IDE | Integrated Development Environment (Ambiente de Desenvolvimento Integrado) |
| IEC | International Electrotechnical Commission |
| MIEEC | Mestrado Integrado em Engenharia Eletrotécnica e de Computadores |
| ODE | Open Dynamics Engine |
| PLC | Programmable Logic Controller |
| PS | Pascal Script |
| RE | Rising Edge |
| RTU | Remote Terminal Unit (Unidade Terminal Remota) |
| SCADA | Supervisory Control And Data Acquisition (Supervisão, Controlo e Aquisição de Dados) |
| SFC | Sequential Function Chart |
| SO | Sistema Operativo |
| ST | Structured Text |
| UC | Unidade Curricular |
| XML | Extensible Markup Language |

# Capítulo 1

# Introdução

## 1.1 Enquadramento

Na Unidade Curricular (UC) 'Sistemas e Automação' do segundo ano do Mestrado Integrado em Engenharia Eletrotécnica e de Computadores (MIEEC) da Faculdade de Engenharia da Universidade do Porto (FEUP), são apresentados aos estudantes os principais conceitos dos sistemas de automação modernos. O programa é vasto e inclui uma grande variedade de exemplos práticos tais como a implementação de Máquinas de Estado, Grafcets, Redes de Petri entre outros, em que os alunos podem aplicar os conhecimentos adquiridos.

No entanto, a visualização dos resultados do controlo do sistema automático criado fica, por vezes, aquém das expectativas, não existindo ainda uma interface onde estudantes ou professores consigam projetar de forma simples um simulador dos elementos que foram abstratamente controlados. Na indústria existem *softwares* específicos que permitem a simulação de diferentes máquinas, robôs e até fábricas inteiras para verificar e validar o correto funcionamento. Desta forma as empresas garantem que não terão problemas quando colocarem o sistema real a funcionar. No ensino os simuladores mais desenvolvidos estão muito focados na criação de simulações de robôs ou braços robotizados, deixando espaço à criação de simulações que auxiliem os estudantes no controlo de sistemas mais simples.

Todo o controlo é realizado numa ferramenta, FEUPAutom, criada de raiz pelos os professores da UC para aplicação nesta. Começando a ser desenvolvida em 2006 no *Integrated Development Environment* (IDE) Delphi 7 (de 2002) foi durante muitos anos mantida nesta ambiente de desenvolvimento, o que torna o seu atual desenvolvimento lento, desatualizado, e impeditivo de grandes melhoramentos, devido às limitações e antiguidade do IDE usado.

## 1.2 Objetivos e Motivação

A dissertação realizada tem como principal objetivo a criação de um simulador que seja, ao mesmo tempo, um SCADA, permitindo assim aos alunos adquirir conhecimentos não só sobre os sistemas SCADA mas também sobre a importância da simulação antes de implementar os controladores projetados nos equipamentos reais.

Nesta dissertação é também apresentado o trabalho realizado na melhoria do SoftPLC FEUPAutom de forma a torna-lo mais intuitivo, garantindo que segue de perto a norma do GRAFCET a IEC 60484, são ainda apresentadas as melhorias da migração efetuada de Delphi para Lazarus.

Todo este projeto é muito desafiante, uma vez que existem enormes dificuldades inerentes à criação de um *software* genérico, que tenha como foco o ensino. Todo o trabalho desenvolvido irá permitir ajudar professores e estudantes a criar projetos mais complexos e atrativos, uma vez que existirá, desde logo, uma componente visual à qual poderão associar os controladores que forem criando. Por outro lado a atualização do FEUPAutom permitirá um desenvolvimento mais acelerado deste software com inúmeras vantagens para os estudantes.

Na Figura 1.1 é apresentado o conceito do sistema implementado. Tal como é observável o simulador criado (FEUPSim) é um programa que comunica com FEUPAutom através de entradas e saídas que permitem executar funções nos objetos do simulador, tornando-o interativo. Toda a comunicação entre os *softwares* é transparente para os utilizadores, através do protocolo Modbus.

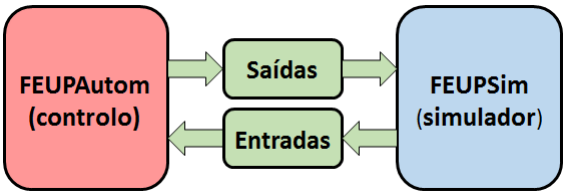



**Figura 1.1** – Conceito geral do sistema

## 1.3 Estrutura do Documento

Para além deste capítulo introdutório, este documento é composto por mais cinco capítulos.

No capítulo 2 é realizada uma descrição do estado da arte de vários componentes de interesse para a realização da dissertação, tal como os simuladores de automação, SCADAs, definição de SoftPLCs e uma descrição das normas aplicáveis.

No capítulo 3 são apresentados todos os procedimentos seguidos na projeção do simulador. No capítulo 4 é efetuada uma descrição da metodologia seguida e demonstradas todas as funcionalidades implementadas do simulador. No capítulo 5 é apresentado o simulador no seu estado final, alguns casos de estudo efetuados e discutidos os resultados obtidos.

No capítulo 6 são demonstradas todas as atualizações executadas ao FEUPAutom.

Por fim, no capítulo 7, são apresentadas as conclusões de todo o trabalho efetuado e é ainda referenciado algum trabalho futuro passível de ser executado no âmbito da presente dissertação.

# Capítulo 2

# Revisão Bibliográfica

Neste capítulo é apresentado o estado da arte de cada vertente de interesse no âmbito da dissertação a realizar. Para isso, são estudados os simuladores de automação mais influentes na área de ensino, os sistemas SCADA de referência para o trabalho a ser desenvolvido, alguns SoftPLCs que poderão ser integrados com a aplicação final e, por último, é realizada uma descrição das normas mais influentes na área da modelação/programação de controladores de sistemas de eventos discretos.

## 2.1 Simuladores de automação

Em automação industrial, a necessidade de simulação de todo o ambiente fabril tem vindo a crescer à medida que cada vez mais máquinas e robôs partilham o mesmo espaço que os seres humanos. Este crescimento, aliado ao facto da segurança imposta e das normas aplicáveis para o uso de robôs nestes locais serem muito restritas, leva a que a simulação se apresente como um ambiente sem riscos que permite recriar, de forma segura e sem danificar qualquer componente do robô ou máquina, todos os testes necessários. Com estes testes, é possível verificar todo o tipo de falhas que existam, expor erros e problemas que não foram considerados, servindo também para treinar utilizadores para as condições de ambiente real (simulação aeronáutica, braços robóticos, trajetos de robôs dentro de uma fábrica, entre outros). Mas nem tudo nestes sistemas de simulação é perfeito. Uma das maiores dificuldades consiste em simular de forma precisa o sistema em causa e, para além disso, por vezes o custo de aprendizagem da programação destes *softwares* é demasiado grande face aos benefícios referidos [1].

De seguida serão descritos 2 simuladores de automação importantes no contexto da preparação para a dissertação a ser realizada, uma vez que ambos são utilizados em ambiente educacional.

### 2.1.1 SimTwo

O SimTwo [2] é um ambiente de simulação gratuito desenvolvido na FEUP pelo Professor Doutor Paulo Costa e tem como grande foco o rápido desenvolvimento de simulações funcionais, tanto a nível educacional como de investigação. É possível simular robôs móveis, omnidirecionais ou não, manipuladores industriais, entre outros. Estes robôs possuem um elevado grau de realismo de movimentos, uma vez que a sua dinâmica é obtida através da decomposição dos robôs em corpos rígidos e motores elétricos, tal como referenciado em [3]. Toda a simulação é controlada através de um código desenvolvido pelo utilizador em linguagem Pascal.

Para conseguir manter uma interface simples, ser robusto e ao mesmo tempo ser muito versátil, este programa utiliza várias bibliotecas livres para facilitar o seu desenvolvimento. Dentro delas destacam-se a *GLScene* [4] que permite a virtualização de objetos em três dimensões, a biblioteca Open Dynamics Engine (ODE) que se encontra encarregue da simulação de corpos rígidos e a *Pascal Script*.

Na Figura 2.1 é apresentado o ambiente de desenvolvimento do SimTwo de acordo com a seguinte enumeração:

1. **SimTwo:** Janela de visualização da simulação.
2. **Code Editor:** Janela onde o utilizador escreve o seu código de controlo da simulação.
3. **Config:** Local onde se encontram todas as configurações do programa, tanto a nível de controlo, questões de simulação da física dos objetos, entradas e saídas entre outras.
4. **Chart:** Janela de evolução de variáveis ao longo do tempo.
5. **Sheets:** Uma folha de cálculo onde se pode criar elementos como botões, constantes ou variáveis a serem usados na simulação.
6. **XML Scene Edit:** Uma página de configuração dos objetos presentes na simulação.

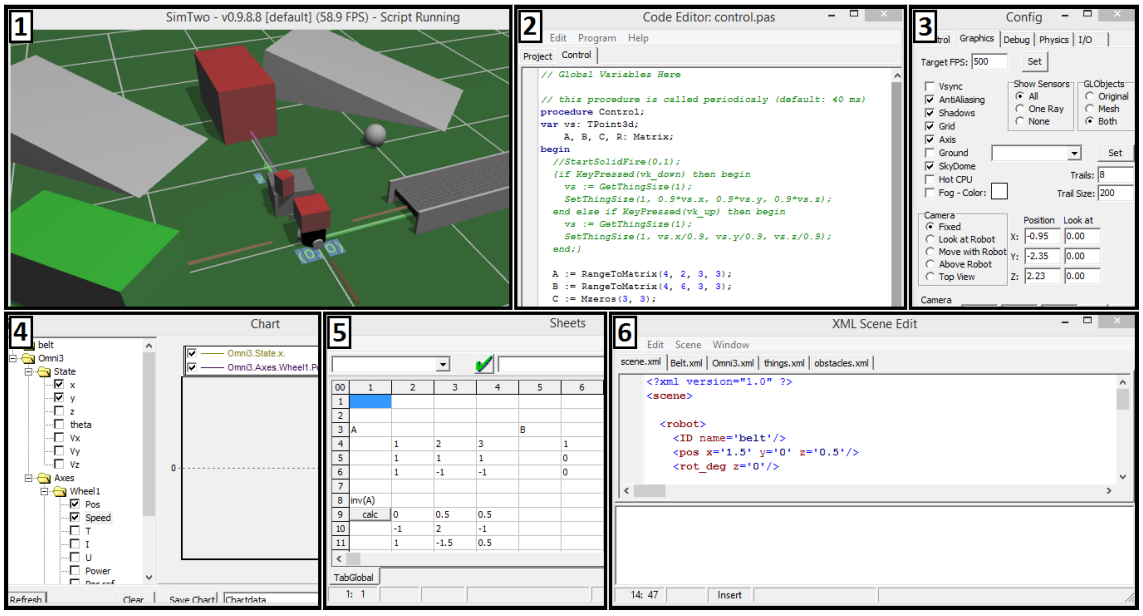



**Figura 2.1** – Ambiente de desenvolvimento do SimTwo

É ainda de salientar que este simulador foi construído para representar de forma realista todos os seus componentes, dotando-os de características lineares e não lineares. Estas tornam o seu comportamento muito mais próximo do real, o que se traduz, para o utilizador final, como uma mais-valia, uma vez que a passagem do código de simulação para a realidade é muito mais eficiente do que em simuladores que apenas usam características lineares.

Por último, é importante referir que este programa é usado em diversas Unidades Curriculares do curso de Eletrotécnica da FEUP, tendo obtido sempre críticas positivas por parte dos alunos, devido à sua facilidade de utilização e programação.

### 2.1.2 Factory I/O

O Factory I/O é um simulador, em tempo-real, de automação desenvolvido pela empresa portuguesa *Realgames* [5]. Este *software* é capaz de simular uma fábrica industrial e ligar-se com as tecnologias mais comuns de automação, tanto para treino como para uso educacional. A simulação inclui gráficos e física de objetos de alta qualidade, inspirada na indústria de automação e transformando assim toda a experiência muito mais realista [6].

O método de funcionamento faz uso de bibliotecas compostas por componentes que podem ser facilmente arrastados para a 'cena' a ser simulada e usar esses componentes em testes de tempo real.

Alguns dos componentes passíveis de criar com este simulador são [7]:

- Tapete de transporte de objetos;
- Alinhadores e *pushers*;
- Sensores: capacitivo, indutivo e ótico;
- Botões: emergência, início e paragem;
- Indicadores luminosos;
- Elevadores e braços de *'pick & place'*;
- Plataformas, escadas e proteções;

O fornecedor deste *software* garante que é possível criar e simular todos os componentes mencionados de forma rápida e eficaz, permitindo construir uma fábrica em pouco tempo e simular todo o seu funcionamento. Nessa simulação é possível selecionar opções tais como: o seguimento de um objeto específico, visualização em primeira pessoa, câmara lenta, entre outras [8].

Sendo o *Factory I/O* um programa de altíssima complexidade e permitindo aos utilizadores trabalharem com um número muito grande de cenários industriais, este utiliza um sistema de licenças pagas para ser usado. Recentemente os seus programadores lançaram um SDK (*Software Development Kit*) que permite aos utilizadores criarem interfaces entre as simulações e tecnologias externas.

### 2.1.3 Comparação

Além destes dois simuladores, foi ainda efetuado um estudo sobre alguns dos mais sofisticados *softwares* no âmbito do ensino na área de automação, tendo sido, para isso, realizada uma tabela comparativa (Tabela 2.1). Abaixo são explicadas as escolhas dos *softwares* para comparação, assim como os parâmetros usados para realização da mesma.

*Softwares* usadas para comparação:

**SimTwo** - Criado com foco na investigação e ensino, sendo por isso simples de criar e simular ambientes de forma rápida e eficiente.

**Factory I/O** - Permite a simulação de fábricas e componentes presentes nestas, de forma rápida e funcional, servindo assim para o treino e ensino do funcionamento destes sistemas.

**Gazebo** - Gratuito, permitindo simular múltiplos robôs em ambientes complexos.

**V-REP** - *Software* baseado numa arquitetura de controlo distribuída que permite que cada objeto seja controlado individualmente, o que facilita o controlo através de módulos.

**Ezphysics** - Facilidade de simulação e animação (parecido com vídeo jogos) com a possibilidade de integração com o *Matlab*, o que o torna muito útil em termos académicos.

Os critérios usados para comparação foram:

**Linguagem de programação** - Quanto maior o número de linguagens permitidas pelo *software* maior será, em principio, a facilidade de aprendizagem do utilizador, uma vez que poderá já estar familiarizado com alguma das linguagens apresentadas.

**Motor de dinâmica** - Este critério refere-se ao facto de o simulador poder ser portador, ou não, de uma maior dinâmica na simulação dos objetos e, por isso mesmo, trazer um maior realismo a todo o ambiente de simulação.

**Comunicação** - Possibilidade de comunicação com outros *softwares* ou *hardwares*.

**Sistema Operativo (SO)** - É um aspeto crucial, uma vez que o sistema operativo em que o simulador funciona terá que ser compatível com o equipamento do utilizador.

**Licença** - Sendo o custo essencial na escolha de um simulador para uso educacional, este fator foi utilizado para demonstrar o tipo de licenciamento usado por cada simulador.

**Tabela 2.1** – Comparação entre simuladores

| | **Linguagem de Programação** | **Motor de Dinâmica** | **Comunicação** | **Sistema Operativo** | **Licença** |
|---|---|---|---|---|---|
| **SimTwo** | Pascal | ODE | Porta série e UDP | Windows | Grátis |
| **Factory I/O** | Matlab LabView | N/A | TCP/IP | Windows | Subscrição |
| **Gazebo** | C, C++, Java | Bullet, ODE, DART | TPC/IP | Linux | Grátis |
| **V-Rep** | C, C++, Lua Java, Matlab | Bullet, ODE, Vortex | Porta série e TCP/IP | Multi (Portátil) | Pro (paga) Edu (grátis) |
| **Ezphysics** | C++, Matlab (simulink) | ODE | TCP/IP | Windows | Grátis e open-source |

## 2.2 SCADA

De forma a supervisionar, controlar e gerir sistemas complexos (fábricas, infraestruturas elétricas, aeroportos...) eficientemente, é normalmente usado um *software* centralizador que agrega toda a informação dispersa que compõe o sistema. Este *software* é denominado por *Supervisory Control And Data Acquisition* (SCADA). Tal como o seu nome indica, o foco principal é a supervisão e controlo do sistema de forma a melhor gerir todo o processo envolvente. Exemplos onde os SCADAs são usados: controlo de tráfego, indústria química, geração de energia, entre outros.

Um dos aspetos fulcrais de um SCADA é a sua capacidade de adquirir dados do sistema físico a que se encontra associado. Estes dados são disponibilizados através de uma interface ao utilizador e normalmente guardados em histórico para posterior análise, tanto a nível de problemas ocorridos, como de possíveis melhoramentos no processo. A interface representa a componente de supervisão através da qual é possível verificar todos os dados adquiridos e o correto funcionamento do sistema. É nesta que são despoletados avisos de avarias e alarmes de erros ou situações de emergência. Por fim, existe ainda a possibilidade de executar remotamente operações sobre o sistema implementado, acionando ou parando determinado componente [9, 10].

Os SCADAs são uma camada de *software* que se posiciona no topo de todo o hardware e inclui normalmente os seguintes subsistemas:

- Remote Terminal Unit (RTU) ou Programmable Logic Controller (PLC) que efetuam as trocas de dados através de I/O (entradas/saídas);
- Uma Interface Homem-Máquina (HMI) através da qual o operador controla o processo;
- Redes de comunicação que interligam os vários componentes ao sistema;
- Um histórico de todo o sistema;
- Um módulo de gestão de alarmes;

Os SCADAs têm uma forte relação com a interface a que estão associados pois é nesta que explicitam todas as mudanças de variáveis do sistema. Para além disso, é nesta que estão representados os processos que compõem o sistema através de sinópticos ilustrativos das suas funções. Nesta interface são também incluídos alguns campos que os operadores podem alterar de forma a melhorar o processo, aumentando assim a produtividade de todo o sistema.

Nas versões mais recentes destes *softwares*, tem vindo a ser adotada cada vez mais uma política orientada à *Internet*, onde os serviços na 'nuvem' conseguem reduzir drasticamente o custo de manutenção e proporcionar aos utilizadores, em tempo real, um maior número de informações.

Dada a importância dos SCADAs na supervisão e controlo automático, foi realizado um estudo sobre três *softwares* que permitem o desenvolvimento de SCADAs. A escolha foi realizada de acordo com três fatores: A recomendação de uma das maiores empresas a nível internacional no ramo de automação, a *Schneider* que recomenda o *CitectSCADA*; Um software utilizado a nível educacional e gratuito, o *Winlog*; E por último, como cada vez a *Internet* é um fator decisivo no crescimento de qualquer empresa, foi estudado o *Integraxor* que é completamente baseado em tecnologias *web*.

### 2.2.1 CitectSCADA

A *Citect* foi uma companhia comprada pela *Schneider Electric* em 2008 que era especializada na criação de *software* para a indústria de automação e controlo. O seu produto principal é o *CitectSCADA* que consiste num pacote de *software* líder de mercado na área dos SCADAs. Este teve a sua versão inicial em 1987 e desde então tem sofrido várias melhorias, por forma a manter-se sempre na vanguarda de toda a tecnologia neste setor [11].

A importância deste *software* para a dissertação a ser realizada prende-se no facto de este ser um dos mais avançados sistemas de criação de SCADAs do mundo e, por isso, um exemplo de boas práticas a seguir.

Algumas das funcionalidades mais importantes do *CitectSCADA* são:

- Arquitetura escalável e fiável permitindo ainda múltiplos servidores redundantes para uma maior segurança ao nível de falhas.
- Criação de gráficos, animações e interfaces com o utilizador de alta qualidade.
- Gestão de alarmes avançada, com vários tipos de alarmes (filtrados, atrasados, múltiplos).
- Gráficos de tendência e Histórico em tempo-real.
- Duas linguagens criadas de raiz para facilitar o uso do *software* e garantir que as aplicações criadas são flexíveis e usam sempre o máximo potencial: *Cicode* e *CitectSCADA VBA*.
- Além de incluir uma vasta gama de *drivers* (100+) permitindo ligar a vários controladores, possui um cliente OPC (*Open Platform Communications*), *standard* na indústria de conexão a equipamentos, e existe ainda uma *Application Program Interface* (API) para conexão com aplicações criadas pelos clientes.
- Uma extensa biblioteca de conhecimentos *online*, que acumula experiência, dúvidas e detalhes técnicos de vários utilizadores ao longo dos anos.

### 2.2.2 Winlog

O *Winlog* [12] é um pacote de software que permite a criação de aplicações SCADA da empresa italiana *Sielco Sitemi*. Este pacote é usado em contexto educacional, uma vez que possui uma vertente grátis denominada *Winlog lite* que, embora não contenha todas as funcionalidades da versão paga, permite aos alunos desenvolver pequenas aplicações funcionais e simples. Toda a ambientação e desenvolvimento de aplicações permite aos alunos adquirir conhecimentos fulcrais da importância dos SCADAs. Como esta versão é grátis, só permite o uso de 24 *tags* para comunicação e não inclui a biblioteca gráfica de símbolos usada na versão paga, limitando assim experiência do utilizador.

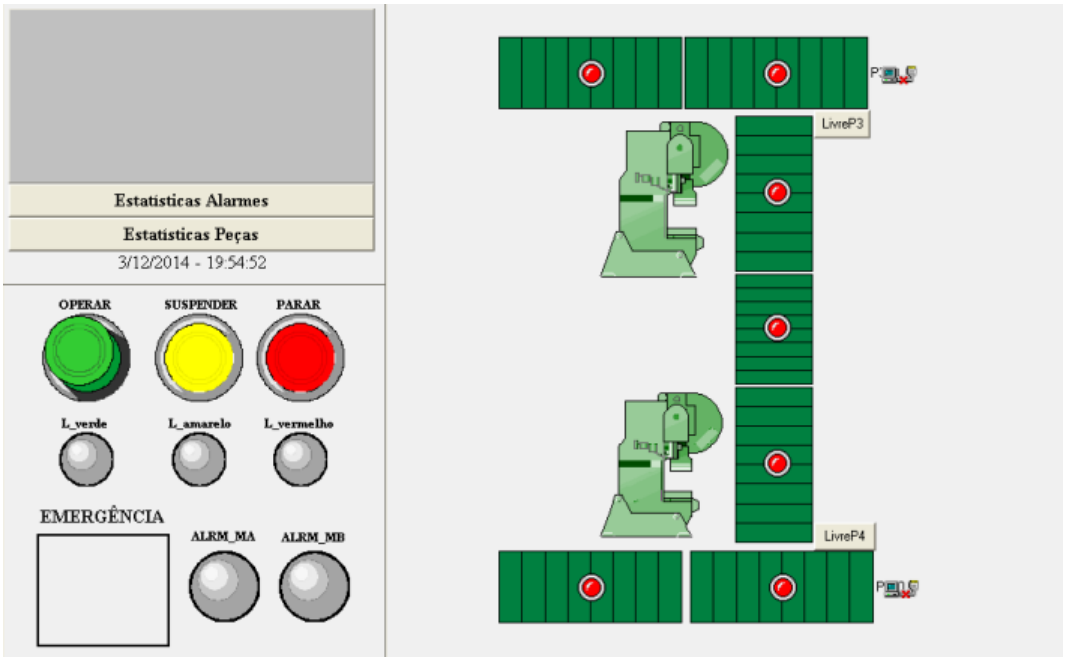



**Figura 2.2** – Representação de uma aplicação realizada em *Winlog Lite*

Na Figura 2.2 é possível visualizar uma representação de uma aplicação criada em ambiente educacional com o *Winlog*. Neste exemplo sobre a operação de peças através de duas máquinas é possível visualizar as três componentes que compõe este *software*:

**Supervisão** - O utilizador pode visualizar todo o processo e tem *LEDs* para verificar se existe uma situação de emergência ou alarme;

**Controlo** - É possível operar, suspender ou parar o processo;

**Aquisição de dados** - Através dos *LEDs* vermelhos que indicam a presença das peças;

### 2.2.3 Integraxor

A empresa *Ecava* desenvolveu o SCADA *Integraxor* [13] de raiz usando tecnologias *web* para permitir aos seus utilizadores criarem aplicações sofisticadas de sistemas Tempo-Real.

Algumas das vantagens deste *software* prendem-se com o facto de permitir ao utilizador saber a qualquer momento o estado do sistema que está a controlar, uma vez é possível visualizar a aplicação implementada através de qualquer navegador de computador ou telemóvel. Ao usar tecnologia *web* como o HTML e o *Javascript*, que se encontram estandardizadas é muito fácil encontrar informação sobre como programar tanto o lado do servidor como o cliente. Para além disso suporta ainda comunicação *Modbus* e *OPC*, funções de alarme, múltiplas base de dados e realização de relatórios.

## 2.3 SoftPLC

Hoje em dia, uma das decisões cruciais na industrialização dos processos de uma empresa consiste na escolha do sistema que controla a sua fábrica. Com a evolução cada vez mais acentuada dos computadores em termos de desempenho, diminuição de custo, aumento de fiabilidade e principalmente facilidade de integração e programação, o uso destes sistemas tem vindo a aumentar em relação ao uso único de PLCs.

Embora os PLC tenham sido desenvolvidos especificamente para substituir as caixas de interruptores e painéis de relés, melhorando a flexibilidade e fiabilidade destes mas mantendo a sua robustez e desempenho, o seu desenvolvimento tem vindo a diminuir face à constante evolução verificada no mundo dos computadores. Com esta evolução, cada vez mais os computadores têm vindo a ser criados com funcionalidades acrescidas, que até ao momento eram somente caraterísticas dos PLC. A adição de sistemas operativos capazes de trabalhar em tempo real também tem vindo a contribuir para esta mudança.

Complementarmente ao referido foram sendo criados *softwares* que transformam os computadores no que é conhecido como um soft-PLC. Estes ambientes de desenvolvimento além de permitirem programar os PLC, emulam o seu funcionamento e são capazes de controlar entradas/saídas de um processo real ou simulado a partir de um computador [14].

### 2.3.1 FEUPAutom

O FEUPAutom [15] é um softPLC gratuito concebido na FEUP pelo Professor Doutor Armando Sousa. Este teve a sua origem na necessidade da criação de um *software* que substituísse os dispendiosos PLC's necessários para o ensino da problemática do controlo de sistemas de eventos discretos na UC de "Sistemas e Automação" da FEUP.

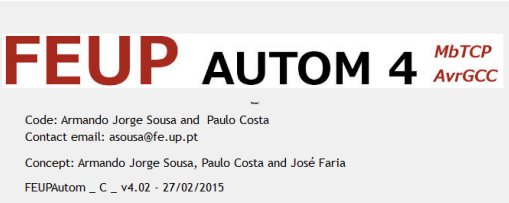



**Figura 2.3** – Logótipo do FEUPAutom

#### 2.3.1.1 Características e aplicações

As características principais deste SoftPLC são: a portabilidade, a forma intuitiva de programar e as suas capacidades de depuração. Desta forma pode ser usado no ensino e desenvolvimento de programas de controlo automático tanto no laboratório como em casa, permitindo assim aos estudantes terminar as aulas em casa e aprofundar os seus conhecimentos na área de automação, uma vez que existe uma grande abundância de exercícios que poderem ser resolvidos usando o

FEUPAutom como controlador principal.

Este *software* pode ser programado em Structured Text (ST) ou em Grafcet. Enquanto a primeira linguagem foi criada por forma a seguir de perto as definições da norma *International Electrotechnical Commissio*n (IEC) 61131-3, a segunda é uma adaptação da norma IEC 60848. Existe ainda suporte para comunicação *ModBus TCP*, o que se traduz numa enorme facilidade de envio e receção de dados por parte do FEUPAutom, permitindo assim ligá-lo com simulações externas.

Na Figura 2.4 é apresentado o ambiente de desenvolvimento do FEUPAutom de acordo com a seguinte enumeração:

1. **FEUPAutom:** Nesta janela é possível editar algumas definições de projeto, escrever programas em ST e ver a sua conversão para linguagem C e Pascal. É também aqui que são mostradas mensagens de erros do programa e/ou mensagens transmitidas por este.
2. **IO Leds:** Representação do estado das entradas e saídas do sistema. É ainda possível forçar determinado *bit* a *True* ou *False*.
3. **Grafcet Draw Grid:** Janela que permite a criação de programas em ST. Todo o desenho do Grafcet é realizado na grelha e existe uma area de programação onde o utilizador pode escrever as transições ou ações que devem ser executadas.
4. **Log:** Representação de traçados temporais de variáveis escolhidas pelo o utilizador.
5. **ST Editor:** Nesta janela que está contida dentro da 1ª apresentada é onde o utilizador escreve o seu código ST ou onde verifica o código ST gerado pelo o Grafcet.

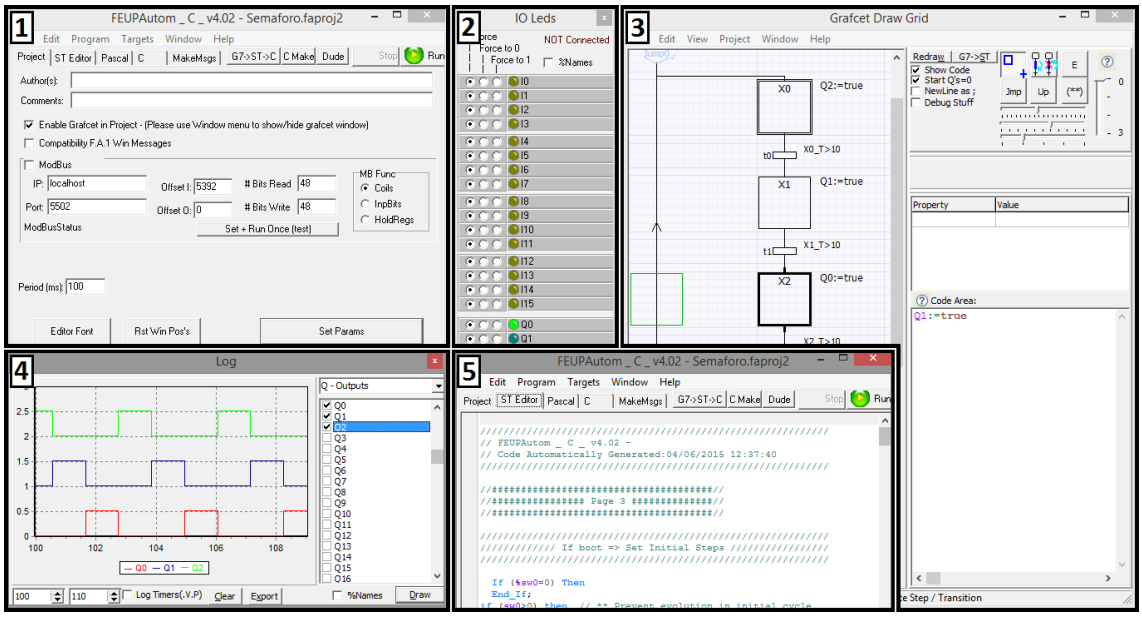



**Figura 2.4** – Ambiente de desenvolvimento do FEUPAutom

#### 2.3.1.2 Depuração

Uma vez que o FEUPAutom se encontra em constante desenvolvimento, quer seja por novas adições, novas exigências ou por sugestões feitas pelos alunos, este possui várias ferramentas de depuração para melhorar o seu uso:

- No modo Grafcet, o FEUPAutom pode converter o Grafcet realizado para uma máquina de estados, transformar o desenhado em código ST e dar uma tradução dos objetos desenhados para um formato de texto (tal como é indicado na Figura 2.5). Desta forma, o utilizador pode verificar se todos os objetos são corretamente criados e pela ordem em que este os fez. A área de escrita de código tem um sistema auto-complemento o que permite uma maior rapidez no desenvolvimento de algoritmos, existindo ainda a possibilidade de deixar comentários em cada objeto.
- Na janela principal, é possível assistir à conversão de Grafcet em ST, de ST em Pascal e linguagem C. Existe também uma janela para erros de compilação onde é apresentada a linha do erro em ST e, uma vez que são usados os mesmos nomes que o Grafcet para as variáveis, o utilizador pode rapidamente identificar onde se encontra o erro no Grafcet.
- A última ferramenta de depuração é o *Log* Temporal (o número 4 da Figura 2.4). Esta é crucial, uma vez que é perfeita para visualizar o estado das variáveis do sistema, à medida que o tempo avança.

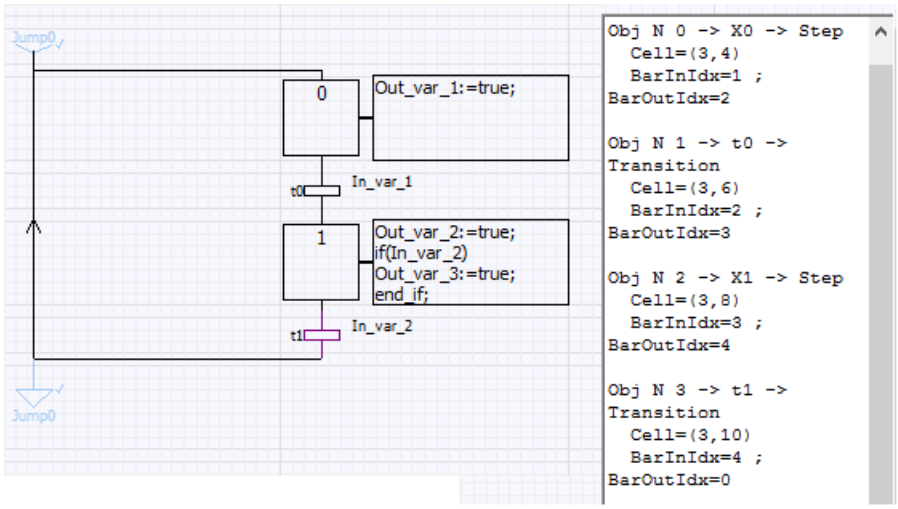



**Figura 2.5** – Ferramentas de depuração do FEUPAutom

#### 2.3.1.3 Simulações

Para se tornar numa ferramenta verdadeiramente inovador na área de ensino de controlo de sistemas de eventos discretos, houve uma necessidade de desenvolver alguns exemplos de trabalho para que os alunos pudessem testar os seus códigos. Para isso, o Professor Doutor Paulo Costa criou três simulações de sistemas reais que permitem aos alunos treinar, com exemplos práticos, as matérias lecionadas, ajudando-os assim a alcançar melhores resultados. Os três simuladores são mostrados na Figura 2.6 pela seguinte ordem:

1. **Parque 3D:** Com este simulador os alunos são incentivados a desenvolver o programa de controlo de um parque de estacionamento. Primeiramente só existem carros a entrar até um limite de 6. De seguida é introduzida uma segunda barreira na saída e o controlo tem que gerir também o números de carros a sair, tornando o exercício um pouco mais complexo.
2. **Portão 3D:** Neste simulador o objetivo é controlar a abertura e fecho de um portão de garagem e garantir que, se intensidade da luz natural for inferior a um determinado nível, a lâmpada acende.
3. **Thunderbird 3D:** O último simulador apresentado é o sistema de luzes traseiras de um carro, no qual os alunos têm que implementar o controlo de três funções: virar à direita, virar à esquerda e luzes de emergência.

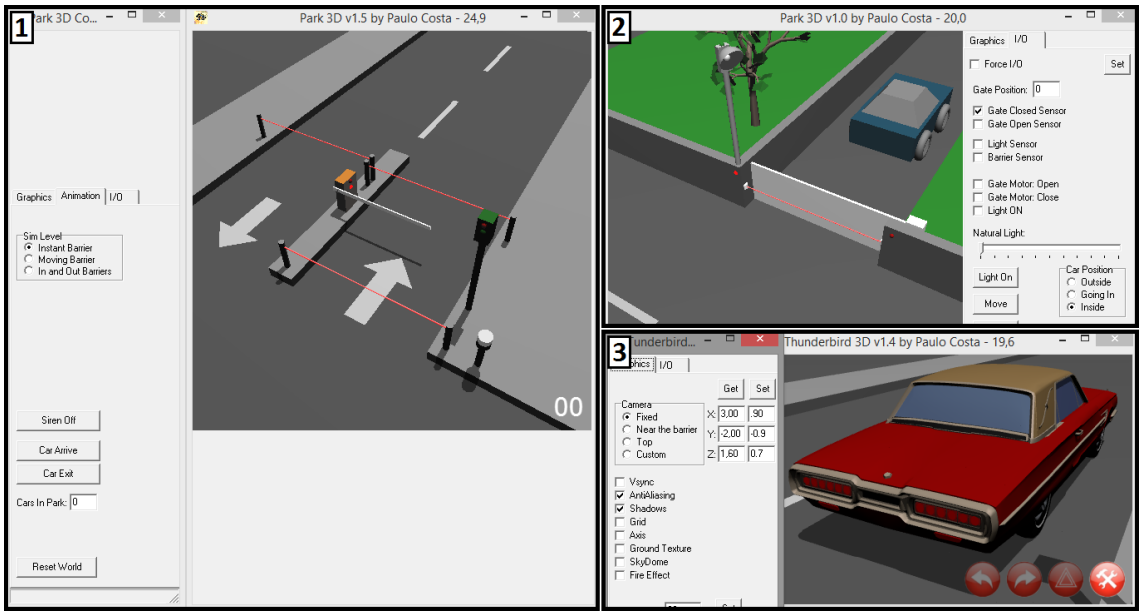



**Figura 2.6** – Simulações associadas ao FEUPAutom

### 2.3.2 Beremiz

O Beremiz [16] é um ambiente de desenvolvimento integrado (IDE) grátis e multi-plataforma que permite a criação de programas de automação e que respeita totalmente a norma IEC 61131-3. O grande foco foi desenvolver uma aplicação que fosse independente do vendedor e/ou da máquina podendo tornar assim qualquer processador num PLC. Este ambiente é composto por 4 subprojectos:

**CanFestival** - Providência uma biblioteca de E/S e uma interface com a framework de CANOpen;

**SVGUI** - Ferramenta de criação de interfaces gráficas e interfaces homem-máquina (HMI);

**The MatPLC's IEC compiler** - subprojecto começado em 2002 pelo Professor Doutor Mário Sousa da FEUP e que pretende compilar código das linguagens IL/SFC/ST para código ANSI-C;

**The PLCOpen Editor** - Edição de programas nas 5 linguagens presentes na norma IEC 61131-3: FBD, IL, LD, SFC, ST;

### 2.3.3 ISaGRAF

O ISaGRaf [17] é um *software* industrial desenvolvido pela ICS Triplex ISaGRAF que é usado para controlo de variados produtos, incluindo micro-controladores, PLC's, PAC's entre outros.

Além de possuir as cinco linguagens de programação da norma IEC 61131-3, respeita na totalidade essa mesma norma e a norma IEC 61499. Além de ser possível programar os controladores, o *software* possui um sistema online de depuração, uma ferramenta de simulação, de gestão de projetos e ainda é possível criar uma HMI para visualização de variáveis e componentes.

Dado possuir tantas funcionalidades, a curva de aprendizagem deste *software* é muito extensa. Devido a isto e ao facto de possuir um sistema de licenciamento bastante caro normalmente, não é usado por pequenas empresas. No entanto é inegável que atualmente é uma das ferramentas de programação industrial mais completas e fiáveis do mercado.

### 2.3.4 Sysmac Studio

O Sysmac Studio é um *software* criado pela empresa *Omron* [18] que concentra em si todas as funcionalidades de controlo de máquinas num ambiente único e integrado. Neste é possível realizar a programação, a configuração, a monitorização e a simulação de todas as máquinas já preparadas para a sua utilização.

Sendo focado na indústria de automação, este *software* usa programação baseada em eventos, sendo completamente compatível com a norma IEC 61131-3. Fazendo uso de tecnologias muito recentes possui formas de deteção de erros automáticos, usando lista de variáveis e seguranças internas pré-definidas. É ainda de referir a possibilidade de simulação de vários componentes e visualização de dados através de gráficos em tempo real.

A empresa *Omron* vende ainda equipamentos que são completamente integrados com o *software* em questão, que têm a vantagem de possuir ligações *EtherCat*, tornando todo o sistema extraordinariamente rápido.

## 2.4 Estudo das normas aplicáveis

Neste capítulo serão são realizadas algumas considerações sobre as normas de interesse no âmbito desta Dissertação. Primeiramente é realizado um estudo sobre a norma IEC 60848 e de seguida uma apresentação sobre as linguagens de programação definidas pela norma IEC 61131-3 e realizada uma comparação entre o *Graphe Fonctionnel de Commande, Étapes Transitions* (GRAFCET) e o *Sequential Function Chart* (SFC).

### 2.4.1 IEC 60848 – Grafcet

O GRAFCET é uma poderosa linguagem gráfica de modelação para o controlo de sistemas de eventos discretos e é por isso essencial no ensino de uma UC introdutória de automação. Todos os conceitos base adquiridos com a introdução desta linguagem são essenciais à aprendizagem de qualquer engenheiro de automação. Uma vez que, com a construção do Grafcet através da especificação do sistema, é possível construir um modelo que será facilmente transformado num programa destinado a realizar o controlo automático de um processo. De forma a homogeneizar os diferentes métodos de modelação de sistemas de eventos discretos existentes através de Grafcets proprietários foi definida, em 1988, a norma IEC 60848 [19], que define todos os símbolos e regras de representação da linguagem, bem como estes devem ser interpretados.

De seguida é realizada uma pequena introdução ao Grafcet que é especificado na norma de forma a ser possível no subcapítulo seguinte realizar uma comparação com o FEUPAutom.

Tal como o seu nome indica, o Grafcet é composto por dois elementos fulcrais: as etapas e as transições. A alternância entre estes dois elementos tem sempre de existir por forma a garantir a correta evolução do sistema. Além da breve apresentação destes dois elementos, será também explicada a noção de ação num Grafcet. Por último, será feita uma reflexão sobre a estruturação da modelação e das formas de controlo hierárquico disponíveis. É importante referir que o Grafcet é uma linguagem estruturante, precisando de ser complementado por linguagem ST ou outra aquando da atribuição de valores a variáveis, o que é diferente das linguagens de programação procedimentais como o C, onde o utilizador realiza um programa que executa todo o controlo.

#### 2.4.1.1 Etapas

Uma etapa define o estado da parte sequencial do sistema, existindo 6 etapas passíveis de representar, tal como indicado na Figura 2.7.


**Figura 2.7** – Representação das etapas (adaptado de [19])

Descrição dos elementos presentes na Figura 2.7:

1 e 3) Uma etapa é designada por um identificador alfanumérico e tem sempre uma variável booleana associada que representa o seu estado. Quando se encontra ativa (representado em 3 da Figura 2.7) a variável associada X* terá o valor '1', se a mesma etapa estiver desativa a variável conterá o valor '0'.

2) Num Grafcet existe sempre pelo menos uma etapa inicial que se encontra ativa aquando do arranque do sistema.

4 e 5) Uma etapa encapsulante consiste numa etapa que tem em si outras etapas incluídas. Tal como numa etapa normal poderão existir etapas encapsulantes iniciais.

6) Por último existem ainda macro etapas que são representações de partes detalhadas de um determinado Grafcet.

#### 2.4.1.2 Transição

Uma transição é um elemento que indica uma possível evolução entre uma ou mais etapas, existindo 4 formas de a representar, tal como é indicado na Figura 2.8:

1) Transição dependente de uma condição.

2 e 3) Transição dependente da mudança da variável '*' no flanco ascendente (0 -> 1) e descendente (1 -> 0), respetivamente.

4) Transição temporalmente dependente, isto é, a condição só é verdadeira passado um tempo t1 depois da variável '*' mudar de estado de 0 para 1 e passa a falsa se passar um tempo t2 após a variável ter mudado do estado 1 para 0.

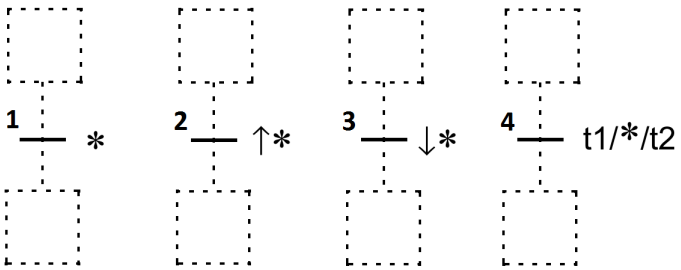



**Figura 2.8** – Representação das transições (adaptado de [19])

#### 2.4.1.3 Ações

Associada a uma etapa poderá existir uma ação que pode estar ou não ligada a uma ou mais saídas do sistema. Existem dois tipos de ações: as contínuas e as memorizadas, tal como ilustrado na Figura 2.9.

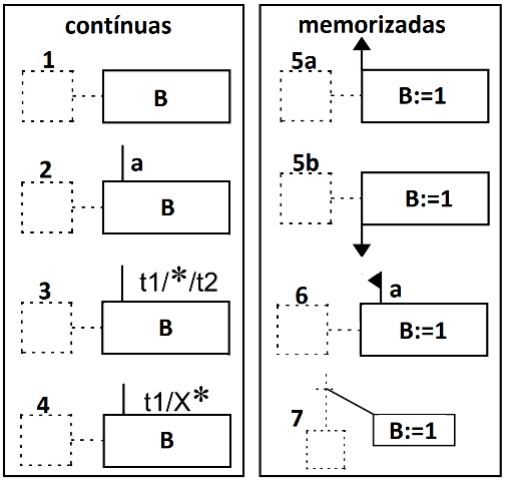



**Figura 2.9** – Representação das ações (adaptado de [19])

Descrição dos elementos presentes na Figura 2.9:

**Ações contínuas**, apenas são executadas enquanto a etapa associada está ativa:

1) Ação é executada enquanto a etapa está ativa.

2) Ação condicionada: É executada enquanto a etapa está ativa e a condição 'a' tem o valor verdadeiro, isto é 'a' é igual a 'true'.

3) Ação temporalmente dependente: É executada enquanto a etapa está ativa e a condição t1<*<t2, isto é, a variável temporal '*' tem de ser maior que o tempo t1 e menor que o tempo t2.

4) Ação atrasada no tempo: É executada passado o tempo t1 depois da etapa ser ativada. Esta ação tem uma dual designada por ação limitada no tempo, que só é executada enquanto ainda não passou o tempo t1 desde da ativação da etapa. Esta é representada de igual forma mas a condição é negada (um risco por cima de t1/X*).

**Ações memorizadas**, são despoletadas por eventos internos e o valor atribuído às saídas é memorizado.

5a e 5b) Em cada um dos casos, a variável B assume o valor 1 na ativação da etapa e na desativação da etapa, respetivamente.

6) A ação é executada quando o evento 'a' é despoletado. Existem três tipos de eventos: ativação de uma etapa (act(etapa)), desativação de uma etapa (dac(etapa)) e disparo de uma transição (clr(transição)).

7) A ação é executada aquando do disparo da transição.

#### 2.4.1.4 Macro-Etapas

De forma a melhorar e a simplificar a compreensão de um Grafcet, é possível representar partes destes através de macro-etapas, diminuindo assim a quantidade de etapas-transições presentes, o que facilita a interpretação do mesmo. Desta forma, o utilizador pode ter uma representação mais superficial do sistema, permitindo que se faça uma descrição do geral para o particular.

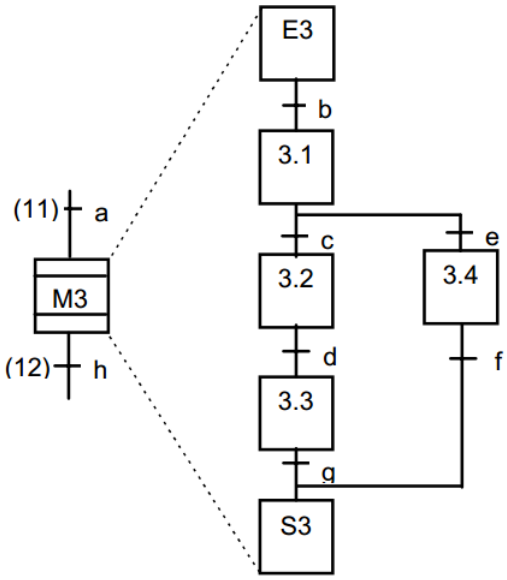



**Figura 2.10** – Representação de uma Macro-Etapa (adaptado de [19])

Na Figura 2.9 está representado um exemplo de uma macro-etapa. É possível verificar que a macro-etapa M3 possui uma etapa de entrada E3 que é ativada após a transição (11) e uma etapa de saída S3.

#### 2.4.1.5 Hierarquia

A hierarquia é uma aspeto muito importante na construção de sistemas de controlo complexos uma vez que permite criar situações de emergência, controlo por níveis entre outras funcionalidades que são essências neste tipo de sistemas. Existem duas formas de realizar controlo hierárquico em Grafcet: por encapsulamento de etapas ou por forçagem de estados, normalmente chamadas de macro-ações.

**Encapsulamento (Enclosure)**

Com o encapsulamento é possível realizar hierarquia através de etapas que contêm Grafcets parciais, implicando que as etapas encapsulantes são superiores hierarquicamente e as etapas encapsuladas dependem destas.

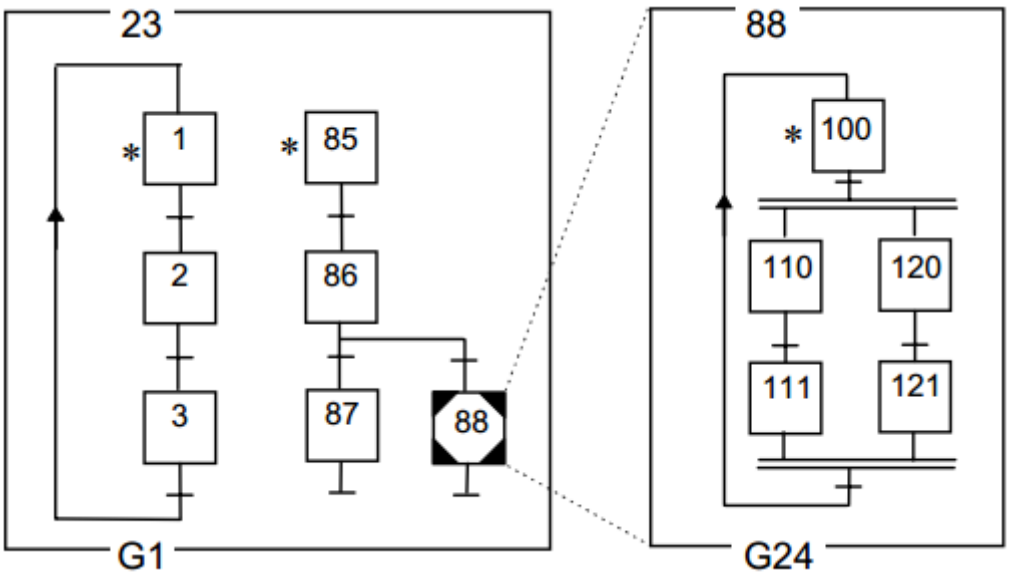



**Figura 2.11** – Representação do conceito de encapsulamento (adaptado de [19])

Na Figura 2.11 é possível observar um exemplo de uma etapa encapsulante (88) que, quando é ativada, implica a consecutiva ativação do Grafcet parcial G24. Se a etapa 88 de G1 for desativada, todas as etapas encapsuladas e Grafcets parciais a ela associados serão desativados.

**Macro-Ações**

As macro-ações ou forçagens implementam o controlo hierárquico através de certas funções especiais que dão a possibilidade de um Grafcet de nível superior controlar um conjunto de Grafcets de nível inferior. Um exemplo comum é o controlo de uma fábrica, onde um Grafcet de nível superior controla o funcionamento de uma célula e um conjunto de Grafcets hierarquicamente mais baixos comandam subsistemas dessa célula (máquinas, tapetes, ...).

Existem então 4 macro-ações:

**1) Imposição de uma situação**: Representado por "G3{8,9,11}", esta macro-ação implica que quando a etapa a que ela se encontra associada for ativada, o Grafcet parcial 3 é forçado à situação de ter as etapas 8, 9, 11 ativas e todas as outras desativadas.

**2) Congelar/Bloquear a situação**: Representada por "G3{*}", esta macro-ação força a que a situação atual do Grafcet parcial 3 se mantenha, isto é bloqueia todas as transições.

**3) Inicializar um Grafcet**: Representada por "G3{INIT}", esta macro-ação força que o Grafcet parcial 3 seja inicializado, mesmo que já se encontrasse a meio da sua execução.

**4) Imposição de situação vazia**: Representado por "G3{ }", com esta macro-ação é possível impor uma situação em que todas as etapas são desligadas.

### 2.4.2 IEC 61131-3

Nesta sub-secção é realizada uma breve introdução à norma IEC 61131-3 [20] fulcral no desenvolvimento de controladores em ambiente industrial, uniformizando as diferentes linguagens utilizadas para programar os diversos PLCs de cada fabricante.

Como referido, a norma IEC 61131-3 surgiu para unificar quais as linguagens de programação que os diferentes fabricantes deveriam permitir em que os seus PLCs fossem programados. Desta forma, foi possível um maior investimento de tempo na aprendizagem das 5 linguagens definidas pela norma, que seria recuperado, uma vez que todos os PLCs poderiam ser programados da mesma forma. Além de serem definidas as linguagens, são também apresentados conceitos sobre tipos de dados, estruturação, organização dos programas e configurações.

No âmbito desta preparação para a Dissertação, apenas serão focadas as linguagens de programação definidas pela norma, com especial atenção à linguagem gráfica SFC [21].

Esta norma possui então cinco linguagens de programação, divididas em duas categorias: gráficas e textuais.

- Gráficas:
  - Function Block Diagram (FBD);
  - Ladder Diagram (LD);
  - Sequential Function Chart (SFC);
- Textuais:
  - Instruction List (IL);
  - Structured Text (ST);

### 2.4.3 Grafcet vs SFC

Foi realizado um estudo comparativo com o Grafcet sobre linguagem SFC, uma vez que esta é uma adaptação da norma IEC 60848 estudada em 2.4.1.

Primeiramente, é necessário referir que o objetivo de ambas as linguagens é diferente. Enquanto o SFC é projetado para ser instalado num controlador, o Grafcet definido na norma é apenas uma representação que exprime o comportamento que um controlador deve executar.

Enquanto no Grafcet da norma, quando existe uma situação de transição divergente do tipo "ou", é da responsabilidade do *designer* garantir que as transições são exclusivas, no SFC todas as transições são exclusivas, uma vez que existe a possibilidade de definir prioridades entre as transições divergentes.

No SFC uma ação pode ser escrita em qualquer uma das 5 linguagens da norma IEC 61131-3 e cada ação é executada num determinado momento pré-definido pelo o utilizador, isto é, pode ser executada de forma impulsional quando a etapa é ativada (P1) ou quando é desativada (P0), de modo contínuo (N) ou memorizado (S), temporalmente limitado (L) ou atrasado (D), entre outros.

Por último, é importante mencionar o facto de o Grafcet evoluir devido às mudanças de variáveis de entrada enquanto a evolução do SFC é realizada pelo o ciclo de leituras de entradas onde este modelo é executado. [22]

## 2.5 Análise crítica do FEUPAutom

### 2.5.1 Análise do Grafcet do FEUPAutom face à norma IEC 60848

A ferramenta FEUPAutom usa uma versão própria do Grafcet que, embora próxima da norma IEC 60848, não a cumpre na totalidade, criando uma versão diferente que os alunos são forçados a conhecer aquando da sua utilização. De forma a simplificar a aprendizagem e aproximar o Grafcet usado no FEUPAutom do Grafcet da norma, foi realizado um estudo de comparação entre ambas as versões e um levantamento das melhorias que poderiam ser realizadas no FEUPAutom.

#### 2.5.1.1 Exclusividade entre ações contínuas e memorizadas

**Contexto**

Como explicado em 2.4.1.3 uma ação pode ser do tipo contínuo, ou seja, só está ativa enquanto a etapa está ativa, ou do tipo memorizado, quando o valor atribuído fica guardado em memória. No FEUPAutom, embora exista a possibilidade de realizar ações de ambas as formas, essa opção é mutuamente exclusiva, isto é, ora se pode ter ações contínuas e não memorizadas ou o inverso.

**Mudança Sugerida**

Uma possível resolução para aproximar o FEUPAutom da norma seria para cada ação existirem duas áreas de código: uma em que se realizariam ações do tipo contínuo e outra onde se realizariam ações do tipo memorizado. Desta forma, seria também possível implementar eficientemente as ações memorizadas, usando uma ferramenta visual onde os alunos apenas escolheriam quando a ação seria memorizada, na ativação de uma etapa (RE), na desativação de uma etapa (FE) ou na eventualidade de um evento (C), tal como é ilustrado na Figura 2.12.

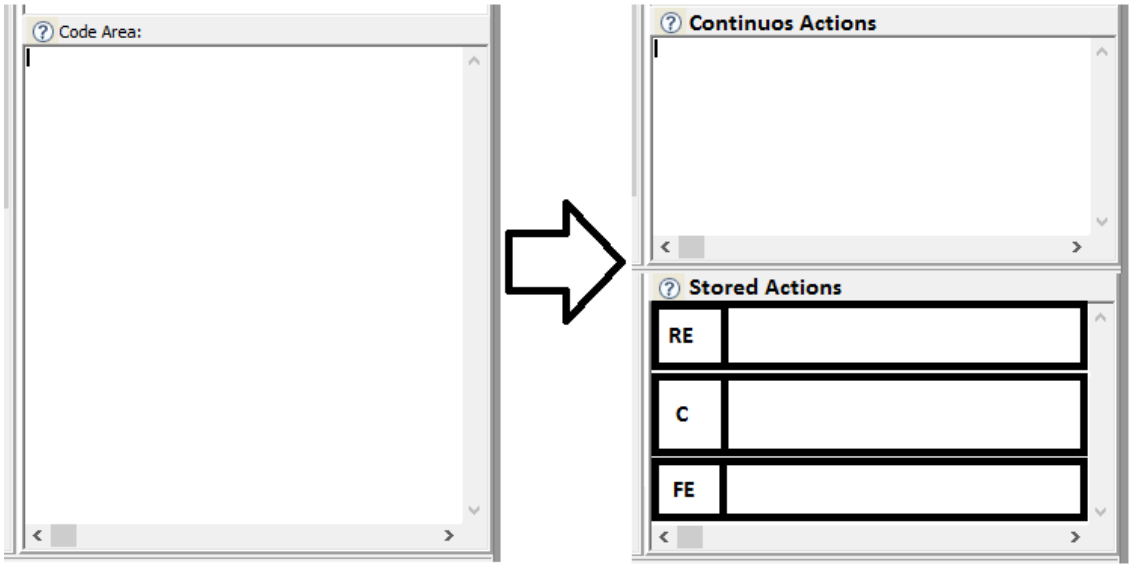



**Figura 2.12** – Integração de ações continuas e memorizadas no FEUPAutom

#### 2.5.1.2 Variáveis temporais associadas às etapas

**Contexto**

Segundo a norma IEC 60848, uma ação ou uma transição temporalmente dependente deve ser realizada no formato 't1/*/t2', tal como foi descrito em 2.4.1.3 e 2.4.1.2. No entanto, no FEUPAutom não é utilizada esta notação em nenhum dos casos, recorrendo-se em alternativa a uma variável extra que realiza a contagem do tempo da etapa e à utilização da linguagem ST para proceder ao controlo temporal.

**Mudança Sugerida**

A sugestão para a realização de uma aproximação à norma seria, neste caso, seria criar uma exceção na compilação de código e realizar uma conversão da notação da norma para código ST.

Embora esta sugestão vá de encontro ao método já implementado, sendo inglório para o programador ter que realizar trabalho extra, é necessário referir que este estudo tem como objetivo deixar o Grafcet do FEUPAutom o mais próximo possível da norma IEC 60848, para o lado de quem realiza o projeto.

#### 2.5.1.3 Garantia de transições mutuamente exclusivas

**Contexto**

Segundo o ponto 6.1.3 da norma, quando existe uma divergência de transições do tipo "ou", tal como representado na Figura 2.13, é da responsabilidade do *designer* garantir que as transições são, quer temporalmente, logicamente ou mecanicamente mutuamente exclusivas. Com alguns testes efetuados ao FEUPAutom, foi possível verificar que esta funcionalidade não foi implementada e que o Grafcet avança para ambas as etapas.

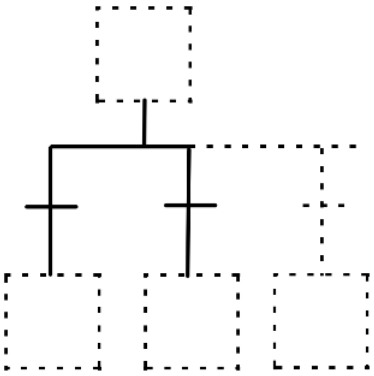


**Figura 2.13** – Divergência do tipo "ou" (adaptado de [19])

**Mudança Sugerida**

A primeira sugestão - e mais simples - é alertar o utilizador deste facto sempre que são criadas divergências do tipo "ou". A segunda sugestão, que só funciona se estivermos perante uma situação de bifurcação, isto é, quando apenas estão presentes duas transições, é verificar se as condições das transições são equivalentes. Embora esta solução não abranja todos os casos, já seria uma melhoria face ao implementado.

#### 2.5.1.4 Representação de ações

**Contexto**

Segundo a norma, quando existe mais de uma ação a ser executada por etapa, cada ação deverá possuir a sua caixa separadamente, tal como ilustrado na Figura 2.14. No FEUPAutom, de momento, não existe qualquer aproximação a nenhuma destas representações, uma vez que esta funcionalidade está implementada através do editor de código, podendo o utilizador nesta caixa de texto executar múltiplas ações relativas à etapa a que está associada.

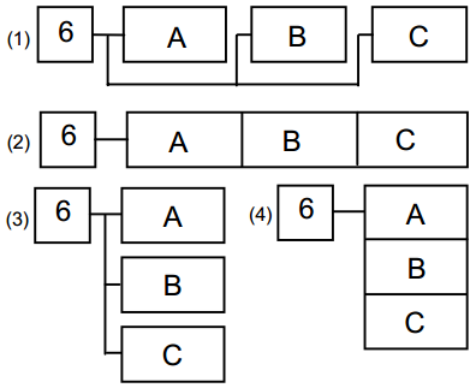



**Figura 2.14** – Múltiplas ações segundo a norma IEC 60848 (adaptado de [19])

**Mudança Sugerida**

A opção mais lógica seria utilizar a representação (2) da Figura 2.14, uma vez que a representação (1) ocupa muito mais espaço e, dado que o FEUPAutom utiliza um sistema de grelhas etapa-transição, o uso das representações (3) e (4) seria mais moroso e complexo de programar, sem que existisse essa necessidade.

#### 2.5.1.5 Representação de Macro-Etapas

**Contexto**

As macro-etapas, tal como explicado em 2.4.1.4, são uma forma de estruturação do Grafcet, permitindo um maior nível de abstração. Esta funcionalidade não se encontra implementada no FEUPAutom.

**Mudança Sugerida**

Uma possibilidade consiste na criação de um novo tipo de bloco chamado de macro-etapa. Este, ao ser duplamente clicado, abre uma nova janela onde é possível definir um conjunto de etapas-transições que ligarão com o Grafcet original a partir de duas etapas especiais que já estariam implementadas, uma de entrada e uma de saída.

#### 2.5.1.6 Controlo hierárquico

**Contexto**

No Grafcet existem duas formas para realizar o controlo hierárquico: por forçagens (macro-ações) ou por encapsulamento, tal como explicado em 2.4.1.5.

No entanto, no FEUPAutom não é utilizado nenhum destes dois métodos. Em contrapartida, existem duas opções distintas para realizar o controlo hierárquico. Um deles consiste na execução de código presente numa ação de etapas especiais que são denominadas "Zones" e que estão distribuídas no momento da compilação pelas várias fases de execução do ciclo do SoftPLC (No cálculo das transições, na desativação das etapas, na execução das ações das etapas ativas, etc...). O outro método recorre ao facto de existirem múltiplas folhas onde se podem modelar diferentes Grafcets para garantir que a última folha a ser compilada é a de número 0, implicando que as decisões que serão efetuadas nesta possam bloquear, inicializar ou até desligar saídas que teriam sido modificadas por folhas de número superior.

**Mudança Sugerida**

A implementação sugerida, e que vai de encontro com o lecionado na Unidade Curricular para o outro *software* utilizado - PL7Junior -, seria então a implementação das forçagens (macro-ações). Estas poderiam ser implementadas recorrendo a uma definição de Grafcets parcias que seriam definidos pelas diferentes folhas onde fossem desenhados. Isto levaria a que se pudesse desenhar somente um Grafcet "controlado"por folha, de forma a que, quando se usasse uma das 4 macro-ações enunciadas em 2.4.1.5, o compilador soubesse imediatamente qual o Grafcet em que deveria atuar.

### 2.5.2 Sugestões de melhorias para o FEUPAutom

Ao longo do estudo e dos testes efetuados para realizar a comparação entre o Grafcet usado pelo FEUPAutom e o recomendado pela norma, foram encontradas algumas possíveis melhorias que tornariam o sistema bastante mais *user-friendly*:

- Quando se clicar num objeto no modo de edição de Grafcet poder-se apagar com a tecla 'DEL' a etapa ou transição selecionada
- Os erros de Grafcet aparecem na janela de edição de Grafcet.
- Existir a possibilidade de mover vários objetos de uma só vez na edição de Grafcet.
- Ser possível visualizar todas as janelas no ambiente de edição de Grafcet, uma vez que neste momento só é possível visualizar a janela de ST.
- Unificar a gravação de projeto num só ficheiro em vez dos dois ficheiros com extensões diferentes que poderão confundir o utilizador.
- Atualizar a interface.
- Criação de um manual completo com todas as funcionalidades do FEUPAutom.

# Capítulo 3

# Projeto do Simulador

Como em qualquer projeto de engenharia, antes de se implementar uma solução é necessário projetar e formular o sistema que se idealizou. Serve o presente capítulo para apresentar os principais fatores que levaram à idealização desta dissertação, o levantamento de requisitos do simulador, o primeiro conceito do sistema a implementado criado e outras decisões relevantes de projeto.

## 3.1 Dados de partida

Como explicado na secção 1.1, na UC de Sistemas e Automação existe ainda um grande fosso entre os controladores que os estudantes criam e os simuladores existentes para estes poderem visualizar o resultado desse mesmo controlo. Isto é mais explícito quando são realizados inquéritos semestrais aos estudantes numa fase final da UC. Parte dos resultados das últimas duas edições destes inquéritos encontra-se ilustrada na Figura 3.1.

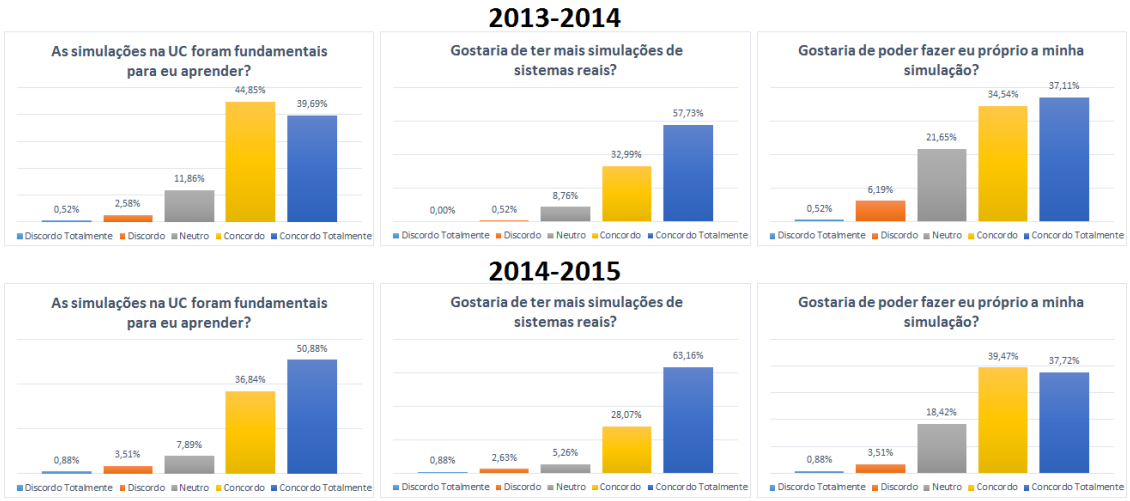



**Figura 3.1** – Inquéritos sobre a importância da simulação realizados na UC de Sistemas e Automação

Tal como pode ser observado, em ambas as edições:

- Cerca de 86% dos alunos concorda que as simulações são fundamentais;
- Quando é perguntado se gostariam de ter mais simulações reais, a percentagem de alunos que concorda sobe para os 91%, sendo que 2/3 destes 'concordam totalmente';
- Por último, é realizada a pergunta se os alunos gostariam de poder realizar eles mesmos a sua simulação, à qual cerca de 75% responde afirmativamente;
- De referir que as respostas negativas (Discordo ou Discordo Totalmente) a qualquer uma das perguntas têm valor médio de 1.47%;

As respostas a estas perguntas estão no génesis desta dissertação uma vez que, com valores tão elevados de estudantes a concordar que as simulações são uma parte fulcral da UC e que gostariam de poder desenvolver eles mesmos a sua simulação, foi então proposto que se criasse o simulador que será de seguida apresentado.

## 3.2 Análise de Requisitos

A fase inicial mais importante de qualquer projeto de alguma complexidade consiste no levantamento de requisitos que o sistema deverá cumprir. Este é normalmente realizado com o cliente e tem como objetivo enumerar todas as funcionalidades que o sistema deve cumprir, quer por imposições/restrições do cliente, quer por sugestões do projetista. O levantamento é realizado por forma a que no final exista um produto de alta qualidade e que vai de encontro ao que o cliente deseja. Por estas razões, a primeira fase de projeção foi o levantamento de requisitos em conjunto com o Professor Doutor Armando Sousa sobre como deveria ser o Simulador/SCADA:

- Requisitos Funcionais:
    1. O projeto deverá ser realizado por forma a ser passível de integrar com o FEUPAutom no futuro;
    2. Possibilidade de guardar o desenho da simulação, bem como as funções num ficheiro;
    3. Poder carregar ficheiros guardados;
    4. O ficheiro tem de ser executável;
    5. Deve ser possível criar vários tipos de objetos:
        - Botões;
        - Indicadores Luminosos;
        - Formas Geométricas;
        - Imagens;
        - Sensores;
- Requisitos não Funcionais:
    1. Ter uma interface simples e apelativa;
    2. O uso da aplicação deverá ser intuitivo;
    3. A aplicação deverá ser versátil de forma a permitir construir várias simulações diferentes;
    4. Deverá existir um manual de instruções;

## 3.3 Conceito do sistema

Depois de realizado todo o projeto do simulador e decididas as tecnologias a usar, foi criado um conceito do sistema a ser implementado. Tal conceito teve o propósito de apresentar ao cliente, o Professor Armando Sousa, o sistema projetado, por forma a verificar se este ia de encontro ao sistema idealizado e pensado pelo o mesmo. Tal conceito encontra-se representado na Figura 3.2, sendo de notar que, o que se quis aprovar aquando da sua realização foi:

- Existência de dois modos de execução: Simulação e Edição;
- Vários objetos possíveis de simular;
- Funções serem muito simplificadas;
- Interface simples e limpa;

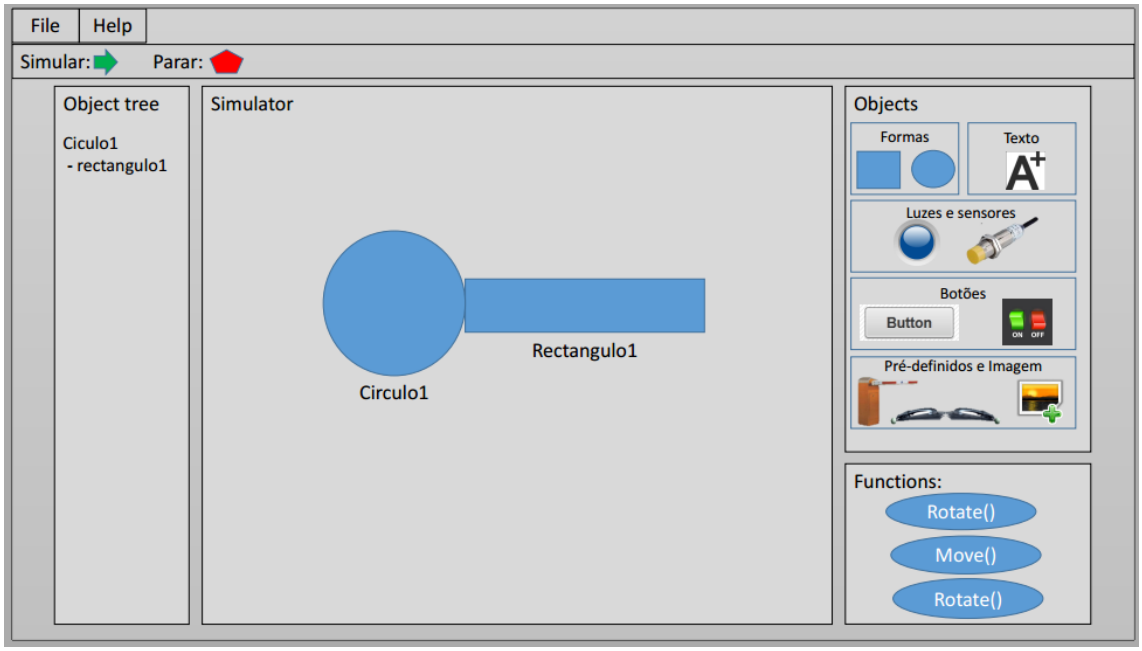



**Figura 3.2** – Conceito inicial do sistema a implementar

## 3.4 Outras decisões de projeto

### 3.4.1 Linguagem

Num esforço para tornar o simulador verdadeiramente universal a linguagem usada para toda a interface foi o Inglês. Nesta decisão pesou também o facto do FEUPAutom estar desenvolvido nesta linguagem, o que tornaria ambos os softwares compatíveis até neste ponto.

### 3.4.2 Nome

O nome dado ao simulador foi FEUPSim: "FEUP" uma vez que é nesta faculdade que foi criado e será futuramente utilizado e "Sim"de simulação que será a sua principal função.

### 3.4.3 Metodologia adotada

Por forma a otimizar o trabalho e garantir que todos os prazos eram cumpridos, foi importante definir desde o início do projeto do simulador qual seria a metodologia adotada. Esta decisão revelou-se de uma importância extraordinária uma vez que foi através dela que se decidiu qual o método de trabalho que seria seguido na implementação do simulador. A decisão tomada foi de encontro à criação de uma tabela semanal que, tal como mostrado na Figura 3.3 refletisse:

- **Trabalho por fazer:** Lista de tarefas e objetivos a completar na presente semana;
- **Problemas:** Dificuldades encontradas na presente semana ou anteriores que ainda não tenham sido ultrapassadas;
- **Trabalho Futuro:** Lista de ideias para trabalho futuro ou implementação de funcionalidades que ainda não eram possíveis de criar naquela semana;
- **Dicas de Trabalho:** Atalhos de teclas para Lazarus e notas sobre o trabalho desenvolvido;

A cor preta era usada para escrever as tarefas a realizar e quando esta ficasse concluída ou um problema fosse resolvido eram passadas com um marcador verde por forma a marcar o ritmo de trabalho desenvolvido.

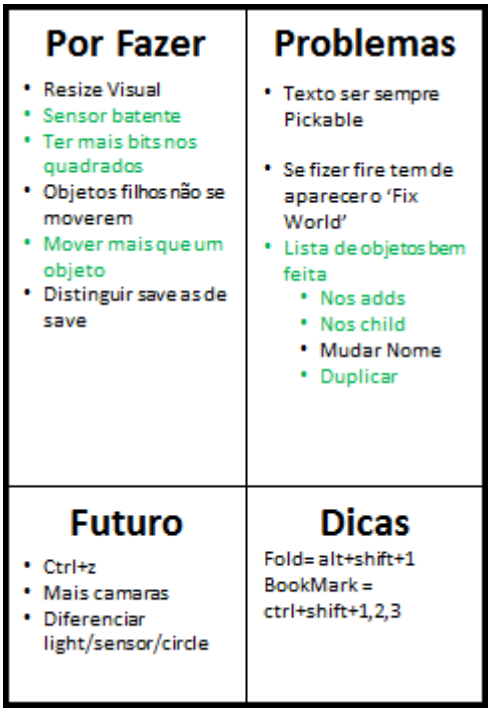



**Figura 3.3** – Metodologia de trabalho adotada

# Capítulo 4

# Implementação do Simulador

Depois de concluído o projeto do simulador, foi realizada a escolha do ambiente de desenvolvimento e linguagem em que o simulador seria implementado. Uma vez finalizada esta escolha e realizada alguma ambientação às ferramentas a utilizar, foi iniciada a implementação do simulador. Neste capítulo será apresentada a arquitetura do sistema, todas as funcionalidades implementadas, os objetos criados e as suas funções e, por último, os algoritmos mais importantes desenvolvidos.

Nota: A partir deste momento, na escrita da dissertação, será assumido o nome dado ao simulador, FEUPSim, para o referenciar sempre que necessário.

## 4.1 Ambiente de Desenvolvimento

### 4.1.1 Lazarus

Sendo o simulador a desenvolver um programa que terá de reagir a diversos eventos despoletados pelo utilizador na interface e/ou pela troca de informação com o programa que o controla, foi optado por usar uma arquitetura de programação orientada a eventos. Dentro dos diversos IDEs e linguagens que existiam para o desenvolvimento do simulador e uma vez que havia todo o interesse em migrar o FEUPAutom para Lazarus [23], este foi o *software* escolhido. Nesta decisão pesaram também as seguintes características deste IDE:

- O facto de ser gratuito e de código aberto;
- Ser uma ferramenta multi-plataforma, permitindo o seu uso em diversos sistemas operativos sem ter de alterar código. O moto do Lazarus é:"*Write Once, Compile Anywhere*";
- O facto de o Lazarus usar um *Free Pascal Compiler* (FPC) [24], que se encontra em constante desenvolvimento;
- A linguagem usada, o Pascal, ter uma curva de aprendizagem relativamente pequena e existir imensa informação disponível sobre esta;
- Existência de várias bibliotecas que facilitam a sua programação;
- Ter várias ferramentas de depuração, facilitando a procura de erros no código, fugas de memória, ou mesmo correr o programa passo-a-passo;
- O desempenho dos programas criados ser equivalente aos desenvolvidos em linguagem C;

Face às alternativas o Lazarus possui algumas desvantagens que são aqui destacadas. Ao longo de todo o projeto, a influência destas não se provaram relevantes mas é importante a sua referência de modo a realizar uma comparação justa entre todos os IDEs disponíveis:

- O Lazarus possui um *footprint* (tamanho do ficheiro) maior do que o Delphi uma vez que no executável é também guardada a informação de depuração;
- O facto de que em relação ao Java, o Lazarus, não ser verdadeiramente portável, isto é, embora o seu moto indique que possa compilar em qualquer lado, isto implica que o Lazarus existia no respetivo SO, o que não acontece com a maior plataforma móvel atual, o Android;
- A linguagem Pascal ser mais recente que a C, o que levou a uma menor adaptação por parte dos desenvolvedores;

### 4.1.2 Bibliotecas e tecnologias utilizadas

Para além da escolha do ambiente de desenvolvimento e linguagem a usar, foi necessário optar entre algumas tecnologias disponíveis para a criação do simulador. Dentro das escolhas realizadas destacam-se o motor de renderização 3D, o protocolo usado para comunicação e o formato de gravação dos ficheiros.

#### 4.1.2.1 GLScene

O GLScene [4] é uma biblioteca baseada no OpenGL, que por sua vez é uma API para realizar renderização de objetos 2D (duas dimensões) e 3D (três dimensões). Desta forma, o GLScene facilita a tarefa do programador uma vez que este não tem de conhecer todas as especificidades do OpenGL e tem ao seu dispor um conjunto de classes e funções que permitem o rápido desenvolvimento da aplicação. Sendo orientado à programação de aplicações ao invés da programação de jogos, o GLScene permite a criação de uma grande variedade de programas, desde de renderização da atmosfera terrestre, a aplicações médicas, desenhos digitais, design de interiores de casas, simulações de braços robóticos e voos entre muitos outros exemplos [25].

Esta biblioteca, quando associada ao Lazarus, permite desenhar e criar cenas 3D tanto em modo de design, através de componentes visuais, como em modo de execução, criando objetos à medida das necessidades. Para além isso está ainda otimizada para desperdiçar o mínimo de recursos do computador onde é utilizada. Por fim, é necessário referir que o GLScene é um projeto distribuído sob a Mozilla Public License (MPL) sendo por isso grátis e de código-aberto.

Dada toda esta versatilidade, o GLScene foi a ferramenta escolhida para a renderização de objetos 2D e 3D do simulador desenvolvido.

#### 4.1.2.2 Modbus TCP

Como protocolo de comunicação foi utilizado o Modbus TCP [26] devido à sua facilidade de implementação e ao facto de o FEUPAutom já comunicar através deste protocolo. Criado pela Modicon em 1979, é usado para estabelecer comunicação mestre-escravo/cliente-servidor entre diversos dispositivos. Devido à facilidade de implementação e uma vez que é gratuito, o Modbus tornou-se no protocolo mais usado em ambientes industriais e educacionais, tendo já sofrido diversas alterações com vista no seu melhoramento.

Como biblioteca de interface TCP foi utilizado o lNet pois este realizava de forma simples e transparente todo o envio/receção de pacotes necessária. De referir também que é assim garantido que o programa de controlo - FEUPAutom - e a simulação possam correr em computadores distintos, uma vez que é usada a Internet como meio de comunicação.

Existem 4 tipos de dados definidos pelo Modbus, tal como é visível na Tabela 4.1.

**Tabela 4.1** – Tipo de dados Modbus

| Nome | Tradução | Tipo de objeto | Tipo de acesso |
|---|---|---|---|
| Discrete Input | Entrada Discreta | 1 bit | leitura |
| Discrete Output (Coils) | Saída Discreta | 1 bit | leitura+escrita |
| Input Register | Registo de Entradas | 16-bit (*word*) | leitura |
| Output Register | Registo de Saídas | 16-bit (*word*) | leitura+escrita |

De uma forma geral, o protocolo Modbus é simples de implementar, uma vez que é apenas composto por duas grandes camadas, sendo elas o *Modbus Application Header* (MBAP) responsável por endereçar a camada de transporte e o *Protocol Data Unit* (PDU) responsável pela camada da aplicação, isto é contendo os dados, tal como ilustrado na Figura 4.1. De notar que os valores entre parêntesis representam o número de bytes que cada campo necessita numa mensagem Modbus. Desta forma, torna-se muito intuitivo o envio, receção e interpretação de mensagens Modbus, dado que depois de estabelecida a comunicação o servidor apenas tem de responder aos pedidos de leitura ou escrita enviados pelo o cliente(s).

O PDU do Modbus é ainda dividido em três tipos:

- PDU de pedido (função e dados)
- PDU de resposta normal (função e dados)
- PDU de resposta de erro (função+128 e código identificador do erro)

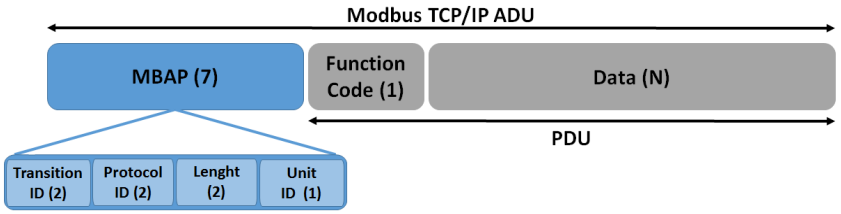



**Figura 4.1** – Trama uma mensagem Modbus TCP (adaptado de [27])

#### 4.1.2.3 XML

A última decisão de grande impacto no desenvolvimento do simulador foi a escolha do formato em que seriam gravados os ficheiros gerados. Para isso, foram estudadas algumas linguagens nativas do Lazarus e o formato de gravação do GLScene, mas rapidamente foram-se descobrindo falhas em ambos, dificuldades de implementação e/ou incompatibilidades entre diferentes sistemas operativos. Foi decidido então que se deveria realizar um estudo sobre a viabilidade do uso da *EXtensible Markup Language* (XML) para a gravação e carregamento de ficheiros gerados.

O XML, tal como o seu nome indica, é uma linguagem de marcação, isto é, define um conjunto de regras que originam um formato de fácil leitura tanto para um humano, como para uma qualquer aplicação computacional que entenda as regras definidas pela linguagem. Dentro das suas vantagens destacam-se:

- Facilidade de descrever dados;
- As etiquetas (*tags*) serem definidos pelo o utilizador, o que se traduz numa total liberdade na organização hierárquica do documento;
- Independência da plataforma onde é aplicado;

O facto do XML servir apenas como forma de descrever dados é também em si uma enorme vantagem, uma vez que torna todo o processo de verificação dos ficheiros apenas dependente da aplicação. Se por ventura houver novos campos dentro de uma etiqueta XML, estendendo assim o objeto em questão, a aplicação não sofrerá qualquer alteração pois apenas verifica os campos conhecidos. Transcrevendo as palavras de [28]: "One of the beauties of XML, is that it can be extended without breaking applications."

De forma a facilitar a compreensão da composição de um documento XML, é apresentado abaixo um exemplo simples de como seria realizado um catálogo de CDs:

```
<CATALOGO>
    <CD>
        <TITULO>The Wall</TITULO>
        <ARTISTA>Pink Floyd</ARTISTA>
    </CD>
    <CD>
        <TITULO>Muse</TITULO>
        <ARTISTA>The Resistance</ARTISTA>
    </CD>
</CATALOGO>
```

Tal como pode ser visualizado existe um objeto raiz, o Catálogo (a castanho), que é composto por diversos objetos hierarquicamente inferiores, os CDs (a vermelho) que por sua vez contem o título e o artista (a azul).

Por toda a sua simplicidade e pelo o facto do Lazarus suportar nativamente a escrita/leitura de ficheiros XML, este formato foi escolhido para a gravação e carregamento das simulações criadas. O formato gerado e as etiquetas específicas criadas serão apresentadas em 4.5.3.

## 4.2 Conceitos importantes

O FEUPSim possui dois modos de execução:

1. **Edição**: Na qual é possível criar e desenhar o simulador, além de:
   - Mover objetos;
   - Aceder às propriedades dos objetos;
   - Selecionar múltiplos objetos;
   - Modificar opções de projeto
2. **Simulação:** Onde o utilizador verifica o funcionamento do simulador criado através da execução das funções dos objetos: rotação, colisões ou cliques de botões;

É ainda importante esclarecer alguns conceitos inerentes ao simulador criado e que serão utilizados ao longo da Dissertação.

### 4.2.1 Cena

A "Cena" é o local da interface onde os objetos são criados e onde ocorre toda a simulação. Todos os objetos são criados na posição (0, 0, 0) podendo ser facilmente deslocados e colocados noutra posição clicando com o botão esquerdo do rato e arrastando-os para a posição desejada. Este elemento é fulcral ao simulador e, tal como os objetos, é renderizado pelo GLScene.

### 4.2.2 Filhos - *Childs*

A noção de filhos aqui apresentada está relacionada com o que em programação se chama "*child*" que tem como significado a inferioridade hierárquica de um objeto em relação a outro. Na aplicação criada este significado deve ser bem compreendido pelo utilizador pois é uma ferramenta poderosa que pode auxiliar na criação de simulações mais complexas e completas. Dentro das funcionalidades implementadas destacam-se:

- A posição do objeto filho é relativa em relação à do objeto pai, ou seja se a posição absoluta em relação à Cena de um objeto filho é (3, 3, 0), e se o objeto pai estiver na posição (1, 1, 0), entaõ a posição do objeto filho indicada será de (2, 2, 0);
- Movendo o objeto pai o objeto filho é movido na mesma razão de deslocamento;
- Se o objeto pai rodar, o objeto filho também roda;
- Um objeto filho sofre do mesmo escalonamento que o pai;
- Se o objeto pai se tornar invisível o objeto filho também se tornará;

### 4.2.3 Colisões

Para diferenciar FEUPSim de um SCADA, dotando-o de capacidades que estes não possuíam, foi desenvolvido um sistema de colisões que permite ao utilizador utilizar objetos do tipo Sensor (a ser explicado em 4.4.1.7) para realizar colisões com outros objetos e enviar a sinalização dessa colisão através de uma entrada Modbus pré-definida.

## 4.3 Arquitetura/Estrutura de Dados

Foi realizada um diagrama de classes da arquitetura do sistema sendo que a sua versão final se encontra apresentada na Figura 4.2.

Tal como pode ser observado, o FEUPSim está ligado a três classes distintas e sem estas o seu funcionamento era impossível. Existe uma ligação Modbus para conectar o simulador a entidades externas, uma janela de Entradas/Saídas para controlo autónomo da aplicação e ainda a Cena onde são criados os objetos.

Existem 7 tipos de objetos com atributos e funções distintas, mas que descendem de uma única classe - TGLObject - esta possui atributos e funções que são comuns a todos os objetos e que podem ser alteradas através das suas propriedades.

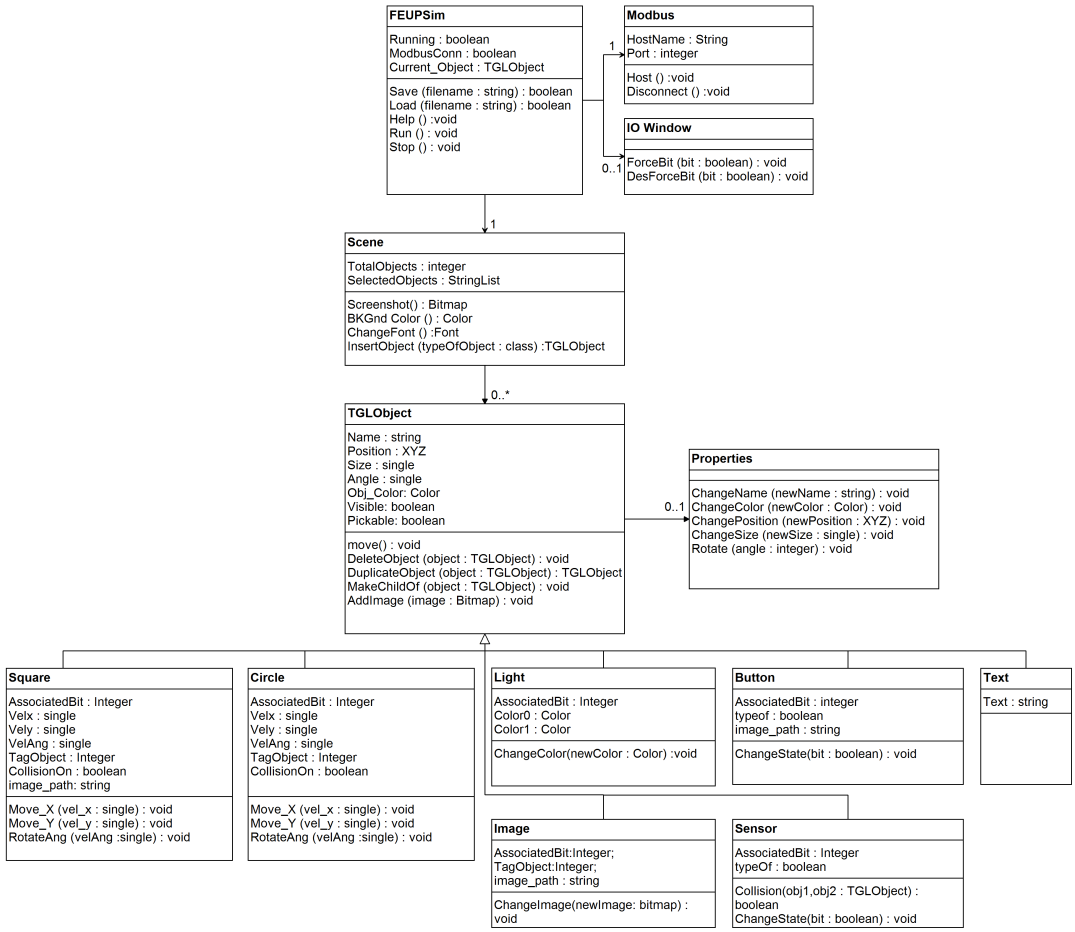



**Figura 4.2** – Arquitetura do sistema

## 4.4 Funcionalidades Implementadas

Nesta secção serão apresentadas todas as decisões tomadas na implementação das funcionalidades do simulador, os objetos passíveis de criar e as suas funções assim como o menu de contexto que auxilia na construção de simulações.

### 4.4.1 Objetos e suas funções

Quer através da barra de inserção de objetos (Figura 5.3) quer pelas opções do menu inserir (Figura 5.2), é possível introduzir novos objetos na Cena. Tal como pode ser visto nas figuras referidas, os objetos são: Imagens, Texto, Quadrados, Círculos, Luzes, Botões e Sensores. Todos estes objetos são agora apresentadas acompanhadas das funções implementadas.

#### 4.4.1.1 Imagens

As imagens existem por forma a facilitar a criação de animações. Alguns exemplos disso são um tapete com "movimento", um elevador a abrir a porta, máquinas com diferentes estados entre outros. A sua função consiste em ter um vetor dinâmico de bits e imagens associados, e quando existe um RE (*Rising Edge*), isto é, o valor do bit passa de FALSE para TRUE, a imagem muda para a que o utilizador definiu na associação bit-imagem. A opção por associar a mudança de imagem ao RE do bit foi tomada por forma a garantir que o simulador não estava constantemente a modificar a imagem a cada iteração, uma vez que se verificou que esta situação consumia muitos recursos computacionais e levava a uma maior lentidão da simulação.

A representação visual criada para as funções das imagens foi:

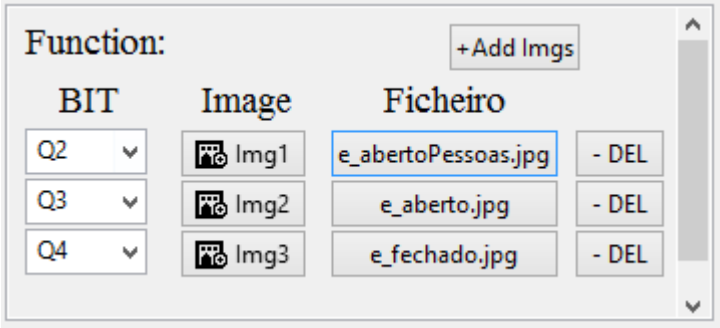



**Figura 4.3** – Função associada a uma Imagem

#### 4.4.1.2 Texto

O texto serve apenas como referência e como sinalização de outros objetos. Não possui qualquer função associada.

#### 4.4.1.3 Quadrados

Objetos cúbicos que permitem criar formas diversas quando conjugados, isto é, quando se juntam vários quadrados com diferentes rotações e tamanhos. É também possível associar uma imagem à face do quadrado. Ambas as características encontram-se representadas na Figura 4.4.


**Figura 4.4** – Exemplo do uso dos Quadrados

As funções de um quadrado incluem ter uma velocidade linear(x,y) e angular(graus/seg) tal como é mostrado na Figura 4.5. Para isso foi criado um vetor dinâmico de bits e velocidades associadas que um quadrado pode ter, até um máximo de 47 (uma por cada bit associado possível). Quando um destes bits está a TRUE, o quadrado adquire a velocidade linear e/ou angular definida para o bit que ficou a TRUE. Não é possível ter mais que um bit associado a TRUE, uma vez que o quadrado teria teoricamente de rodar a mais que uma velocidade ao mesmo tempo e, caso isso aconteça, a simulação dá um erro.

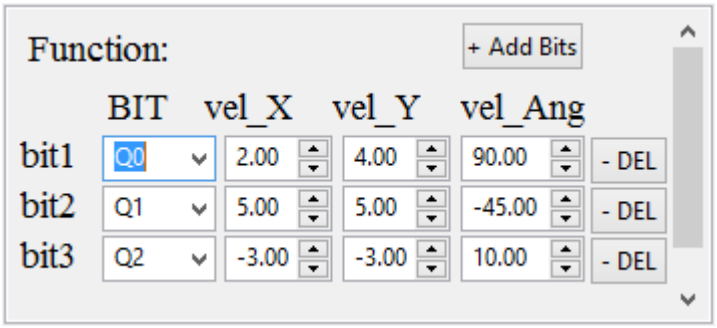



**Figura 4.5** – Função associada aos Quadrados e Círculos

#### 4.4.1.4 Círculos

Com os círculos é também possível criar objetos elípticos usando a propriedade de escala. Da mesma forma que se manipulou os quadrados, é também possível realizar o mesmo para os círculos de forma a criar novos objetos.

Em termos de funções estes objetos têm exatamente as mesmas que os quadrados, ou seja, várias velocidades lineares e angulares. A funcionalidade de movimento está no cerne do simulador uma vez que é o que distingue o FEUPSim de um qualquer SCADA, isto é, o facto de os objetos poderem possuir movimento real é um fator decisivo da diferenciação do FEUPSim.

#### 4.4.1.5 Luzes

O objeto luzes serve como representação de indicações luminosas ou sinalização de estados. A função associada às luzes é então a troca de uma cor por outra quando existe um RE (*Rising Edge* - bit passa de FALSE para TRUE) ou um FE (*Faling Edge* - bit passa de TRUE para FALSE) da variável associada, tal como é passível de ver na Figura 4.6.


**Figura 4.6** – Função associada à Luzes

#### 4.4.1.6 Botões

Assim como em qualquer interface se recorre a botões para efetuar alteração de estado, neste simulador é possível utilizar objetos do tipo Botão têm como função enviar sinais de entrada para o FEUPAutom. Existem dois tipos de botões:

- **Click:** O valor da entrada é TRUE enquanto o utilizador mantém o botão do rato esquerdo para baixo e, quando este deixa de clicar, o bit associado fica a FALSE.
- **Switch:** Com este segundo botão o valor do bit associado é trocado a cada clique por parte do utilizador representado, por exemplo, um interruptor de luz.

#### 4.4.1.7 Sensores

Por último são apresentados os sensores do FEUPSim. Este tipo de objeto permite, tal como os botões, enviar sinais de entrada para o FEUPAutom mas, em vez de funcionar através de cliques do rato, foi implementado um sistema de colisões que permitiu a criação de dois tipos de sensores:

- **Presence:** Sensor de presença em que, enquanto colide com outro objeto, mantém o valor do bit associado a TRUE e quando deixa de colidir coloca-o a FALSE.
- **Stop:** Modo de segurança que, quando algum objeto colide com um sensor do tipo 'stop', a simulação para e é dado um alerta de erro. Para uma melhor perceção de qual o sensor de segurança que colidiu, este começa a "arder", tal como ilustrado na Figura 4.7.

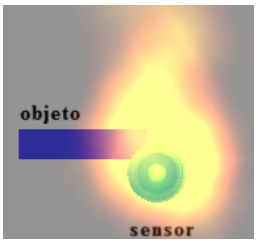



**Figura 4.7** – Exemplo do uso de um sensor do tipo Stop

### 4.4.2 Menu de Contexto

Clicando com o botão direito do rato em cima de um objeto, é possível chamar um Menu de Contexto (ou menu *popup*) que realiza determinadas ações consoante o objeto clicado.

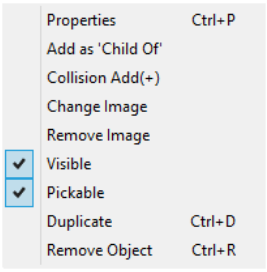



**Figura 4.8** – Menu de Contexto para um objeto Quadrado (exemplo)

Tal como a Figura 4.8 ilustra, existem diversas ações passíveis de serem realizadas, tal como apresentado abaixo.

**Properties - Propriedades**

Chamar a janela propriedades do objeto (ver mais sobre as Propriedades em 5.1.3).

**Add as 'Child Of' – Adicionar objeto como filho de**

Tal como explicado em 4.2.2, uma das funcionalidades cruciais consiste na hierarquização de objetos e, clicando nesta opção, é possível tornar o objeto selecionado como filho de outro.

**Collision Add(+) – Adicionar a colisões**

Apenas disponível para os objetos do tipo "Quadrado" e "Círculo", esta funcionalidade permite a tais objetos colidirem ou não com os sensores. Dando assim total liberdade ao utilizador de escolher quais os objetos que colidem.

**Change Image – Mudar Imagem**

Apenas disponível para os objetos do tipo "Quadrado", "Botão" e "Imagem", esta funcionalidade permite a tais objetos ficarem com uma nova imagem.

**Remove Image – Remover Imagem**

Tal como o seu nome indica, esta funcionalidade remove a imagem da face de um certo objeto.

**Visible – Visível**

Permite tornar um objeto visível ou invisível. Esta funcionalidade é bastante versátil e possibilita, por exemplo, possuir colisões com objetos invisíveis.

**Pickable – Objeto pode se mover**

Quando selecionada permite que um objeto selecionado possa ser movido com o rato.

**Duplicate – Duplicar**

Duplica o objeto com todas as suas propriedades. A implementação realizada é apresentado no algoritmo 1, que mesmo não se encontrando completo, uma vez que só complementa a duplicação de objetos Quadrados, serve para explicar, através de pseudo-código, esta função.

**Algoritmo 1** - Duplicar objeto

**Entrada:** Objeto1 (a duplicar)
**Saída:** Objeto2 (duplicado)

```
1:  if Objeto1 = NULL then
2:      exit;
3:  else
4:      if Objeto1.NomeDaClasse = 'Quadrado' then
5:          Objeto2 = NOVO(Quadrado);
6:          Copiar(Objeto1, Objeto2);
7:          Objeto2.Pai = Objeto1.Pai;
8:      else
9:          {Processo idêntico para outras classes de objetos}
10:     end if
11: end if
```

**Remove Object – Remover Objeto**

Remove o objeto da cena, mas mantém todos os seus filhos.

Na Tabela 4.2 é apresentado um resumo de quais as ações que o Menu de contexto pode realizar para cada objeto.

**Tabela 4.2** – Resumo menu de contexto

| Objeto | Quadrado | Círculo | Imagem | Luz | Texto | Botão | Sensor |
|---|---|---|---|---|---|---|---|
| **Propriedades** | sim | sim | sim | sim | sim | sim | sim |
| **Filho De** | sim | sim | sim | sim | sim | sim | sim |
| **Colidir** | sim | sim | não | não | não | não | sim |
| **Mudar Imagem** | sim | não | sim | não | não | sim | não |
| **Remover Imagem** | sim | não | sim | não | não | sim | não |
| **Visivel** | sim | sim | sim | sim | sim | sim | sim |
| **Movível** | sim | sim | sim | sim | sim | sim | sim |
| **Duplicar** | sim | sim | sim | sim | sim | sim | sim |
| **Remover** | sim | sim | sim | sim | sim | sim | sim |

## 4.5 Algoritmos desenvolvidos

### 4.5.1 Atualização da simulação

Para realizar a renderização 2D/3D foi utilizado o GLScene (como explicado em 4.1.2.1) e todo o controlo da simulação foi pendurado num objeto que esta biblioteca fornece para realizar a auto-progressão das animações, o *Cadencer* (cadência em Português). Este componente permite que as simulações do FEUPSim ocorram de forma suave e sem encravamentos.

Nas Figuras 4.9 e 4.10 são apresentados fluxogramas representativos do algoritmo de controlo da simulação e do algoritmo de verificação das funções dos objetos, respetivamente. De seguida passarão a ser descritos em detalhe ambos os algoritmos.

#### 4.5.1.1 Algoritmo de controlo da simulação

Uma vez iniciada a aplicação, o algoritmo de controlo é automaticamente despoletado e chamado a cada 20 milissegundos, o que se traduz num valor de cadência de 50 Fotogramas Por Segundo (FPS). Este algoritmo, presente na Figura 4.9, está estruturado da seguinte forma:

1. Verificar a existência de RE/FE das variáveis de sistema do FEUPSim;
   Colocar todas as entradas a FALSE;
2. Se o programa estiver em modo de execução e houver uma conexão Modbus, é iniciado o processo de simulação:
   (a) É verificado se existem colisões e/ou botões a ser clicados na interface, as entradas são escritas consoante o valor associado ;
   (b) É realizado um ciclo que itera **todos** os objetos da Cena e ,para cada objeto, é aplicado o algoritmo "alg2", apresentado abaixo na Figura 4.10, executando a função que lhes está associada.
3. Quando o ciclo termina a Cena é atualizada para contemplar as ações dos objetos;

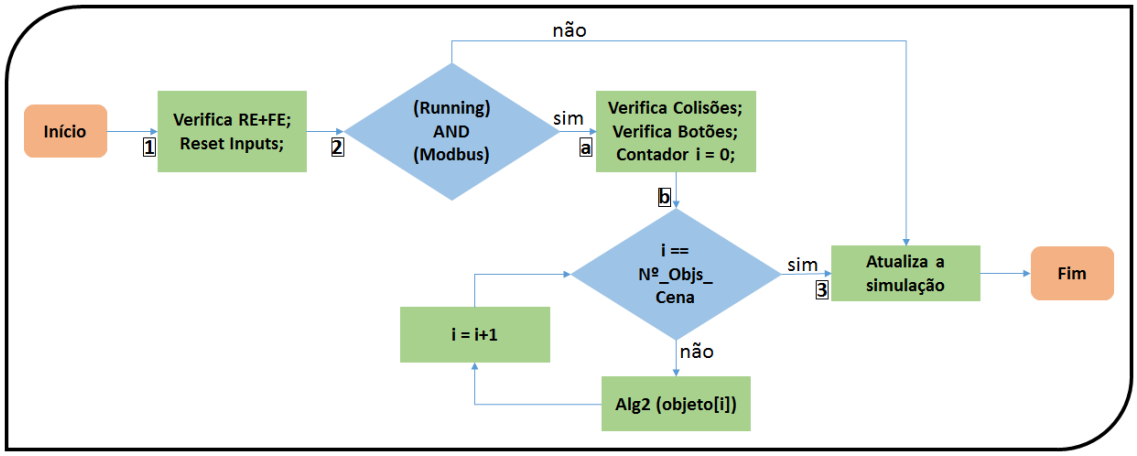



**Figura 4.9** – Ciclo de atualização da simulação

#### 4.5.1.2 Verificação das funções

Para garantir a execução das funções associadas a todos os objetos, foi desenvolvido um algoritmo recursivo que utiliza como entrada um objeto e tem por missão executar as funções associadas a este. Tal algoritmo encontra-se representado no fluxograma da Figura 4.10. As funções presentes na Figura foram explicadas em 4.4.1.

1. Em primeiro lugar é verificado o tipo de objeto passado a esta função;
   (a) Se for do tipo "Quadrado" ou "Círculo" é verificado se tem algum bit associado a TRUE e se assim for é executada a rotação e a translação consoante a velocidade angular e linear a que o objeto esteja sujeito, respetivamente;
   (b) Se o objeto for uma "Luz" é verificado se o bit associado sofreu um RE ou um FE e a cor do objeto mudará de acordo com a condição;
   (c) No caso de ser uma "Imagem"é verificado se o bit associado sofreu um RE e, em caso afirmativo é executada a mudança de imagem;
   (d) Por último, se o objeto não for de nenhum dos 4 tipos acima mencionados o algoritmo avança para o passo seguinte;
2. No final é realizado um ciclo que itera todos os objetos filhos do atual e, sobre cada um destes, é aplicado o presente algoritmo, por forma a que todos os objetos, independentemente da sua posição hierárquica, possam executar a função que lhes está associada.

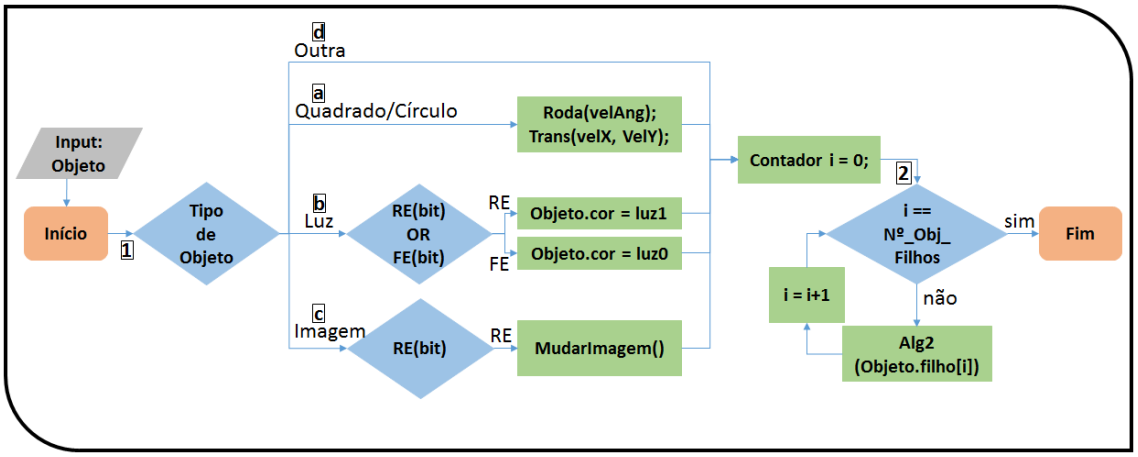



**Figura 4.10** – Verificação das funções - Alg2(objeto)

### 4.5.2 Comunicação Modbus

Como apresentado em 4.1.2.2 o protocolo de comunicação escolhido foi o Modbus. Das 20 funções passíveis de implementar com o Modbus, as utilizadas neste projeto foram as funções com os códigos:

- **1: *Read Coils*** -> Do lado do FEUPSim esta função tem como principal objetivo enviar todas as entradas para o FEUPAutom, isto implica que todos os cliques de Botões e colisões com Sensores são aqui enviados para o FEUPAutom;
- **3: *Read Holding Registers*** -> Complementar da função 1, mas em vez de enviar as mensagens como bits, estas são passadas como registos, isto é *words* (16 bits).
- **15: *Write Multiple Coils*** -> Esta função lida com todas as alterações nas saídas do FEUPAutom, o que se traduz na alteração dos objetos do tipo: Quadrado, Círculo, Imagem, Luz através da execução das funções destes;
- **16: *Write Multiple Holding Registers*** -> Complementar da função 15, mas em vez de enviar as mensagens como bits, estas são passadas como registos, isto é *words* (16 bits).

A escolha pela implementação destas funções surgiu, primeiramente, devido ao facto do FEUPAutom as utilizar e também por serem auto-suficientes no sistema de comunicação criado. Com as funções 1 e 15 era possível comunicar as Entradas/Saídas e as funções 3 e 16 serviriam para a troca de dados de inteiros uma vez que uma *word* permitiria enviar valores sem sinal entre 0 e 65.535 ou valores entre -32,768 e 32,767.

De seguida são apresentadas todas as funções implementadas, através dos pedidos e respostas que estas realizam:

***Read Coils* (1)**

- Pedido (FEUPAutom):
  - Código da função (1);
  - Endereço da primeira *coil* a ler;
  - Número de *coils* a ler (1 até 2000);
- Resposta Normal (FEUPSim):
  - Código da função (1);
  - Número de *bytes* enviados;
  - Estado das *coils* no formato de 8 por cada byte (cada *coil* representa 1 bit);
- Resposta de Erro (FEUPSim):
  - Número 129 (128+1);
  - Código identificador do erro (1,2,3,4);

***Read Holding Registers* (3)**

- Pedido (FEUPAutom):
  - Código da função (3);
  - Endereço do primeiro registo a ler;
  - Número de registos a ler (1 até 125);

- Resposta Normal (FEUPSim):
  - Código da função (3);
  - Número de *Bytes* (2*Nº Registos);
  - Registos enviados no formato de um registo a cada 2 *bytes*;
- Resposta de Erro (FEUPSim):
  - Número 131 (128+3);
  - Código identificador do erro (1,2,3,4);

***Write Multiple Coils* (15)**

- Pedido (FEUPAutom):
  - Código da função (15);
  - Endereço da primeira *coil* forçado/escreito;
  - Número de *coils* a forçar/escrever;
  - Número de bytes que se seguem;
  - Valor das *coils* no formato de 8 *coils* por cada byte (cada *coil* representa 1 bit);
- Resposta Normal (FEUPSim):
  - Código da função (15);
  - Endereço da primeira *coil*;
  - Número de *coils*;
- Resposta de Erro (FEUPSim):
  - Número 143 (128+15);
  - Código identificador do erro (1,2,3,4);

***Write Multiple Holding Registers* (16)**

- Pedido (FEUPAutom):
  - Código da função (16);
  - Endereço do primeiro registo a forçar/escrever;
  - Número de registos a forçar/escrever (1 a 127);
  - Número de *Bytes* (2*Nº Registos);
  - Registos enviados no formato de um registo a cada 2 *bytes*;
- Resposta Normal (FEUPSim):
  - Código da função (16);
  - Endereço do primeiro registo forçado/escrito;
  - Número de registos (1 a 127);
- Resposta de Erro (FEUPSim):
  - Número 144 (128+16);
  - Código identificador do erro (1,2,3,4);

### 4.5.3 Gravação e Leitura de ficheiros XML

Para gravação e leitura de ficheiros foi utilizada a linguagem XML, pela facilidade de edição e transversalidade entre plataformas, tal como explicado em 4.1.2.3.

Por forma a otimizar o tempo de escrita dos ficheiros foi utilizado um algoritmo recursivo que percorre todos os objetos da Cena, escrevendo-os corretamente. No processo contrário, o de leitura dos ficheiros guardados, existe um algoritmo recursivo idêntico que trata de toda a verificação e validação do documento. Se for detetada alguma alteração prejudicial, feita pelo utilizador o carregamento dos objetos na Cena é anulado e apresentada uma mensagem de erro ao utilizador.

Na Figura 4.11 é apresentada a estrutura criada e, como é visível, existem 2 etiquetas principais (marcadas com o número 0 a vermelho escuro):

**<Proj>:** Onde são definidas variáveis globais de projeto, tal como o *zoom* aplicado, a fonte de texto, a porta Modbus e a cor de fundo da Cena.

**<MyWorld>:** Nesta etiqueta são definidos todos os objetos, as suas relações e funções.

1. Em primeiro lugar é definido o objeto com os seus atributos: o nome, a classe do objeto e o número de objetos filhos (caixa azul);
2. São definidas as características comuns a todos os objetos: posição, visibilidade, escalabilidade, o ângulo de rotação e a sua cor (caixa verde);
3. De seguidas são definidas as características específicas de cada objeto na etiqueta *SpecificProperties*, tais como altura, largura, raio, texto (caixa laranja);
4. Existe uma etiqueta denominada *MyProperties*, que trata das propriedades que estão relacionadas com as funções de cada objeto (caixa vermelho);
5. Por último, são especificados os objetos filhos seguindo a mesma estrutura (caixa cinzenta);

```
0<Proj Zoom="50" Font="12" IOWindow="0" BackGroundColor="0.9411764741|0.9411764741|0.9411764741" MB_Port="5502" MB_IP="localhost"/>
0<MyWorld name="MyWorld" NumberOfOwnChilds="7" NumberOfChildsBelow="14" NameNumber="14" DuplicateCount="14">
 1<myButton Name="Desligar bot" ClassName="myButton" NumberOfOwnChilds="1" NumberOfChildsBelow="1">
  2 <Position>-0.6643199325|-1.220906496|0</Position>
    <Pick_Visible Pickable="-1" Visible="-1"/>
    <Scale>1|1|1</Scale>
    <RollAngle>0</RollAngle>
    <Material>....</Material>
  3 <SpecificProperties Width="0.5" Height="0.5"/>
  4 <MyProperties image_path="" AssociatedBit="1" typeof="0"/>
  5 <Childs>
      <TGLAbsoluteHUDText Name="Text14" ClassName="TGLAbsoluteHUDText" NumberOfOwnChilds="0" NumberOfChildsBelow="0">
        <Position>-0.3336904049|-0.3914794922|0.00003051757812</Position>
        <Pick_Visible Pickable="0" Visible="-1"/>
        <Scale>1|1|1</Scale>
        <RollAngle>0</RollAngle>
        <Material>....</Material>
        <SpecificProperties ModulateColor="0|0|0" Text="desligar"/>
        <MyProperties/>
      </TGLAbsoluteHUDText>
    </Childs>
 </myButton>
 1<myButton Name="Ligar_bot" ClassName="myButton" NumberOfOwnChilds="1" NumberOfChildsBelow="1">
  2 <Position>0.3616001308|-1.220159888|0</Position>
    <Pick_Visible Pickable="-1" Visible="-1"/>
    <Scale>1|1|1</Scale>
    <RollAngle>0</RollAngle>
```

**Figura 4.11** – Exemplo de um ficheiro XML gerado pelo FEUPSim

### 4.5.4 Histórico - Log Temporal

Por forma tornar o simulador num programa mais abrangente que contivesse os diversos subsistemas que compõem um SCADA (tal como visto em 2.2), existiu a necessidade de criar um histórico. Para isso, projetou-se quais as funcionalidades que deveriam ser guardadas, os campos, a frequência e o formato de gravação.

O formato de gravação utilizado foi o "*Comma-Separated Values*" (CSV), que pode ser traduzido por "Valores Separados por Vírgulas" e, tal como o nome indica, é uma representação de valores (números ou texto) que são guardados como texto simples de forma ordenada e separados por uma vírgula. Cada conjunto de dados é representado por uma nova linha no ficheiro e contém sempre o mesmo número de campos [29]. Este formato é muito usado para gravação de dados em tabelas uma vez que, usando cada conjunto de dados como uma linha e a separação por vírgulas como colunas, a tabela é construida sem esforço adicional.

Depois de decido o formato de gravação, foi necessário realizar um estudo aprofundado da aplicação desenvolvida sobre quais os campos que iriam ser guardados. Estes teriam de ser abrangentes por forma a que pudessem ser usados por diversos objetos/funções e, por isso mesmo, os campos escolhidos foram:

1. Hora e Data;
2. Tipo de funcionalidade: por forma ao utilizador saber o que gerou os dados, os possíveis eventos que dão origem a novas entradas de dados são:
   - Clicar num botão;
   - Haver uma colisão entre objetos;
   - Troca de Imagem ou Luz;
   - Mudança de velocidade dos Quadrados e Círculos;
   - Acessos às propriedades de um objeto;
3. Nome do Objeto 1;
4. Nome do Objeto 2: utilizado quando existe uma colisão;
5. Bit associado e tipo(I/O);
6. Informações adicionais a dar sobre os dados;

O histórico não é automaticamente iniciado sem ordem do utilizador e é ainda possível definir a periodicidade com que as amostras são tiradas (100ms - 10s). Para garantir que não é gerada informação desnecessária, o algoritmo implementado apenas guarda informação quando existe uma conexão Modbus e o programa está a correr. O ficheiro de histórico é criado na mesma pasta em que o FEUPSim se encontra, com nome "FEUPSimLogFile".

De seguida é apresentado um exemplo real do sistema implementado. Na Tabela 4.3 está representada uma possível disposição dos dados após tratamento, onde o período usado foi de 1 segundo.

Exemplo de dados retirados do ficheiro criado pelo FEUPSim:

26/06/2015-14:58:04, Collision, Sensor7, Square6, I0, Presence

26/06/2015-14:58:04, Light, LuzVerde, NA, O2, FE

26/06/2015-14:58:04, Light, LuzAmarela, NA, O1, RE

26/06/2015-14:58:05, Collision, Sensor7, Square6, I0, Presence

26/06/2015-14:58:05, Light, LuzVermelha, NA, O0, RE

26/06/2015-14:58:05, Light, LuzAmarela, NA, O1, FE

26/06/2015-14:58:05, Square, Square6, NA, O0, RE_Vel

26/06/2015-14:58:06, Collision, Sensor7, Square6, I0, Presence

26/06/2015-14:58:07, Light, LuzVerde, NA, O2, RE

26/06/2015-14:58:07, Light, LuzVermelha, NA, O0, FE

26/06/2015-14:58:07, Collision, Sensor7, Square6, I0, Presence

De seguida criou-se uma representação na forma de tabela dos dados gerados:

**Tabela 4.3** – Dados de de histórico tratados e dispostos numa tabela

| Data - Hora | Tipo | Nome_Obj1 | Nome_Obj2 | I/O | Info |
|---|---|---|---|---|---|
| 26/06/2015-14:58:04 | Collision | Sensor7 | Square6 | I0 | Presence |
| 26/06/2015-14:58:04 | Light | LuzVerde | NA | O2 | FE |
| 26/06/2015-14:58:04 | Light | LuzAmarela | NA | O1 | RE |
| 26/06/2015-14:58:05 | Collision | Sensor7 | Square6 | I0 | Presence |
| 26/06/2015-14:58:05 | Light | LuzVermelha | NA | O0 | RE |
| 26/06/2015-14:58:05 | Light | LuzAmarela | NA | O1 | FE |
| 26/06/2015-14:58:05 | Square | Square6 | NA | O0 | RE_Vel |
| 26/06/2015-14:58:06 | Collision | Sensor7 | Square6 | I0 | Presence |
| 26/06/2015-14:58:07 | Light | LuzVerde | NA | O2 | RE |
| 26/06/2015-14:58:07 | Light | LuzVermelha | NA | O0 | FE |
| 26/06/2015-14:58:07 | Collision | Sensor7 | Square6 | I0 | Presence |

# Capítulo 5

# Utilização e Resultados

O FEUPSim é uma ferramenta de ensino criada com o intuito de facilitar o desenvolvimento de simulações que ajudem os estudantes da UC de Sistemas e Automação a aprender os conceitos básicos de um simulador a nível industrial e introduzindo o conceito de SCADA numa fase inicial do curso. Devido a este facto, existe uma necessidade de, neste capítulo, explicar todas as escolhas efetuadas a nível de interface do simulador e funções extra que este possui, além dos objetos apresentados no capítulo anterior. São também apresentados dois casos de estudo, de forma a mostrar parte do potencial do simulador desenvolvido.

Por último, são apresentados resultados do desenvolvimento do FEUPSim, tanto em termos de satisfação de objetivos e requisitos, como na forma de um inquérito realizado aos estudantes.

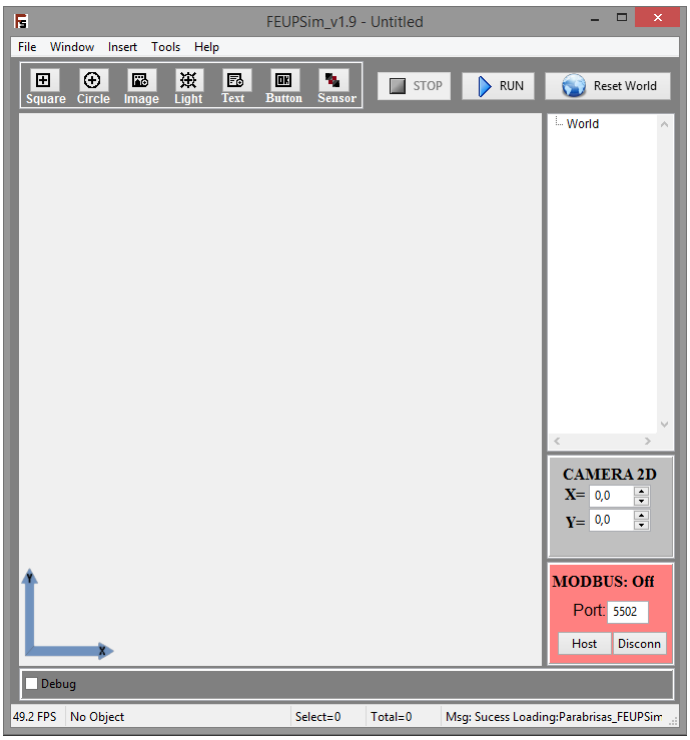



**Figura 5.1** – Interface FEUPsim

## 5.1 Interface Gráfica

Em qualquer aplicação social, móvel ou educacional é, hoje em dia, muito importante despender bastante tempo na criação da interface com o utilizador. Sendo um dos objetivos principais que o FEUPSim possua uma interface simples, apelativa e acima de tudo funcional, foram feitos todos os esforços para cumprir estes requisitos. A interface desenvolvida é apresentada na Figura 5.1 e, de seguida, são descritos os seus constituintes.

### 5.1.1 Janela Principal

Primeiramente é efetuada uma análise à janela de execução da simulação, com ênfase nas funcionalidades que não foram explicadas no capítulo 4, uma vez que se tratam de funcionalidades que não fazem parte base do simulador, mas sim da sua componente de utilização.

#### 5.1.1.1 Barra de menus

A barra de menus permite ao utilizador realizar diversas tarefas (Figura 5.2).

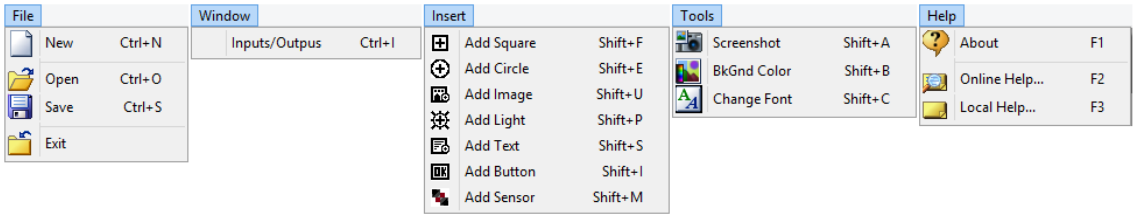



**Figura 5.2** – Vista individual de cada menu

- **File - Menu Ficheiro**
  - New: Criação de uma nova simulação;
  - Open: Abertura de um projeto gravado anteriormente;
  - Save: Gravação do estado atual do projeto;
  - Exit: Fecho da aplicação;
- **Window – Menu Janelas**
  - Visualizar ou não janela de entradas/saídas;
- **Insert – Menu Inserir**
  - Adicionar objetos à Cena, existindo, para cada, um atalho de rápida criação;
- **Tools – Menu de ferramentas**
  - Screenshot: Permite facilmente tirar uma fotografia da Cena;
  - BkGnd Color: Permite modificar a cor de fundo da Cena;
  - Change Font: Permite mudar a fonte do texto da Cena;
- **Help – Menu Ajuda**
  - About: Abre uma caixa de texto com informações sobre a aplicação;
  - Online Help...: Abre um site com uma wiki do simulador;
  - Local Help...: Abre o manual do utilizador;

#### 5.1.1.2 Barra de inserção de objetos

Esta barra (Figura 5.3) tem três funções:

- Inserção de objetos na Cena através de cliques no respetivo botão;
- Executar e Parar o programa;
- Clicando no botão "*Reset World*"é possível reverter todos os objetos às posições iniciais de um ficheiro que estivesse carregado;

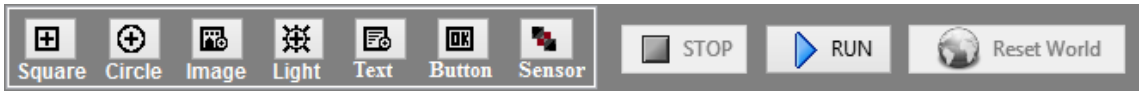



**Figura 5.3** – Barra de inserção de objetos

#### 5.1.1.3 Barra de estados

Nesta barra de estados é possível observar as seguintes variáveis e mensagens:

- Número de Fotogramas Por Segundo;
- O objeto selecionado e sua posição (X,Y,Z);
- O número de objetos selecionados;
- O número total de objetos na Cena;
- Mensagens de erros ou sucesso da aplicação;

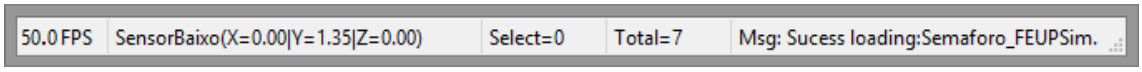



**Figura 5.4** – Barra de estados

#### 5.1.1.4 Comunicação Modbus

Existem quatro estados possíveis para a conexão Modbus, como é visível na Figura 5.5. Para iniciar a comunicação, basta clicar no botão *host* e esperar que um cliente estabeleça uma conexão.

- Vermelho: Não existe conexão;
- Laranja: O simulador está à espera de uma conexão;
- Verde: Representa uma conexão bem estabelecida;
- Amarelo: Ocorreu um erro de comunicação Modbus;

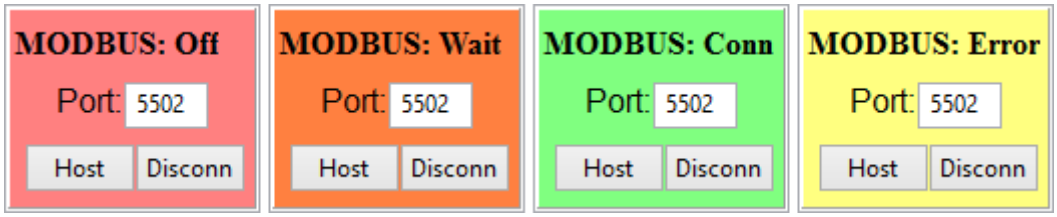



**Figura 5.5** – Janela de conexão Modbus

#### 5.1.1.5 Barra de depuração (debugging)

Janela (Figura 5.6) com algumas funcionalidades de depuração da aplicação:

- **Debug**: Serve para indicar se as opções de depuração estão visíveis ou não;
- **Properties**: Quando selecionada, implica que aquando da criação de um objeto a janela das propriedades seja imediatamente aberta;
- **Axes**: Permite que os eixos da Cena estejam visíveis quando selecionada;
- **Log**: Esta caixa serve para o utilizador decidir se deseja guardar os dados da simulação em histórico. É possível ativar ou desativar a qualquer momento esta opção e, por defeito, esta funcionalidade encontra-se desativada;
- **DelChild**: Indica se os objetos filhos são apagados aquando dos pais, ou se são mantidos;
- **3D cam**: O primeiro botão permite uma mudança de perspetiva da Cena (2D para 3D e vice-versa). No modo 3D o utilizador apenas pode estudar a cena construida;
- **Fire**: Serve para testar a componente de fogo do objeto selecionado.
- **FPS**: Número de Fotogramas Por Segundo a que o utilizador quer que a cena seja atualizada;
- **Log(ms)**: Tempo de atualização do ficheiro de histórico, isto é, tempo em milissegundos a que as variáveis e eventos são gravados para disco. Por defeito ocorre de 1 em 1s;
- **Lupas +/-**: Fazer *zoom* da Cena;

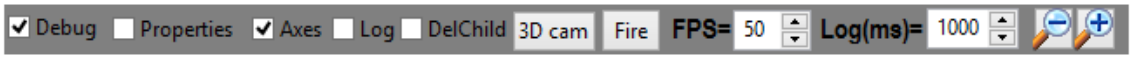



**Figura 5.6** – Barra de depuração

#### 5.1.1.6 Árvore de Objetos

Nesta árvore de objetos é possível verificar todos os objetos da Cena, ligações entre eles (quem é filho de quem) e realizar operações em cada um individualmente.

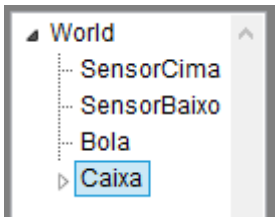



**Figura 5.7** – Árvore de Objetos

### 5.1.2 Janela de Entradas/Saídas

Esta janela foi criada com o intuito de ser possível examinar o estado das entradas/saídas do sistema e também ser possível forçá-las para verificar o correto funcionamento, antes de ligar o simulador ao FEUPAutom, transformando-o independente para pequenos testes. Quando uma caixa de seleção do lado dos *Outputs* (saídas) está desselecionada, o bit associado é FALSE e, quando se seleciona, clicando na caixa correspondente, e esta ficar com um "certo", então o bit associado fica a TRUE. A caixa de verificação do lado dos *Inputs* (entradas) não é possível de alterar, uma vez estes dependem do estado dos sensores ou botões que lhes estão associados.

### 5.1.3 Janela de Propriedades

Na imagem seguinte é possível verificar as propriedades que são comuns a todos os objetos. Sendo que as funções específicas de cada objeto foram já apresentadas na secção 4.4.

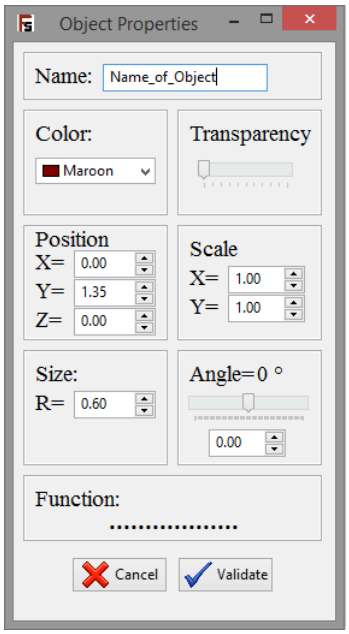



**Figura 5.8** – Janela de propriedades de um objeto

É importante referir que as propriedades vão sendo modificadas em tempo-real mas, caso o utilizador se arrependa pode cancelar e o objeto retornará ao seu estado inicial. Esta funcionalidade foi conseguida à custa da gravação prévia de todas as propriedades passíveis de modificar.

**Name – Nome do objeto**

Mudar o nome do objeto a editar.

**Color + Transparency – Cor + Transparência**

Com esta propriedade é possível modificar a cor bem como o nível de transparência do objeto em questão.

**Position - Posição**

Posição global na Cena. A componente Z é limitada de -1 a 1 para permitir objetos à frente uns dos outros.

**Scale - Escala**

Com esta propriedade é possível escalar um objeto em X e em Y.

**Size – Tamanho**

Permite modificar o tamanho do objeto. Se for uma Círculo/Sensor/Luz tem raio, se for um Quadrado/Imagem/Botão tem altura e largura.

**Angle – Ângulo de Rotação**

Com esta propriedade é possível rodar o objeto em relação à Cena, num ângulo de -180 a 180 graus.

## 5.2 Casos de estudo

Após a apresentação de todas as funcionalidades do FEUPSim, é importante, no âmbito da escrita da presente Dissertação, mostrar alguns casos de estudo que serviram para a realização de testes do simulador. Para isso, de seguida são apresentados dois exemplos e explicadas todas as características de interesse dos mesmos.

### 5.2.1 Luzes de Semáforos

O primeiro exemplo consiste na simulação de um semáforo de trânsito, uma vez que na UC de Sistemas e Automação o primeiro exercício que os alunos resolvem é precisamente o controlo de das entradas de um parque onde também existe um semáforo. A simulação, à esquerda na Figura 5.9, é constituída por 3 objetos "Luz" que, consoante o bit associado, mudam para a cor correspondente à do sinal de trânsito. Além das 3 luzes, existe ainda o poste do semáforo por forma a tornar a simulação mais atrativa visualmente. Isto demonstra que se forem utilizados vários objetos em conjunto com as funcionalidades até aqui descritas, é possível criar uma simulação muito próxima da realidade.

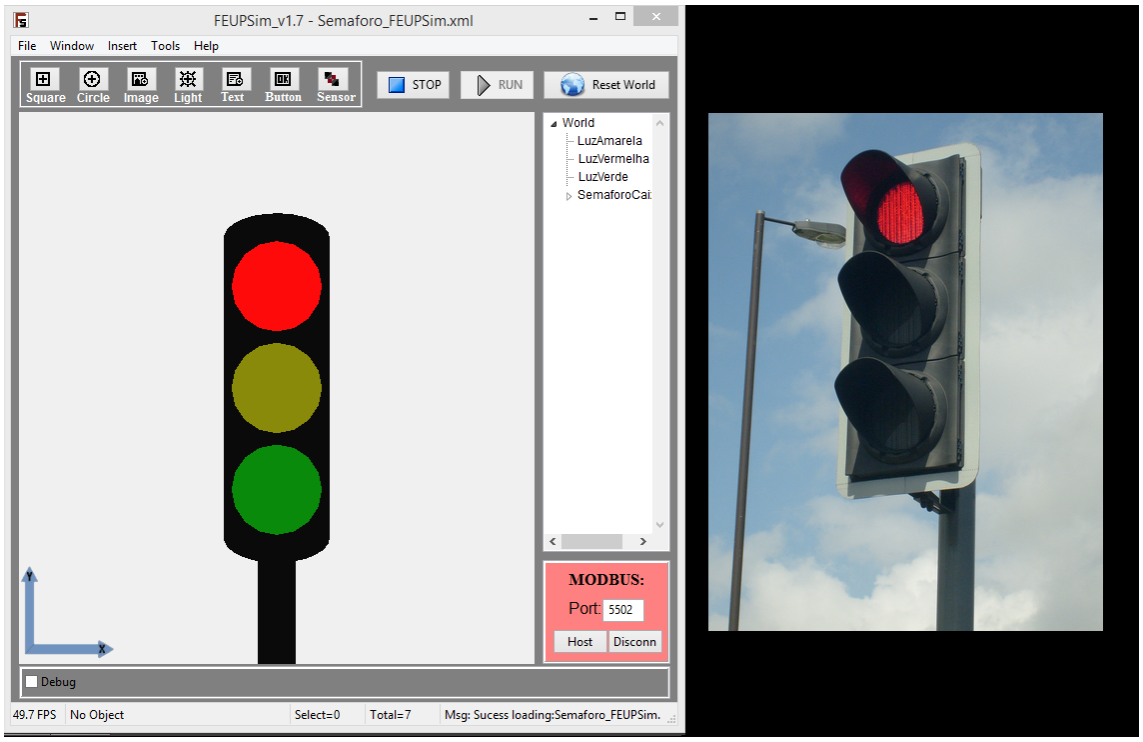



**Figura 5.9** – Comparação entre a simulação de um semáforo e a realidade (adaptado de[1])

[1] https://upload.wikimedia.org/wikipedia/commons/9/91/Modern_British_LED_Traffic_Light.jpg

### 5.2.2 Limpa Para-Brisas

Como segundo exemplo foi escolhido um limpa para-brisas de um carro, como é possível ver na Figura 5.10. Este exemplo é mais complexo que o anterior, necessitando por isso de uma decomposição mais detalhada, uma vez que contém quase todos os objetos passíveis de criar.

De forma a facilitar a explicação de cada elemento que compõe a simulação, foram sinalizados, na Figura 5.10, os números correspondentes à seguinte enumeração:

1. Para simular a escova, foi usado um objeto "Quadrado" com uma imagem de escova, que é filho de um objeto "Círculo" que possui uma velocidade angular permitindo a rotação no eixo central da simulação;
2. De seguida, foram colocados dois "Sensores" para que a rotação parasse aquando da colisão da escova com estes;
3. Foram ainda acrescentadas duas "luzes"para permitir visualizar qual dos motores estava ativo, se o motor 1 se o motor 2;
4. Por último, foram introduzidos dois botões que permitem ligar e desligar o sistema, através da simulação;

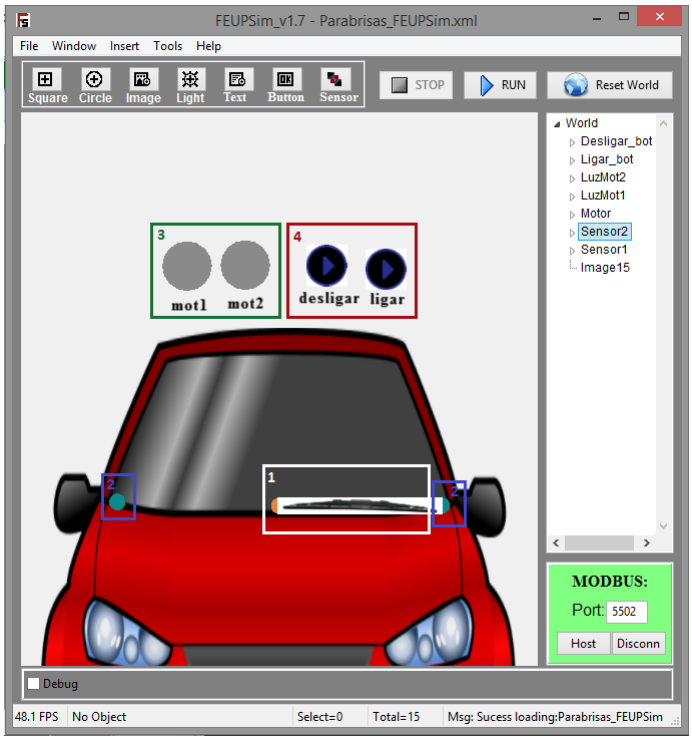



**Figura 5.10** – Simulação de Limpa Para-Brisas

## 5.3 Resultados Obtidos

Qualquer ferramenta atualmente desenvolvida tem, no final, que provar o seu valor, passar em testes de qualidade e ser melhor do que as concorrentes mais diretas. A fim de validar o FEUPSim foram verificados os requisitos, efetuados testes com estudantes e realizado um inquérito de satisfação da aplicação. Tal como mostrado na secção anterior (ver 5.2) foram realizados algumas simulações exemplo que facilitaram o uso e ambientação de estudantes e professores ao simulador. Uma das características fulcrais do simulador é o facto de este poder ser usado tanto por estudantes como professores de forma indiferenciada, possibilitando a partilha tanto de exercícios pelo professor para os estudantes resolverem, como também entre os próprios estudantes, de forma a otimizar o seu estudo.

### 5.3.1 Satisfação dos Requisitos

Na secção 3.2 da presente Dissertação foram descritos os principais requisitos idealizados para a aplicação e, nesta sub-secção, é efetuada a verificação do cumprimento dos mesmos.

Todos os requisitos funcionais foram implementados na aplicação, de entre os enunciados destacam-se o facto de possuir vários tipos de objetos passíveis de simular, a possibilidade de gravação e leitura de simuladores e a integração com o FEUPAutom. Foram ainda desenvolvidas bastantes funcionalidades extra não pensadas inicialmente, tal como: a personalização dos objetos, a existência de sensores que pegavam "fogo" e a possibilidade do simulador ser independente do FEUPAutom para certo tipo de testes.

Quanto aos requisitos não funcionais existe, uma maior dificuldade na avaliação os resultados obtidos e, por isso, foi realizado um inquérito a um grupo de estudantes, que será apresentado de seguida. Os únicos requisitos não funcionais que podem ser avaliados sem a necessidade é a existência do manual de instruções e do caderno de exemplos que se encontram em anexo à presente dissertação, Anexo A e Anexo B, respetivamente.

### 5.3.2 Testes com estudantes

De forma a obter uma resposta mais fidedigna em relação às vantagens do uso do FEUPSim em ambiente educacional e de forma a validar o programa criado, foram realizados testes com os estudantes que frequentaram a UC de Sistemas e Automação no ano curricular de 2014/2015.

#### 5.3.2.1 1ª fase - testes iniciais

Numa fase preliminar destes testes foi adotada a filosofia de iniciar com um grupo muito restrito de três alunos, de forma a verificar se o programa era intuitivo e funcional e o que precisaria de ser implementado antes de uma versão pública para todos os alunos.

Estes testes correram excecionalmente bem e a aplicação recebeu elogios por partes dos alunos, assim como algumas sugestões, tal como apresentado de seguida.

- Pontos positivos:
  1. "Excelente adição ao FEUPAutom"
  2. "Permite testar os programas em casa confirmando o seu funcionamento"
- Sugestões:
  1. Possibilidade de mover mais que um objeto de cada vez;
  2. Poder realizar um Reset ao mundo;
  3. Ter mais que duas velocidades nos quadrados e círculos;

#### 5.3.2.2 2ª fase - Inquéritos

Depois de analisadas e implementadas as sugestões efetuadas pelos os estudantes da 1ª fase de testes e após as novas adições que foram sendo desenvolvidas, houve um teste mais alargado e o *feedback* recebido foi formalizado num inquérito (anónimo) a que todos os estudantes responderam, sobre a sua opinião em relação ao FEUPSim. Os resultados desses inquéritos serão agora analisados.

A escala adotada está compreendida entre 1 e 5 onde cada um dos números corresponde a: 1) Discordo Totalmente, 2) Discordo, 3) Neutro, 4) Concordo, 5) Concordo Totalmente.

As duas primeiras perguntas consistiram na relação entre os dois *softwares* e estão representadas na Figura 5.11.

Quando perguntados se o FEUPSim seria uma boa adição ao FEUPAutom cerca de 73% dos estudantes concorda totalmente e apenas um valor residual de 7.7% discorda ou é neutro nesta questão.

Já na pergunta sobre a ligação entre FEUPAutom e o FEUPSim as opiniões não são tão consensuais e em alguns comentários foi denotado que a ligação não se encontra bem documentada, podendo os resultados obtidos ter sido influenciados por este fator. No entanto cerca de 70% concorda que a ligação é fácil e funcional.

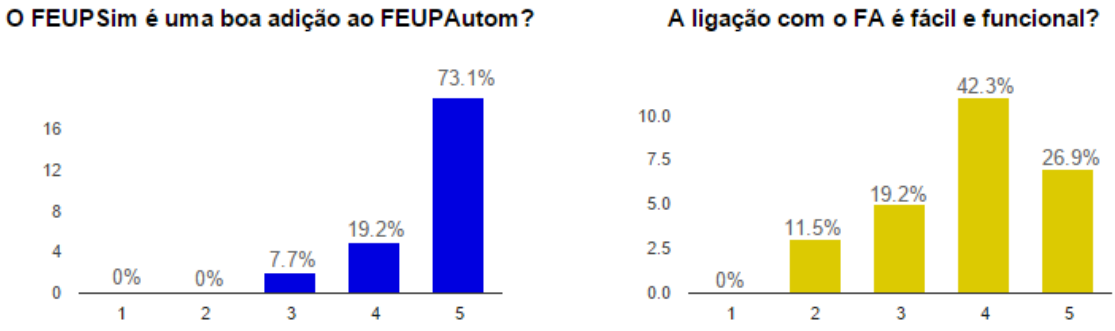



**Figura 5.11** – Inquéritos: Relação entre o FEUPSim e o FEUPAutom

As perguntas apresentadas na Figura 5.12 serviram para validar os requisitos não funcionais propostos, uma vez que se relacionavam com questões subjetivas do simulador implementado.

Desta forma, foi possível aferir se o simulador era considerado pelos os estudantes, simples e funcional, onde os resultados são bastante satisfatórios, uma vez que mais de 90% dos inquiridos respondem afirmativamente à simplicidade de interação com o simulador.

Quando perguntados se a ambientação ao simulador foi rápida, os valores da concordância rondam os 86.3%, com 42% dos alunos a responder com a componente máxima. É de notar que não existe nenhum aluno que discorde da pergunta. Com esta questão, pode concluir-se que a curva de aprendizagem do uso do simulador não é acentuada e que, com algum tempo investido, é possível um estudante inteirar-se de todas as suas funcionalidades.

Sendo ambos os resultados bastante positivos, conclui-se que os requisitos não funcionais 1 e 2 apresentados em 3.2 foram cumpridos com sucesso.

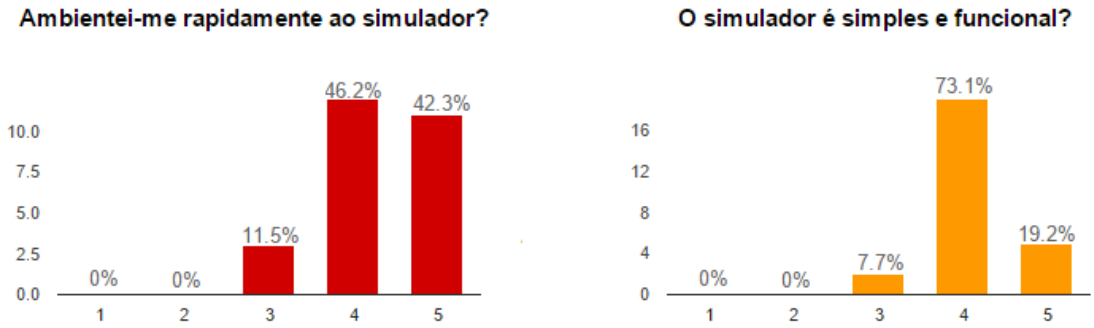



**Figura 5.12** – Inquéritos: Aptidões do FEUPSim

As duas últimas perguntas de resposta múltipla prenderam-se com questões mais estruturais do FEUPSim.

Na primeira foi questionado se o Manual e o Guião de Exemplos (ambos em anexo a esta dissertação) tinham sido úteis para a compreensão do funcionamento do FEUPSim. As ilações a tirar das respostas dos estudantes é que a escrita destes documentos foi uma mais valia, visto que mais de 80% concorda que ambos os documentos os ajudaram.

Já a última questão é um pouco distinta das restantes, uma vez que, perguntava aos alunos se o FEUPSim, com as simulações simples que consegue criar, serviria de facto para uma introdução mais precoce do FEUPAutom UC, permitindo assim aos alunos desenvolver um conhecimento mais aprofundado desta ferramenta. As opiniões são positivas, com 80% dos alunos a concordar que seria sim uma forma de iniciar a aprendizagem do FEUPAutom mais cedo na UC.

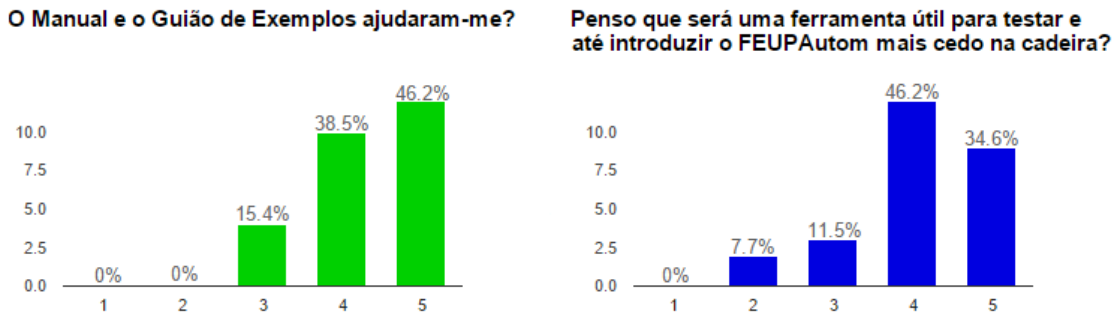



**Figura 5.13** – Inquéritos: Questões estruturais do FEUPSim

A última pergunta do questionário era de resposta aberta onde se pedia aos estudantes a sua opinião geral e quais eram, a seu ver, os pontos fortes e os pontos fracos do FEUPSim. Parte dos testemunhos dados pelos estudantes são aqui apresentados:

- "A ideia de criar facilmente um simulador foi muita boa, gostei do manual por ser simples e incidir no essencial. Em geral é bastante positivo e funcional. Porém tem muito por onde crescer e espero que assim aconteça"
- "O FEUPSim está um programa simples, no entanto cumpre bem a sua função e o seu objetivo. Penso que qualquer aluno poderá facilmente entender o seu funcionamento e em algumas horas aprender a trabalhar com ele a 100%. Bom Trabalho!"
- "Boa ferramenta adicional que permite mais interatividade o que nos dá mais incentivo para realizar as nossas simulações"
- "A função ctrl-z faz falta, mas penso que não seja indispensável. Uma possível integração com o FEUPAutom seria uma mais valia para os dois softwares."
- "Pontos fortes: ambiente gráfico. Pontos fracos: ligação ao FEUPAutom"
- "INCRÍVEL! Gostaria de ver, numa versão mais avançada, a opção de rodar os objetos em torno de um eixo que não apenas o seu centro, com o utilizador a escolher as coordenadas (ou até clicar no ponto) do eixo pretendido. Isto permitiria, por exemplo, fazer um pêndulo."
- "Na minha opinião a funcionalidade principal está bem desenvolvida. O FEUPSim funciona sem problemas e permite testar um grande leque de sistemas simples."
- "Se o FEUPAutom já era uma forte ferramenta de aprendizagem, o FEUPSim é um dos melhores *upgrades* que se poderia obter. Tem simplicidade de adaptação e utilização sendo, ao mesmo tempo,s possível explorar a complexidade de ambientes simulados ao sabor da imaginação e criatividade do utilizador. Poder observar uma simulação dinâmica ao invés de apenas ver "luzes"associadas ao valor booleano das variáveis é, sem dúvida, um ponto forte."

Todo *feedback* recolhido foi no geral positivo, com vários louvores ao facto da simplicidade inerente ao FEUPSim não prejudicar a sua utilidade e possibilidade de criação de diferentes simulações, tanto por parte dos estudantes como dos professores no ensino de Sistemas e Automação.

# Capítulo 6

# Atualização do FEUPAutom

Esta Dissertação contemplava dois grandes objetivos, tal como o seu título indica. Após a implementação do simulador, toda a atenção foi focada na atualização do FEUPAutom. Neste capítulo são descritos os objetivos específicos a atingir, a metodologia adotada e o resultado final da atualização efetuada, com alguns testes comparativos.

## 6.1 Problema e Objetivos

O FEUPAutom, como visto em 2.3.1, é uma ferramenta fulcral no ensino de controlo de sistemas de eventos discretos na UC de Sistemas e Automação. Com ele os alunos aprendem a modelizar problemas e resolver alguns exercícios práticos através da linguagem ST e Grafcet. Sendo uma ferramenta que se encontra em constante evolução, uma vez que todos os anos é recolhido algum *feedback* dos estudantes, foi necessário repensar se o *software* no qual o FEUPAutom estava a ser desenvolvido seria de facto o ideal para continuar a melhorar cada vez mais este softPLC.

O IDE onde o FEUPAutom se encontrava a ser desenvolvido era o Delphi 7, que era já muito antigo (Agosto de 2002) e o *setup* de programação envolvia o acesso a uma Máquina Virtual de Windows XP, estando por isso completamente obsoleto.

Por forma a garantir o contínuo desenvolvimento do FEUPAutom foi proposto que se atualizasse e migrasse todo o código de Delphi para Lazarus, uma vez que a linguagem de programação era a mesma (Pascal), daí do Lazarus ter surgido por forma a ser uma alternativa livre do Delphi, que é licenciado pela Embarcadero Technologies. Tal como explicado em 4.1.1, o Lazarus tem inúmeras vantagens das quais, para a atualização do FEUPAutom, se destacam:

- **Ser gratuito:** Desta forma será possível continuar a desenvolver o FEUPAutom sem ter que se pagar qualquer tipo de direitos de autor ou de licenciamento.
- **Ser multi-plataforma:** Uma das principais críticas dos estudantes relativamente ao FEUPAutom é o facto de ser apenas possível executá-lo no sistema operativo Windows.
- **Facilidade de Depuração:** Possuindo diversas ferramentas de depuração é possível, em Lazarus, encontrar e resolver erros mais facilmente, melhorar o código desenvolvido e ainda realizar um controlo da execução da aplicação passo-a-passo.

Tendo o Lazarus todas estas vantagens, o leitor pode ser levado a questionar o facto de não se ter optado, desde início, pela implementação do FEUPAutom neste IDE. Isto não aconteceu, uma vez que, na altura de criação da ferramenta o Lazarus não possuía todas estas funcionalidades e o Delphi 7 era o melhor ambiente de desenvolvimento para aplicação que se tornaria o FEUPAutom.

Para que a migração fosse bem sucedida foram definidos diversos objetivos:

1. Obter uma versão que compilasse em Lazarus
2. Converter o motor de *scripting* de *Delphi Web Script* (DWS) para *Pascal Script* (PS);
3. Obter um ficheiro único de XML para gravação e carregamento de projetos FEUPAutom;
4. Tornar o Modbus independente da biblioteca Indy;

## 6.2 Metodologia

Tendo em vista os objetivos propostos, a metodologia apresentada nesta secção pretende dar a conhecer todos os passos dados, assim como dificuldades e soluções encontradas.

O primeiro passo na migração do FEUPAutom consistiu em converter todo o seu código por forma a que a ferramenta compilasse em Lazarus. Tendo este novo IDE sido criado como alternativa gratuita ao Delphi, a sua equipa de desenvolvimento fornece uma ferramenta que possibilita a conversão de projetos criados em Delphi para Lazarus.

Uma vez efetuada esta conversão automática, surgiram diversos problemas. A inexistência de bibliotecas e componentes que eram utilizados no Delphi impediram que existisse desde logo uma versão preliminar funcional do FEUPAutom. Para partir o problema em diversos pedaços que pudessem ser tratados individualmente, todo o código e componentes não existentes foram comentados e foi realizada uma lista de tarefas.

Para se chegar à primeira versão compilada do FEUPAutom em Lazarus não foi necessário realizar alterações drásticas ao código fonte, uma vez que o processo automático de conversão já tinha efetuado parte da atualização necessária. No entanto, foi realizado bastante trabalho a nível de conversão de parâmetros das interfaces e alteração de código relativamente à execução de ficheiros FEUPAutom. Uma vez compilado o FEUPAutom em Lazarus, foi possível aferir quais os módulos funcionais e quais os não funcionais.

Serão agora descritas todas as adaptações/modificações realizadas e melhorias implementadas a todos os módulos não funcionais do FEUPAutom.

### 6.2.1 Motor de Scripting

Sendo o FEUPAutom um programa de controlo de eventos discretos, necessita de um código que permita executar esse mesmo controlo. Para isso fornece aos utilizadores um editor textual de linguagem ST e um editor visual de Grafcet, onde é criado o seu controlador para um dado sistema. Para que seja possível correr em tempo de execução código gerado pelos os utilizadores sem ter que recompilar todo o programa, é necessário que a aplicação possuía um motor de *scripting*, isto é, um motor que compile e execute todo o código escrito pelos os utilizadores.

No Delphi era utilizada a biblioteca DWS que era capaz de interpretar código Pascal, o que levou à criação de um conversor de linguagem ST e Grafcet para Pascal. No entanto, tal biblioteca não existe para Lazarus. Dada a existência de uma alternativa bastante viável ao uso do DWS, chamada *Pascal Script*, da empresa RemObjects no IDE Lazarus, foi optado por usar esta nova biblioteca que possuía vantagens em relação à usada anteriormente. Dentro das vantagens destacam-se o facto de ser livre e de código-fonte aberto, ser mais rápido que o DWS, permitir o uso de apontadores de variáveis e ainda ao facto de ser possível pré-compilar e guardar os programas executados. Como desvantagens, apresentava a notória falta de documentação que foi colmatada fazendo uso dos exemplos fornecidos e recorrendo ao fórum oficial do Lazarus.

A maior alteração a nível de implementação realizada ao FEUPAutom foi então a troca do motor de *scripting*. Uma vez que o PS permitia o uso de apontadores de variáveis, a comunicação entre o código de execução e o código gerado pelos os utilizadores foi muito simplificada, isto é, ao invés de haver uma constante reinterpretação das variáveis de código do utilizador, todo o processo ficou transparente e as variáveis de sistema utilizadas por ambos são as mesmas. A Figura 6.1 pretende representar a alteração efetuada onde, em cima é apresentado o motor de *scripting* DWS e em baixo a nova versão usando o PS.

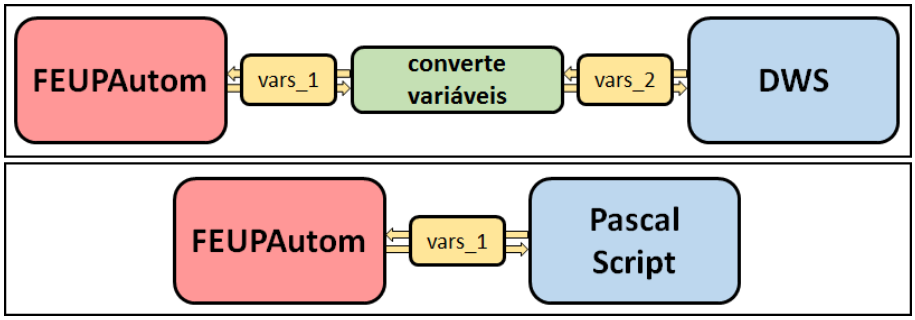



**Figura 6.1** – Motores de *scripting* FEUPAutom: em cima o DWS e em baixo o Pascal Script (PS)

Para verificar os resultados foram efetuados 5 testes e medidos os tempos de compilação da aplicação antiga, com DWS, e da nova versão do FEUPAutom, com PS, utilizando para isso o mesmo computador[1]. Tal como é visível na Tabela 6.1, o motor de PS foi, em média, cerca de 133% mais veloz em todos os testes efetuados, que o DWS.

**Tabela 6.1** – Comparação entre o tempo de compilação do DWS com o do PS

| | **DWS (ms)** | **Pascal Script (ms)** | **PS - DWS (ms)** | **DWS/PS (%)** |
|---|---|---|---|---|
| **Exe 1** | 412 | 267 | 145 | 154% |
| **Exe 2** | 361 | 332 | 29 | 109% |
| **Exe 3** | 746 | 477 | 269 | 156% |
| **Exe 4** | 478 | 361 | 117 | 132% |
| **Exe 5** | 1450 | 1277 | 173 | 114% |
| | | **Média =** | **147** | **133%** |

[1]SO: Windows 8.1; CPU: Intel i5-4200M @ 2.50GHz; RAM: 6GB DDR3; Gráfica: GeForce GT 740M 2GB

### 6.2.2 Leitura e Gravação de projeto FEUPAutom (XML)

Uma das críticas frequentes ao FEUPAutom consistia na forma como este grava um projeto. Tais críticas referiam-se ao facto do FEUPAutom gerar dois ficheiros separados, o que levava, por vezes, a confusão de qual o ficheiro de entrega dos trabalhos executados. Os dois ficheiros utilizavam diferentes tecnologias de gravação: o projeto e código ST eram gravados recorrendo a um sistema de ficheiros *INI* e a modelação Grafcet era gravado no formato XML, usando a biblioteca ECXMLParser. Dada a inexistência desta biblioteca em Lazarus, o facto do XML ser uma linguagem estandardizada com todas as vantagens enumeradas em 4.1.2.3 e aliado ao facto do Lazarus suportar nativamente o XML, esta linguagem foi a escolhida para criar um ficheiro único de gravação e leitura de projetos FEUPAutom.

Para juntar ambos os ficheiros foi criada a estrutura XML, apresentada na Figura 6.2, sendo esta composta por duas etiquetas principais:

- **FAProject -** Contém todas as variáveis de projeto:
  1. FAProject
  2. Janelas do FEUPAutom: Main, Variables e IOLeds;
  3. Propriedades das janelas: Variáveis próprias, geometria da janela, o código ST gerado e o nome de todas as variáveis;
- **G7Projeto -** Contém toda a modelação Grafcet:
  1. G7Projeto
  2. Etiqueta de Objetos;
  3. Objetos individuais e propriedades únicas: nome, localização XY, tipo entre outras;

```
1<FAProject>
2 <FMain>
3  <Props1 ActiveTab="1" MessagesHeight="100"/>
   <Props2 ModBusIP="localhost" ModBusPort="5502" ModBusNRead="48" ModBusNWrite="48" ModBusOffs_I="5392
3  <Geometry top="0" left="0" height="728" width="538"/>
3  <ST_Code count="283">
  </FMain>
2 <FVariables>
3  <Props Visible="0" NextVar="-1" Filter="0"/>
3  <Geometry top="48" left="746" height="456" width="269"/>
3  <Vars_Names count="641">
  </FVariables>
2 <FIOLEDs>
3  <Props Visible="-1"/>
3  <Geometry top="0" left="538" height="728" width="203"/>
  </FIOLEDs>
</FAProject>
1<G7Project>
  <Graph>
2  <Objects>
3   <Obj Name="X0" Page="0" CellX="1" CellY="4" BarInIdx="6" BarOutIdx="0" Type="1" Flags="1">
3   <Obj Name="t0" Page="0" CellX="1" CellY="6" BarInIdx="0" BarOutIdx="1" Type="2" Flags="0">
3   <Obj Name="X1" Page="0" CellX="1" CellY="8" BarInIdx="1" BarOutIdx="2" Type="1" Flags="0">
    </Objects>
  </Graph>
</G7Project>
```

**Figura 6.2** – Novo XML

Com este novo formato de gravação torna-se mais fácil guardar variáveis extra do sistema e é esperado que os estudantes percebam melhor todo o funcionamento do FEUPAutom. Uma vez que as versões anteriores do FEUPAutom não poderiam ser deixadas de parte, foi garantida a compatibilidade com o formato antigo de leitura de ficheiros.

### 6.2.3 Comunicação Modbus

Aquando da implementação do Modbus na versão original do FEUPAutom, foi utilizada uma biblioteca (DelphiModbus) que já disponibilizava todas as funções de Modbus necessárias. A desvantagem desta é que usava uma outra biblioteca associada que era bastante pesada computacionalmente, o Indy. Uma vez que ambas as bibliotecas tinham já sido migradas para Lazarus não foi colocada qualquer questão quanto ao seu uso mas, quando se tentou instalar estas bibliotecas numa máquina Linux - o interesse é que o FEUPAutom se torne multi-plataforma -, estas continham incompatibilidades.

Por forma a dar mais um passo no sentido de tornar o FEUPAutom verdadeiramente multi-plataforma, foi decidido que se adaptaria o código das funções Modbus necessárias para um novo formato e utilizando, para isso, a biblioteca de Lazarus 'lNet' que já tinha sido usada na implementação do simulador. Esta escolha foi efetuada uma vez que a biblioteca 'lNet', acrónimo de *Lightweight Networking Library* que se pode traduzir em português por 'Biblioteca leve de redes Internet', e que tal como o seu nome indica, é leve e pode-se assim diminuir a pegada computacional do Indy, tal como tinha sido proposto nos objetivos.

A nível de implementação as alterações foram impercetíveis para o código principal, uma vez que as funções usadas por este foram refeitas, mantendo os mesmo dados de entrada e saída. Para isso foi criado uma nova unidade no projeto de Lazarus intitulada de *Modbus_Utils*, onde todas as funções Modbus utilizadas pelo o FEUPAutom foram implementadas, tornando assim a aplicação dependente apenas de uma única biblioteca, que é multi-plataforma, em vez das duas usadas nas versões antigas, que não cumpriam este requisito.

### 6.2.4 Novas funcionalidades

Por último é necessário referir que foram realizadas diversas alterações gráficas no FEUPAutom, tanto a nível estético como de funcionalidades. Dentro delas destacam-se:

1. Atualização do logo FEUPAutom e da versão para 5.0;
2. Novos botões de correr e parar o programa que uniformizam o design e posterior inclusão na janela de Grafcet, facilitando assim a execução dos controladores desenhados;
3. A tecla 'Del' já funciona para eliminar objetos em Grafcet;
4. Novas opções adicionadas ao menu;

## 6.3 Resultados Obtidos

Por forma a avaliar a qualidade do trabalho efetuado na atualização do FEUPAutom foram executados alguns testes comparativos entre a nova versão em Lazarus e antiga em Delphi.

Parte dos resultados foram já apresentados em 6.2.1, onde se compara o tempo de compilação do algoritmo de ambas as versões com o Pascal Script, sendo sempre mais rápido que o DWS usado na versão antiga. Quanto ao novo formato de gravação, o XML, dada toda a sua versatilidade, as melhorias são notórias tanto a nível de desempenho e velocidade de gravação/leitura, como de interpretação dos dados que se tornam agora muito mais percetíveis, em termos de organização e hierarquização, para um utilizador humano. Além destes fatores o XML, unificou ambos os ficheiros que eram criados na versão antiga do do FEUPAutom num único. Foi também refeita toda a comunicação Modbus, tornando-a muito mais estandardizada a nível de bibliotecas que a aplicação usa para a realizar. Ao longo de todo o trabalho de atualização do FEUPAutom, foram realizadas diversas melhorias na interface e código, dotando-o assim de uma maior fluidez e usabilidade para os utilizadores finais.

Em suma, o FEUPAutom encontra-se atualmente muito mais próximo de se tornar um programa verdadeiramente multi-plataforma. A migração para Lazarus e todas as adaptações realizadas aos módulos essenciais, permitiram que o FEUPAutom seja agora um programa que pode continuar o seu desenvolvimento, livre de limitações impostas pelo o IDE, uma vez que o Lazarus está em constante atualização e encontra-se muito mais evoluído que o “velhinho” Delphi 7 ,que era usado até agora.

# Capítulo 7

# Conclusões e Trabalho Futuro

## 7.1 Satisfação dos Objetivos

A busca por alternativas gratuitas de *software* para uso educacional é cada vez mais dificultada, tanto pela sua inexistência, como pela crescente adoção de sistemas de subscrição para uso das aplicações. Sendo cada vez mais importante avaliar e gerir da melhor forma o dinheiro disponível nos cursos de ensino superior, a opção pelo desenvolvimento de ferramentas específicas para os estudantes torna-se mais atrativa.

O principal objetivo desta dissertação prendeu-se com a desenvolvimento de uma ferramenta que permitisse a criação de Simuladores/SCADAs, por parte de estudantes e professores. Tal ferramenta devia estar intrinsecamente ligada ao FEUPAutom e possibilitar a simulação da maior quantidade de cenários possíveis. Para cumprir todos os requisitos propostos, o projeto foi desenvolvido por forma a tornar a ferramenta o mais genérica possível, deixando a cargo do utilizador o processo de desenho das simulações.

Dentro de todas as funcionalidades implementadas no FEUPSim, destacam-se a diversidade de objetos passíveis de simular, a simplicidade da interface, o sistema de colisões implementado, o histórico de eventos e ainda a facilidade de gravação/leitura das simulações criadas. Com estas funcionalidades foi possível criar diversas simulações para testes da ferramenta, tendo estas se revelado um sucesso a nível de exemplo para os estudantes. Para além disso, estes testes podem ser considerados uma prova da viabilidade do uso do FEUPSim na indústria como um SCADA, dada a sua possibilidade de funcionar remotamente do centro de controlo. É ainda de referir que todos os requisitos e objetivos inicialmente propostos foram cumpridos e validados, tanto pela a análise crítica efetuada, como pelos estudantes da UC.

Na atualização do FEUPAutom para Lazarus foram encontradas dificuldades devido, principalmente, à inexistência de diversas bibliotecas que eram utilizadas na versão antiga, programada em Delphi. Todos os problemas foram resolvidos, quer por criação de novos métodos e algoritmos, quer por uso de novas bibliotecas. Desta forma garantiu-se que o FEUPAutom se encontra apto a posteriores desenvolvimentos e mais próximo de se tornar um programa multi-plataforma. É ainda de ressalvar todo o trabalho realizado na transformação dos ficheiros de gravação/leitura

para o formato XML, que sendo *standard* abre hipóteses à adição de mais variáveis de projeto de forma simples e sem quebrar a compatibilidade com versões antigas.

Uma das mais importantes decisões tomadas nesta Dissertação prendeu-se com ambiente de desenvolvimento escolhido. No final, a escolha pelo Lazarus revelou-se em tudo acertada uma vez que, embora as duas aplicações tenham sido desenvolvidas sob o SO Windows, o FEUPSim já compila e executa tanto em Windows como Linux e, por sua vez, o FEUPAutom compila e executa em Windows está em testes na plataforma Linux.

Por todos os resultados obtidos, é possível constatar que os estudantes, de facto, têm muito a ganhar com o desenvolvimento de ferramentas criadas de acordo com as suas necessidades e que esta solução é viável no mundo educacional. Ambas as ferramentas abordadas nesta Dissertação surgiram de diferentes necessidades dos estudantes da UC de Sistemas de Automação e foram recebidas por estes como um fator decisivo na sua aprendizagem.

Além de todo o trabalho já mencionado, foi ainda realizado um artigo, já submetido e aceite, no âmbito da conferência EDULEARN15, a decorrer de 6 a 8 de Julho de 2015 em Barcelona. Tal artigo intitulado de "*Development of a student-centred tool that promotes deep learning in the technical area of control of discrete event systems*" pode ser consultado no Anexo C e centra-se na questão mais explorada ao longo desta Dissertação, isto é, na importância de criar ferramentas específicas que vão de encontro às necessidades dos estudantes. O principal foco é o FEUPAutom, uma vez que aquando da escrita o FEUPSim ainda não estava completo. Além do artigo foi ainda realizada uma apresentação virtual para a conferência em questão.

Em suma, pode ser concluído que **todos os objetivos propostos foram cumpridos** e que a aplicação criada e a atualização efetuada são, de facto, uma mais-valia para o ensino da UC de Sistemas e Automação.

## 7.2 Trabalho Futuro

Uma ferramenta de ensino nunca está totalmente concluída e é sempre possível melhorá-la. Nesta secção serão explicitadas algumas das funcionalidades que poderiam ser acrescentadas a ambos os programas desenvolvidos.

Embora se tenha implementado diversas funcionalidades extra para além das inicialmente pensadas, o **FEUPSim** tem ainda muito por onde crescer e se tornar cada vez mais numa ferramenta essencial no ensino. A listagem efetuada, representa algumas das funcionalidades de valor acrescido, possíveis de ser adicionadas ao FEUPSim:

- Redimensionamento visual dos objetos;
- Implementar a funcionalidade de 'Ctrl+z' por forma a desfazer a última alteração efetuado no desenho do simulador;
- Mais tipos de objetos: Controlo de nível, *sliders*, entre outros;
- Maior flexibilidade no tipo de funções executadas pelos objetos, isto é em vez de apenas serem variáveis booleanas existirem valores inteiros que tivessem outras funcionalidades;
- Integração com o FEUPAutom para estudo do comportamento temporal;

Todo o trabalho de atualização do **FEUPAutom** foi executado por forma a possibilitar que esta se tornasse multi-plataforma. Embora alguns testes já tenham sido efetuados, seria necessário um estudo mais aprofundado nesta área, uma vez que a ferramenta utiliza bibliotecas específicas da plataforma Windows. Sugere-se ainda que seja realizado algum trabalho a nível de otimização do ciclo de controlo por forma a tornar a aplicação mais eficiente e, por último, seria necessário implementar as sugestões sugeridas em 2.5.1, por forma a tornar o Grafcet do FEUPAutom totalmente compatível com a norma IEC 60848.

Seria ainda necessário realizar testes com *hardware* real, por forma a validar quer o FEUPAutom quer o FEUPSim a nível industrial como ferramenta de SCADA. Para isso seria necessário desenvolver um subsistema que permitisse comunicação entre vários servidores na rede Modbus, o que implicaria a criação de um *router*.

Por fim, poderiam ainda ser estudadas formas de automatização e validação de programas de controlo criados no FEUPAutom pelos estudantes, através da análise do histórico do FEUPSim. Isto possibilitaria a correção automática de testes dos estudantes.

# Anexo A

# Manual prático de utilização do FEUPSim

Neste anexo é apresentado o guião prático criado para iniciação do FEUPSim. Este está escrito numa linguagem simples e contem informações essenciais ao uso do FEUPSim.

# 1 - Janelas que compõe o simulador:

## 1.1 – Janela principal:

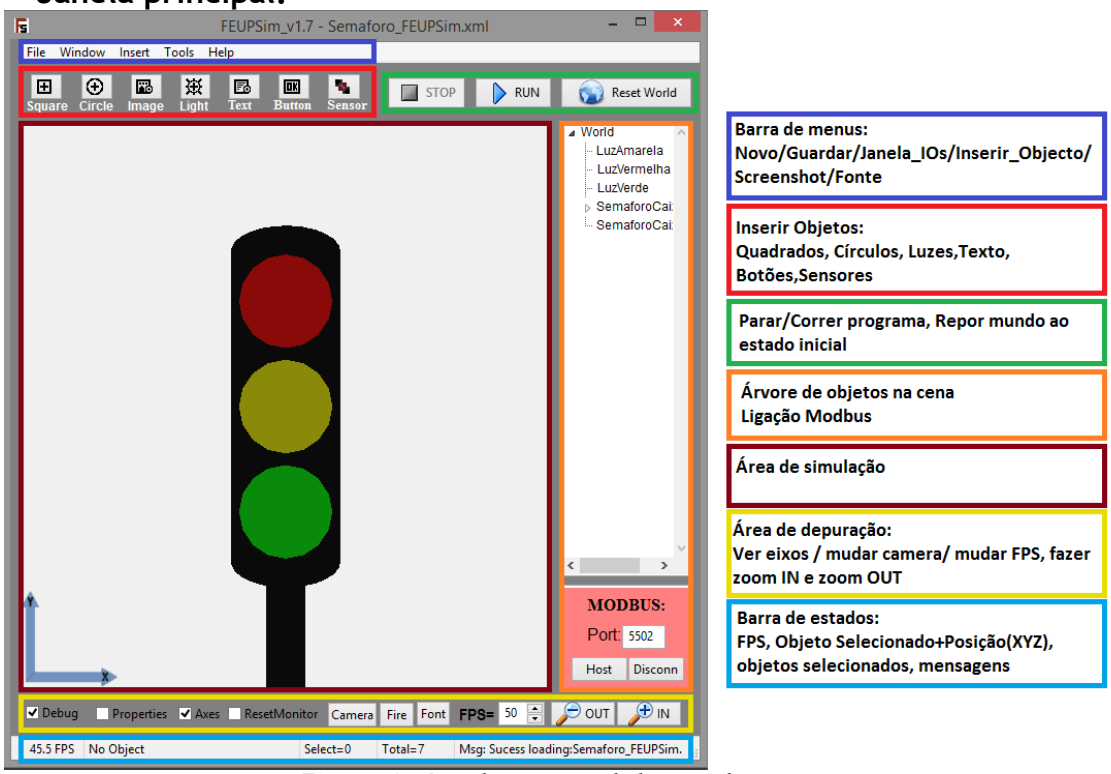



**Figura 1- Janela principal de simulação**

## 1.2 – Janela de I/O e Janela de Propriedades:

Na janela de I/O (esq na Fig) é possível examinar o estado das entradas e saídas do sistema e forçá-las para verificar o correto funcionamento antes de ligar o simulador ao FEUPAutom.

Na janela de Propriedades (dir na Fig) é possível verificar e modificar em tempo real as propriedades que são comuns a todos os objetos.

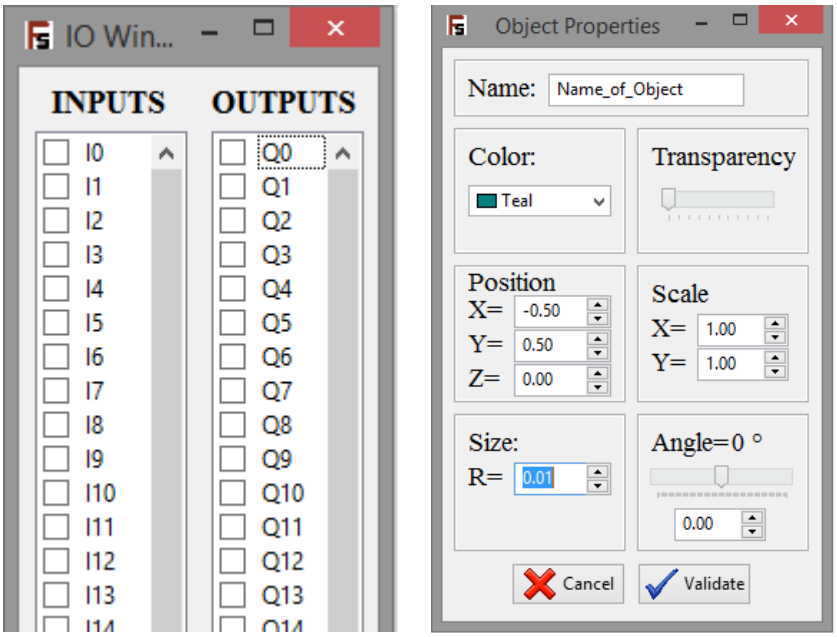



**Figura 2 - Janela de IO (esq) e de Propriedades (dir)**

# 2 – Execução e Simulação:

Existem dois modos de execução a **edição** na qual é possível criar e desenhar o simulador **e a simulação** em si onde o utilizador pode verificar o funcionamento do simulador criado. No entanto, antes de esclarecer os modos de edição e simulação é importante explicar algumas noções e conceitos inerentes ao simulador criado.

## 2.1 – Noções importantes

### 2.1.1 – Cena

A Cena é o local onde os objetos são criados e é aqui que ocorre toda a simulação. Todos os objetos são criados na posição (0,0,0) podendo ser facilmente deslocados e colocados numa posição pretendida clicando com o botão esquerdo do rato e arrastando-os.

### 2.1.2 - Filhos

A noção de filhos aqui apresentada está relacionada com o que em programação se chama "child" tendo como significado a inferioridade hierárquica de um objeto em relação a outro. Na aplicação criada o significado desta noção deve ser bem compreendido pelo utilizador, pois é uma ferramenta poderosa que pode auxiliar na criação de simulações.

Este conceito é bastante importante uma vez que: A posição do objeto filho é relativa em relação à do objeto pai; Movendo o objeto pai o objeto filho é também movido; Um objeto filho roda se o objeto pai rodar;

### 2.1.3 – Comunicação Modbus com FEUPAutom

A comunicação entre o FEUPAutom e o FEUPSim é realizada através de Modbus TCP que é um protocolo muito robusto e que para o utilizador é de muito fácil ativação:

1. No FEUPAutom deve:
   a. Clicar na caixa de seleção de modbus;
   b. Colocar a porta e o nome do anfitrião;
2. No FEUPSim deve:
   a. Clicar em Host e quando o FEUPAutom começar a correr a caixa ficará verde;

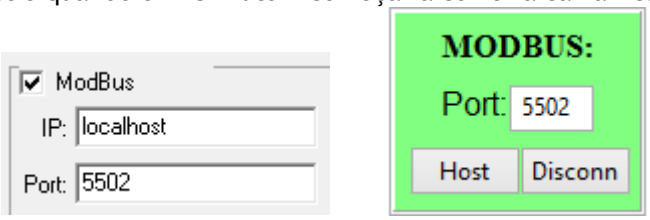



**Figura 3 - Modbus do FEUPAutom (à esq) e FEUPSim (à dir)**

### 2.1.5 – Gravação e Carregamento de ficheiros

Para gravação e carregamento de ficheiros foi utilizada a linguagem XML pela facilidade de edição e pela sua transversalidade entre plataformas.

É necessário ter cuido com as mudanças efetuadas diretamente no XML do FEUPSim uma vez que este possui mecanismos de deteção de correpção de ficheiros por parte do utilizador.

## 2.2 – Funções

### 2.2.1 – Criação de objetos

Quer através da barra de inserção de objetos quer pelas opções do menu inserir, é possível introduzir novos objetos na cena. Os objetos são:

**Quadrados:** Objetos retangulares que permitem criar formas diversas se por exemplo se juntar vários quadrados filhos com diferentes rotações e tamanhos. É também possível associar uma imagem a face do cubo.

As suas funções incluem ter uma velocidade linear e angular. Tem para isso uma forma de criar novas 'linhas de bits' que permitem que o objeto tenha diferentes velocidades consoante o bit associado que esteja a TRUE. Não se pode ter mais que um bit associado a TRUE, pois para a simulação e dá um erro.

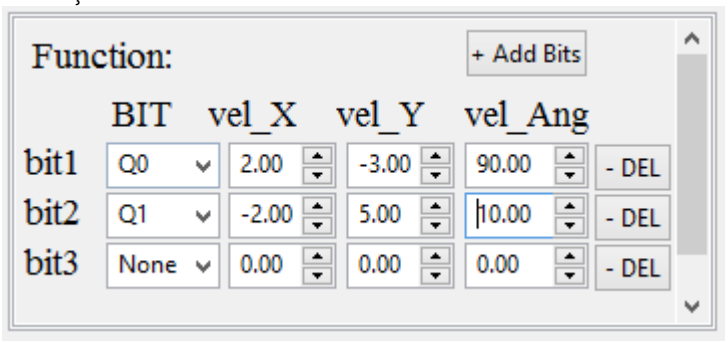



**Figura 4 - Função associada a um objeto Quadrado**

**Círculos:** Objetos circulares ou elípticos usando a propriedade scale.

Em termos de funções estes objetos têm exatamente as mesmas que os quadrados, várias velocidades lineares e angulares.

**Imagens:** As imagens existem por forma a facilmente existirem animações com troca de imagens, um tapete com "movimento", um elevador a abrir a porta, etc…

A sua função consiste então em ter vários bits associados que consoante existe um RE (Rising Edge) mudam para a imagem que o utilizador definiu. Existe várias imagens para testes na pasta 'zFEUPSim-PImages'.


**Figura 5 - Função associada a uma Imagem**

**Luzes:** O objeto luzes serve tanto para indicações luminosas como sinalização de estados.

A função associada as luzes é então a troca de uma cor por outra quando existe um RE ou um FE da variável associada, tal como é passível de ver na figura 13.

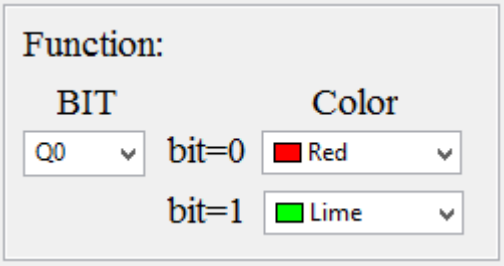



**Figura 6 - Função associada às Luzes**

**Texto:** O texto serve apenas como referência e não possui qualquer função associada.
É necessário ter cuidado uma vez que o texto é sempre 'pickable' mesmo clicando num lugar onde não existe qualquer objeto, isto é, num lugar onde está vazio. Se existir um objeto do tipo texto o simulador assume o texto como objeto corrente. (para anular este efeito é por a propriedade 'pickable' a FALSE.)

**Botões:** Tal como um botão em qualquer interface é possível neste simulador usar objetos do tipo botão que têm como função enviar sinais de entrada para o FEUPAutom.
Existem dois tipos de botões:
- Click: O valor TRUE é mantido enquanto o utilizador clica com o botão esquerdo durante a simulação, e quando deixa de clicar o bit associado fica a FALSE.
- Switch: O valor do bit associado é trocado a cada clique com o botão esquerdo por parte do utilizador.

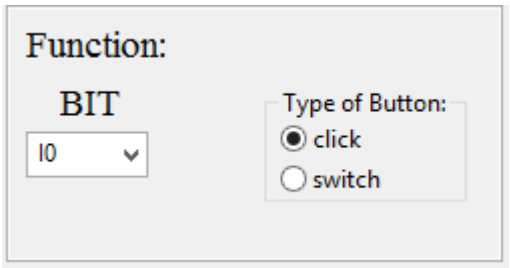



**Figura 7 - Função associada aos Botões**

**Sensores:** Este tipo de objeto permite tal como os botões enviar sinais de entrada para o FEUPAutom, mas em vez de funcionar através de cliques, funciona como um sensor e existe em duas formas:
- presence: Sensor de presença em que, enquanto colide com outro objeto (que tenha o modo de colisão ligado) mantêm o valor do bit associado a TRUE e quando deixa de colidir coloca-o a FALSE.
- stop: Modo de segurança, quando algum objeto colide com um sensor do tipo stop a simulação para e é dado um alerta de erro e o sensor fica a "arder", sendo necessário fazer o 'fix world'.

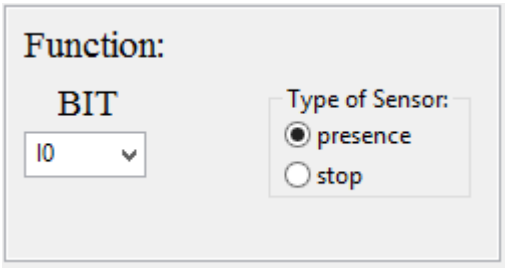



**Figura 8 - Função associada ao Sensor**

### 2.2.4 – PopUp-Menu

Clicando com o botão direito em cima de um objeto é possível chamar um *PopUp-Menu* que realiza determinadas ações consoante o objeto clicado, dentro delas, e tal como pode ser visualizado na figura seguinte:

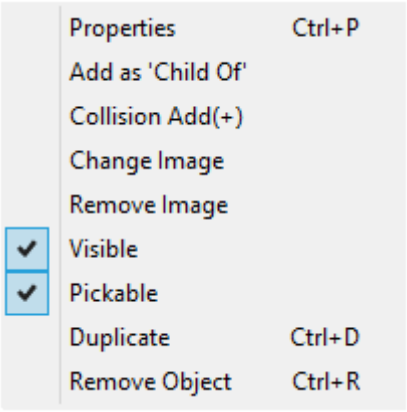



**Figura 9 - PopUp-Menu**

**Properties - Propriedades**
Chamar a janela propriedades do objecto (ver mais sobre propriedades me 2.1.3).

**Add as 'Child Of' – Adicionar objeto como filho de**
Adicionar o presente objeto como filho de outro.

**Collision Add(+) – Adicionar a colisões**
Apenas funcionando para objetos do tipo "Square" e "Circle" esta funcionalidade permite a tais objetos colidirem ou não com os sensores.

**Change Image – Mudar Imagem**
Apenas funcionando para objetos do tipo "Square", "Button" e "Image" esta funcionalidade permite a tais objetos ficarem com uma nova imagem na sua face.

**Visible – Visível**
Permite tornar um objeto visível ou invisível, funcionalidade boa se queremos ter objetos que rodem em torno de eixo invisível.

**Pickable – Objeto pode se mover**
Permite que se selecionado um objeto possa ser movido ou não.

**Duplicate – Duplicar**
Duplica o objeto com todas as suas propriedades.

**Remove Object – Remover Objeto**
Remove o objeto da cena.

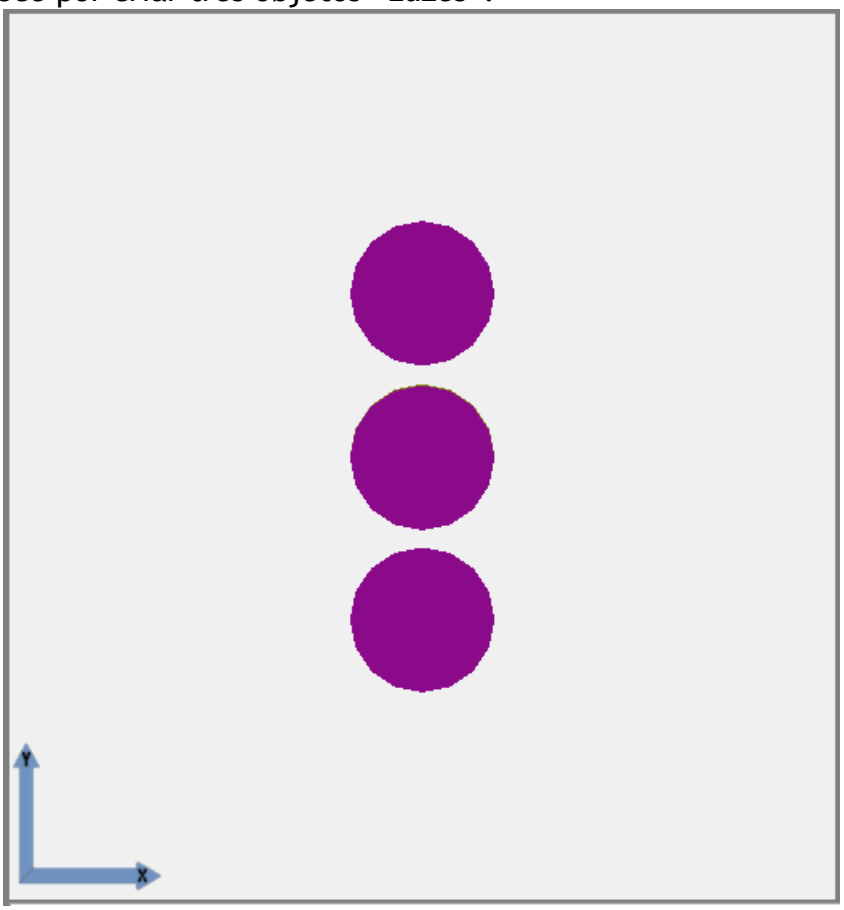
# Anexo B

# Guião de exemplos FEUPSim

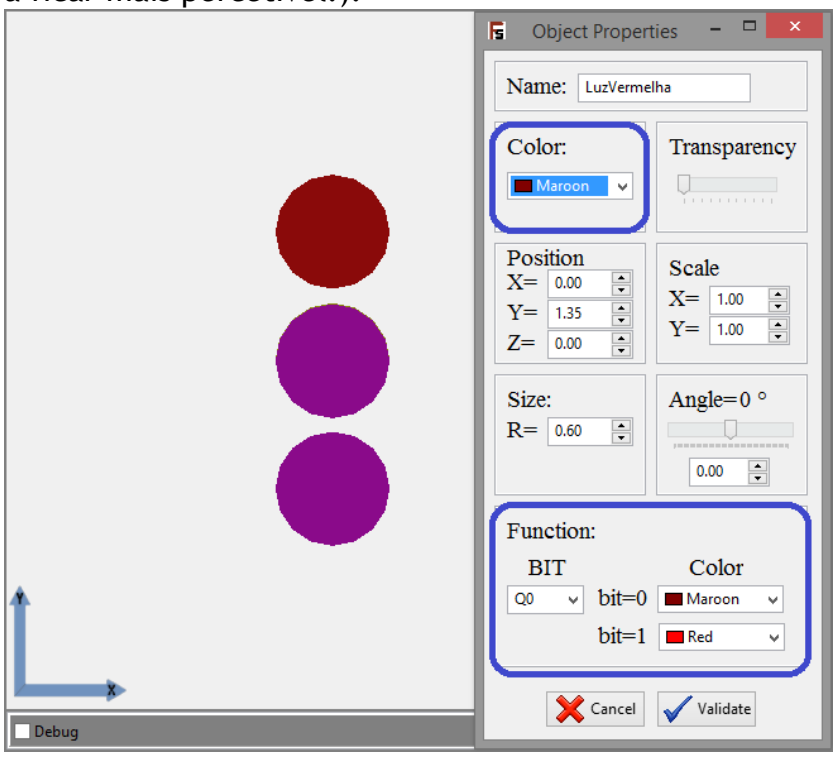
O presente anexo existe por forma aos utilizadores do FEUPSim terem exemplos práticos da aplicação desta ferramenta. Escrito na forma de guião, este documento descreve todos os passo necessários para a criação de dois exemplos de simulações passíveis de criar com o FEUPSim.

# 1 – Semáforos de trânsito

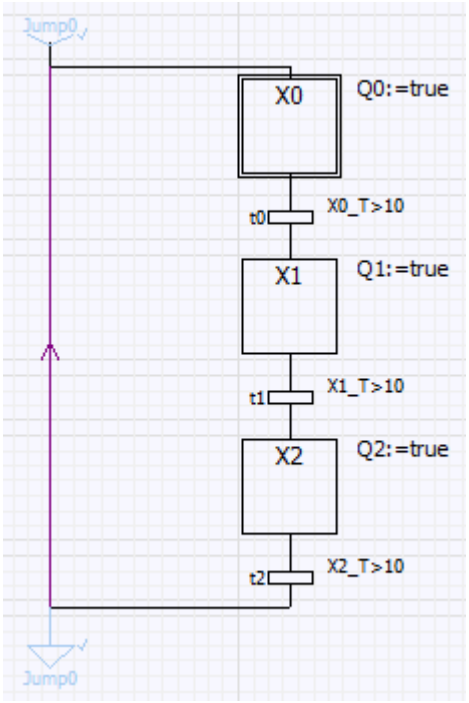
1.1 – Comece por criar três objetos “Luzes”:



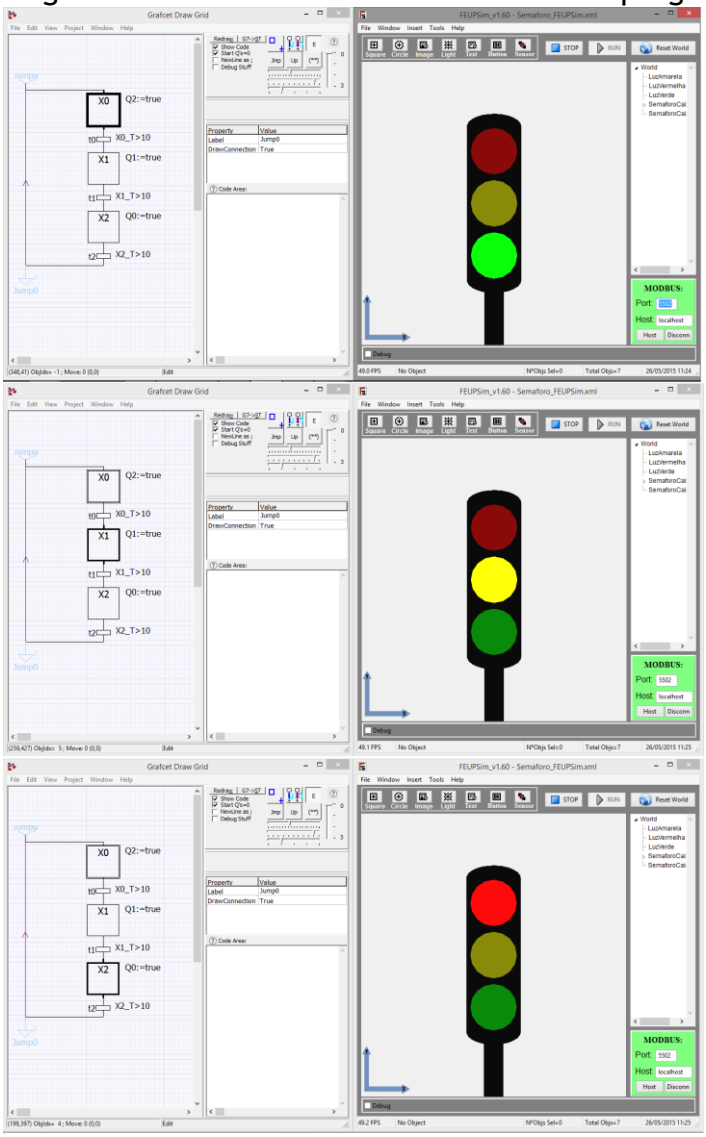

1.2 – Mude as cores associadas ao bit de cada uma das luzes de acordo com a ordem de um semáforo (neste exemplo iremos ainda mudar a cor frontal do objeto para ficar mais percetível.):



1.3 – Escreva o controlo do semáforo em FEUPAutom:

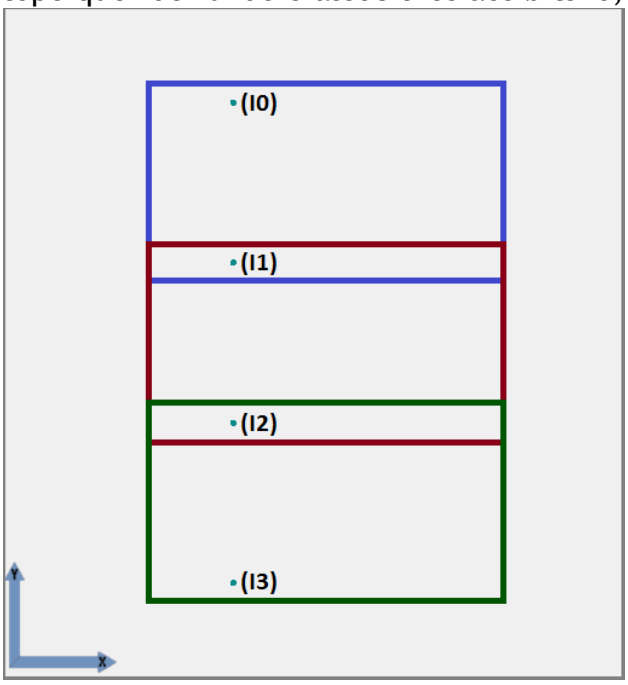



1.4 – Por último ligue o FEUPSim ao FEUPAutom e corra o programa:

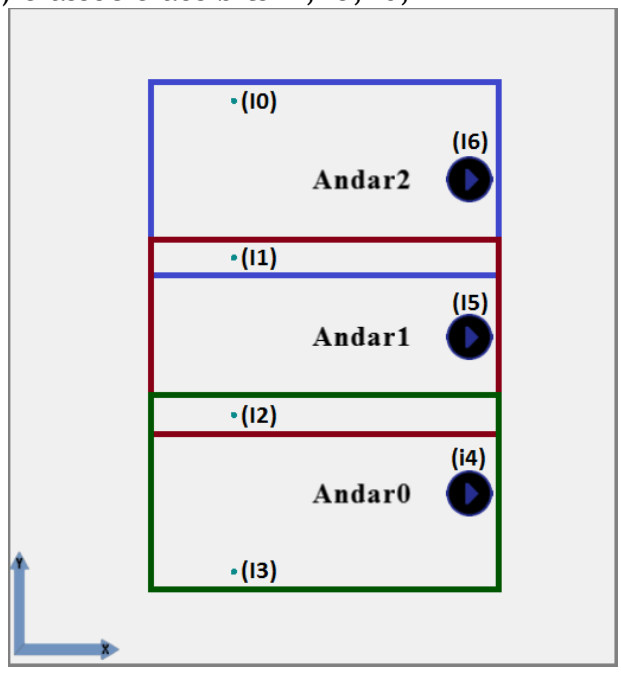


## 2 - Elevador de serviço 3 pisos:

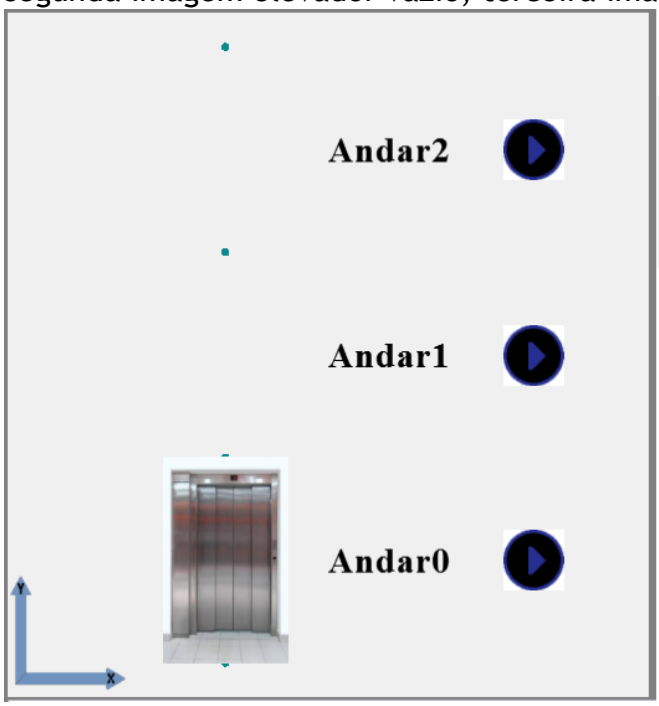
2.1 – Comece por criar quatro objetos sensores que serão os sensores dos andares, quer de topo quer de fundo e associe-os aos bits I0, I1, I2, I3;

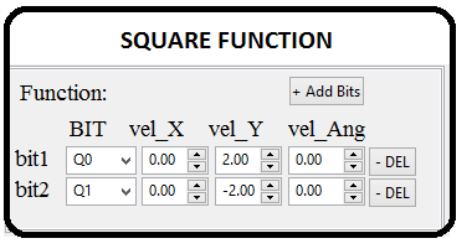


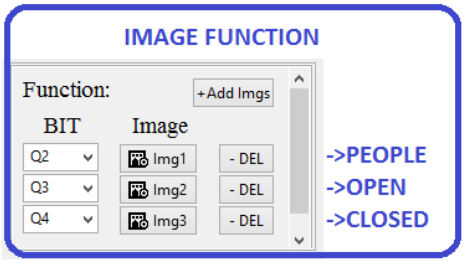

2.2 – De seguida crie um texto indicativo do andar e um botão por andar (andar0, andar1…) e associe aos bits I4, I5, I6;



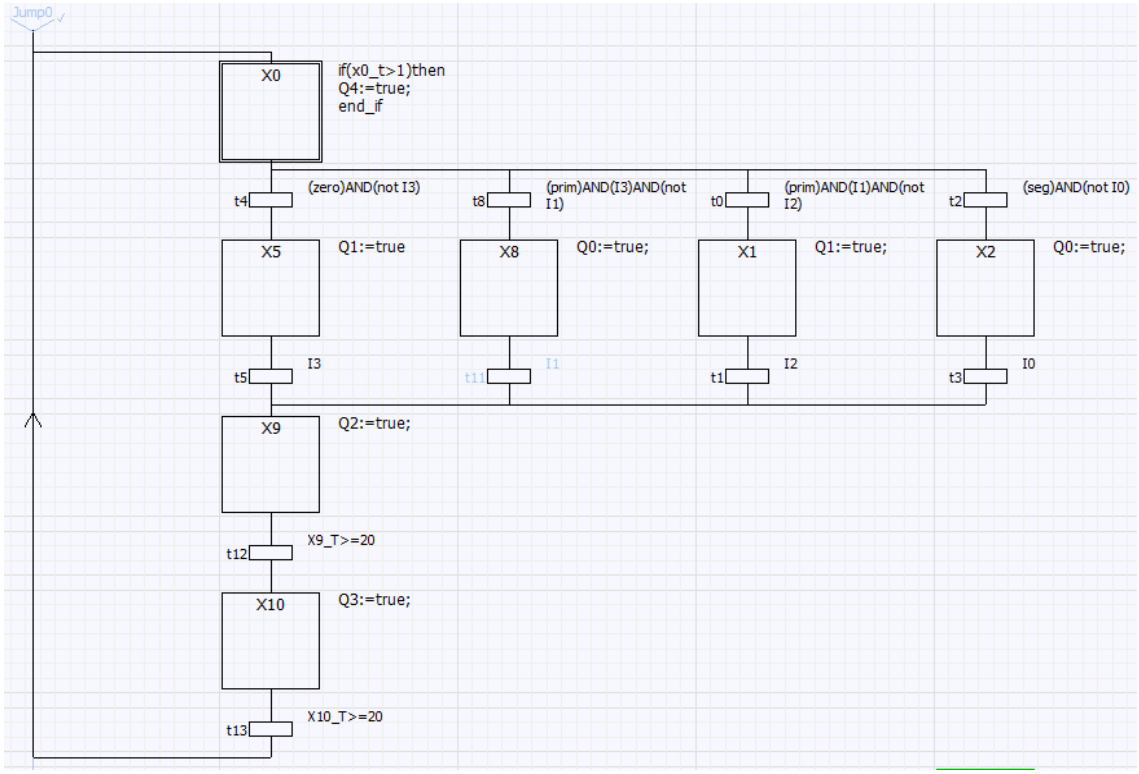
2.3 – De seguida crie um ‘Quadrado’ com as especificações a preto da figura. Acrescente uma ‘Imagem’ ao simulador que seja filha do quadrado anterior com Z=0.6 e com as especificações a azul, primeira imagem a pessoas a sair, segunda imagem elevador vazio, terceira imagem elevador fechado.



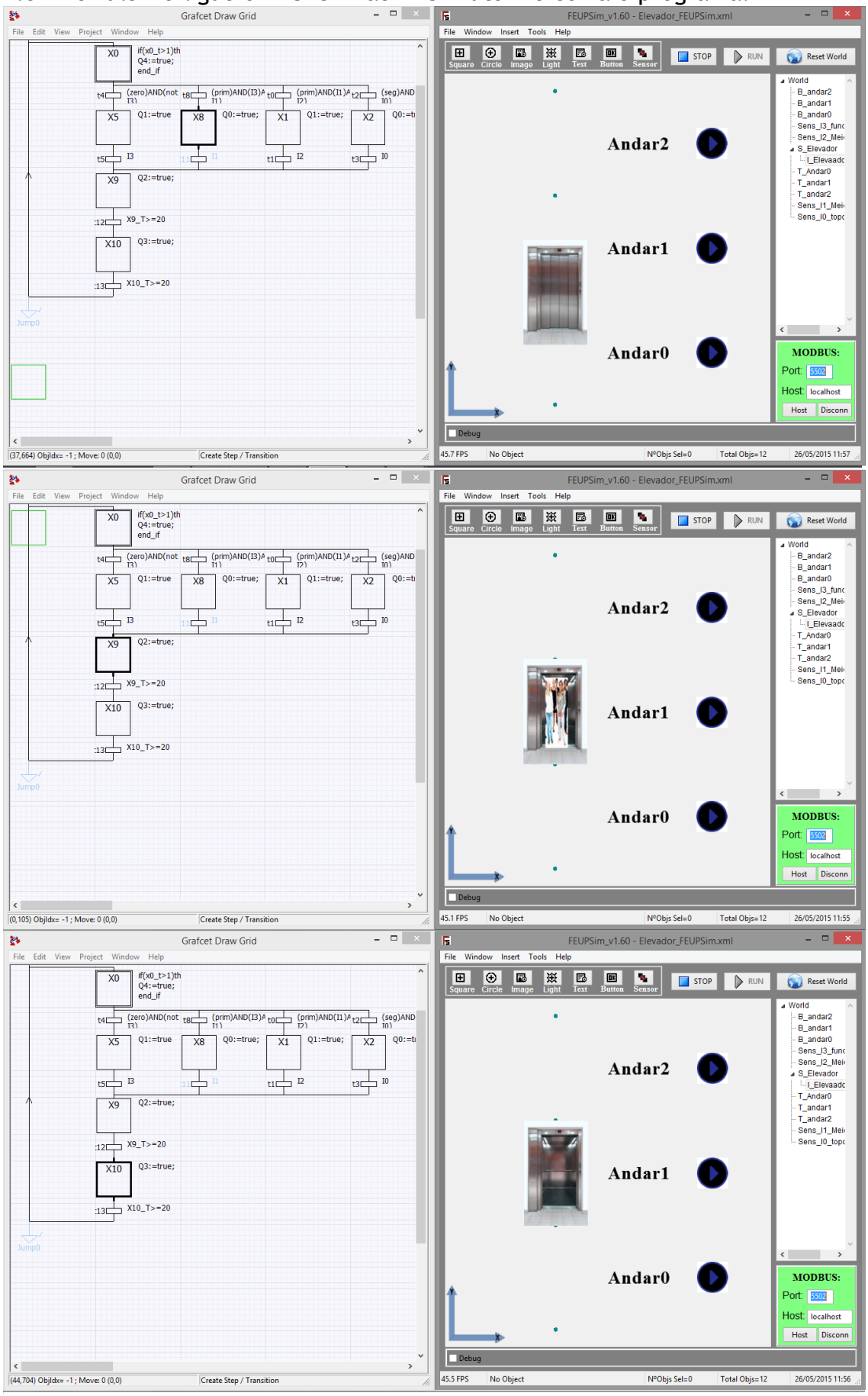




2.4 – Escreva o controlo do elevador em FEUPAutom:



## 2.5 – Por último ligue o FEUPSim ao FEUPAutom e corra o programa:



# Anexo C

# Artigo EDULEARN15

No âmbito da presente Dissertação foi realizado um artigo para a sétima conferência internacional em educação e novas tecnologias de aprendizagem - EDULEARN15. A conferência terá lugar em Barcelona de 6 a 8 de Julho de 2015 e tem como foco o ensino, tanto a nível de métodos desenvolvidos e projetos educacionais como novas tecnologias aplicadas à educação e investigação.

O artigo foi escrito com a orientação e colaboração do Professor Doutor Armando Sousa e do Professor Doutor Paulo Costa. Com o título "*Development of a student-centred tool that promotes deep learning in the technical area of control of discrete event systems*", este artigo explica o interesse da criação de ferramentas específicas para o ensino. Através da demonstração das funcionalidades do FEUPAutom e da sua influência no ensino da UC Sistemas e Automação da FEUP, é demonstrado que é possível envolver ativamente os alunos na sua própria aprendizagem.

O artigo foi parcialmente apoiado pelo FEDER - Fundo Europeu de Desenvolvimento Regional através do Programa COMPETE, por fundos nacionais da FCT - Fundação para a Ciência e Tecnologia, dentro do projeto «FCOMP- 01-0124-FEDER-037281» e ainda pelo INESC TEC. A todas estas entidades deixo aqui uma palavra de apreço.

# DEVELOPMENT OF A STUDENT-CENTRED TOOL THAT PROMOTES DEEP LEARNING IN THE TECHNICAL AREA OF CONTROL OF DISCRETE EVENT SYSTEMS

**A. Sousa[1], B. Augusto[2], P.Costa[1]**

*[1]INESC TEC - INESC Technology and Science (formerly INESC Porto) and FEUP - Faculty of Engineering, University of Porto (PORTUGAL)*
*[2]FEUP - Faculty of Engineering, University of Porto (PORTUGAL)*
*asousa@fe.up.pt, ee10156@fe.up.pt, paco@fe.up.pt*

## Abstract

The article will present the development of the tool FEUPAutom, used at the Faculty of Engineering of the University of Porto (FEUP) in the automation engineering Technological & Scientific area.

In FEUP, the pressure to deliver well trained engineers is steadily high in the last two decades, thus producing a well-known situation of massification in Higher Education Institutions, namely in engineering degrees. In the school year of 2013/14, the course where the tool was used had about 270 students, despite quite low retention rates. The article also includes a brief characterization of the engineering program, course and expected outcomes in full alignment with the ideas promoted by the EUR-ACE referential for the accreditation of engineering programs and also in strong consonance with the ideas defended by the Bologna process. The course includes lab work and a part of those uses Problem Based Learning (PBL) methodology.

In the last decade, the professors of the mentioned course have tried to limit the usage of real world industrial equipments because of budget concerns, always without hindering the learning process. Adequate simulation tools were sought on the market but not found, mainly because the needs of a full blown engineer are frequently not the same as those of an early engineering student. At that point, the decision was made to develop an in-house tool, adequate for students. Industrial-grade equipment was not totally set aside, only reserved for latter stages and the actual usage strategy allowed the number of equipments to be halved.

The article will go on briefly describing the FEUPAutom tool and new strategies available for lab classes and PBL. As control groups would be unethical, students' quiz data from the two last editions of the course are used to evaluate learning (self-assessed). Grading strategy and coordination with the university's LMS is also addressed. Final grades of the course and satisfaction are also discussed.

The students' assessment is that the FEUPAutom tool is very useful for the learning process and easier to use than the available industrial counterpart.

Continuous improvement efforts have tried to push students to adequate PBL work only possible with the tool, with some results hinting deep learning in the technical area at stake.

Some final thoughts, lessons learned and future work are also present in the article.

Keywords: Innovation, engineering, automation, learning, technology in education.

## 1 INTRODUCTION

In this chapter it will be made a brief synopsis of the faculty, program and course where the tool is used.

### 1.1 Faculty

Faculty of Engineering of the University of Porto (FEUP) is a renowned faculty of Portugal, being among the best by the international standards and European rankings. Its activities include education, research and innovation in the fields of engineering. FEUP tries the best to give the tools to both students, teachers, researchers and all the staff so that it can train, develop and form the best engineers.

In numbers, FEUP community is one of the biggest in Portugal with an total area of 93.918 m2, counting with 8.199 students, 547 Teaching staff and 301 non-academic staff in the 2013/2014 year.

## 1.2 Program

The Integrated Master in Electrical and Computer Engineering is then one of the 9 possible Integrated Masters to take in FEUP. It distinguish himself by giving the students solid and fundamental scientific training in the fields of Electrical and Computer Engineering in the first two years and then in the 3rd year it gives the possibility to choose from 3 majors: Automation, Energy and "Telecommunications, Electronics and Computers" where the students deepen their studies. This implies that although the students specialize in a certain field of Electrical and Computer Engineering the bases and background of all students is the same and very cemented on the scientific knowledge.

The course is accredited under the EUR-ACE system, in accordance to the Bologna treaty. Although an integrated master planned as a 5 year MsC. engineering degree (Bologna second cycle), students are given a mobility certificate of basic engineering sciences that covers the skills recommend by the EUR-ACE consortium, including basic technological awareness.

## 1.3 Course

One of the main courses in the second year of the program is ‘Systems and Automation’ (SA), because it is one of the courses that will influence in the choice of the major. In this course the students are presented with the concept of designing and implementing an Automation System (of limited complexity). At the end of the course, students are to be able to apply discrete events systems control technology to problems of limited complexity, including:

- Design and interpret State Machines, Grafcet, and Petri Nets to model discrete event systems;
- Implement State Machines in Microcontrollers and Programmable Logic Controllers (PLC);
- Use Grafcet/ST to control Automatic Systems of low to medium complexity;
- Design and implement an Event Driven Control System, for problems of medium complexity.

Strategies include theoretical lectures and hands-on approach lab work in a “Learn by doing” set of exercises that lead through the issues at stake in a progressive manner. Additionally, Moodle is used as a Learning Management System (LMS) including formative evaluation after some lab classes and as the main tool to communicate, remove doubts and present contents to the students. Additional strategies include several summative “small” tests instead of a final exam.

Class size is typically around 250 but may vary much among years. As there is a great interest in lab sessions, fails (grade below 10 out of 20) are not very common.

# 2 PROBLEM

As said, this is a course where the students learn how to implement Grafcet in PLC and in FEUP there are only 12 PLCs available to SA. Once the course is given in two separate labs, the teaching staff had to decide what was best for the students given that the cost of buying 12 equal PLCs to the old ones or buying 24 new PLCs was too expensive. The alternatives were to not experience real hardware from industrial reality or too work with siblings labs that would desynchronize in the matters being taught in the classes.

To better solve both problems, letting the students experience real hardware and, at the same time, allowing both labs be synchronized it was necessary to create an alternative for the lab that hadn't PLCs. It was opted to create a softPLC that had to be free, easy to use and portable. This softPLC (FEUPAutom) is now fully operational and with many years of development it is a fully mature tool to aiding in the class of SA and others.

# 3 TOOL - FEUPAUTOM

FEUPAutom [1] is a free softPLC designed in FEUP by Professor Armando Sousa. As said in chapter 2 it had its origin in the need to create a software that would serve as the viable substitute to the expensive PLC's needed to the course.

## 3.1 Features and Application

It main characteristics are: portability, easy to use and program, and very intuitive. This way it can be used in the teaching and in the development of automatic control programs both in the classes an at home. Letting not only the students complete their lab work at home, if they have no time in the classes, but also allowing the students to further deepen their knowledge in the automation field, once there are plenty of exercises they can solve using this software has the main control program.

This software can be programmed in ST or Grafcet. While the first language was created following the definitions of IEC 61131-3 [2], the second is an adaptation of IEC 60848 [3] in order to better suit the needs of the matters being taught.

There is also support for Modbus TCP communication, which translates into huge facility of sending and receiving data, thereby making it possible to connect FEUPAutom with external simulations.

In Figure 1 is shown an example of the development environment of FEUPAutom divided by its components:

• FEUPAutom (black): In this window it's possible to edit some project settings, write ST programs, see their conversion to C and Pascal. It is also here that are shown program error messages and/or messages conveyed by this.

• Grafcet Draw Grid (red): Window that allows to create programs in Grafcet, all design is done on the grid and there is an area of programming where the user can write transitions or actions that must be performed using ST language..

• IO LED (blue): State representation of the Inputs and Outputs of the system. It is also possible to force determined bit to True or False.

• Log (green): Representation of time plotted variables chosen by the user.

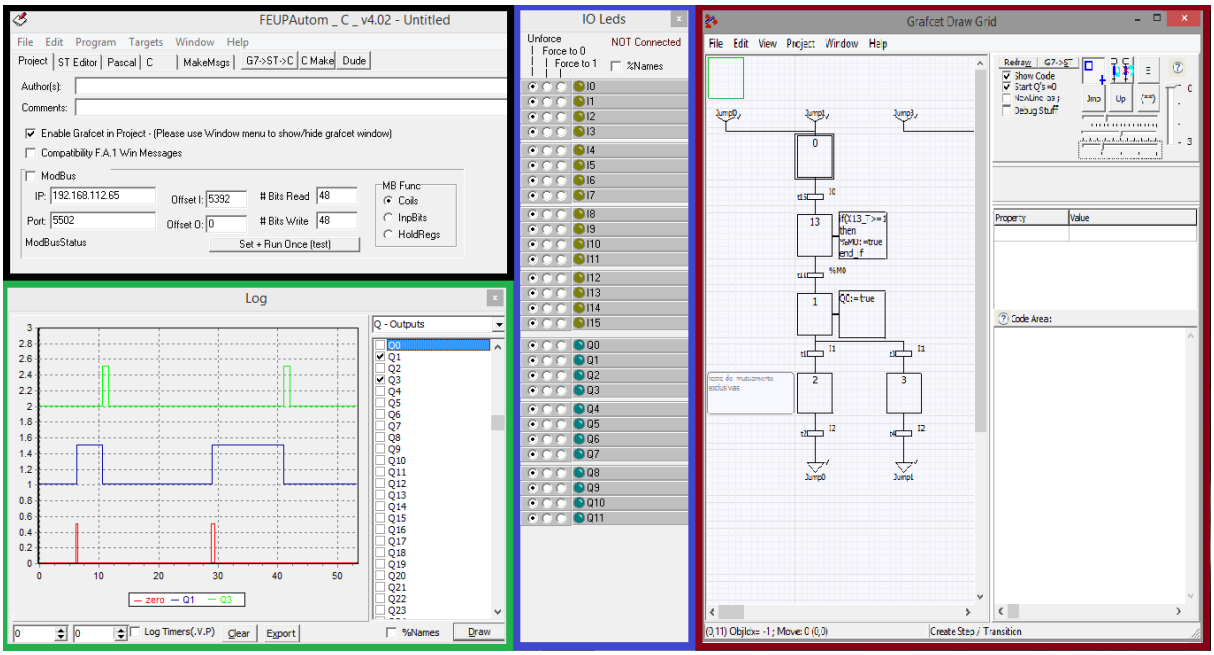



**Figure 1 -Development environment of FEUPAutom**

## 3.2 Debugging

FEUPAutom being a software that is constantly in development, either by new additions made by the develop team or new requirements and suggestions made by the students, it has several debugging tools to improve it usage:

• In Grafcet mode it can convert the Grafcet made to a State Machine, give a translation of the objects drawn to a text format (as seen in Figure 2) so the user can watch if all the objects are correctly created, and by the order they made them, the code area has an auto-complete systems that allows for fast development of algorithms, and there is the possibility to leave comments in each object so the user can write what the object is doing.

• In the main window it's possible to watch the conversion of Grafcet in ST, the ST conversion in Pascal and C language, there is also one window for compilation errors that show the user what the errors are in ST language, because it uses the same names that the Grafcet it is easily to see what is the error in the Grafcet draw.

• The last debugging tool is the Time Logger (green one in figure 1) that is crucial, once it is the perfect way to see what is happening with the system variables as time goes by. It can draw several plots at the same time.

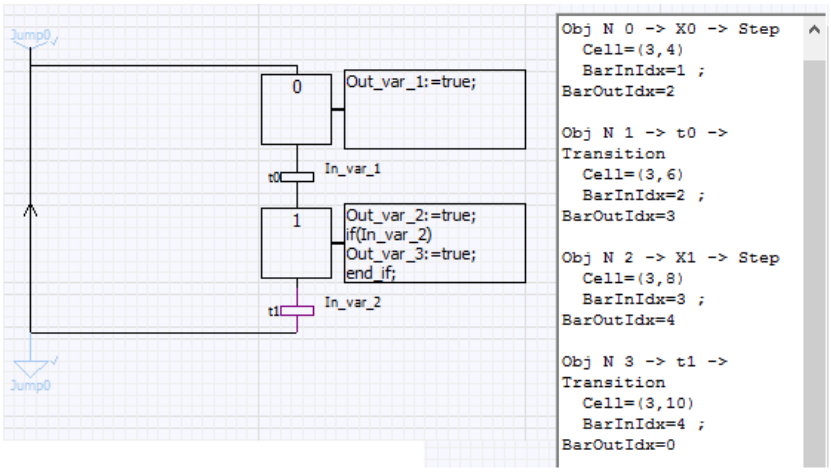



**Figure 2 - Debugging tools**

## 3.3 Simulations

To become a truly innovative tool of discrete events and control teaching, there was a need to develop some working examples so the students could test their control code. For that Professor Paulo Costa, created three working simulations that would allow the students to train the skills learned and help them keep trying to achieve better results. The three simulators are shown in Figure 3.

• Park 3D (green): With this simulator the students are encouraged to develop the control program for a parking lot, at first there is only cars moving in with a limit space of 6 cars, then it becomes more difficult with the introduction of a second barrier for the cars that are moving out.

• Gate 3D (red): In this simulator the objective is to control a garage gate with the benefit of a light that needs to be on if the natural light intensity is lower than a certain level.

• Thunderbird 3D (blue): In the last simulator it is presented the control system of the back lights of a car, where the students need to implement the control of three functions: turn right, turn left, and emergency.

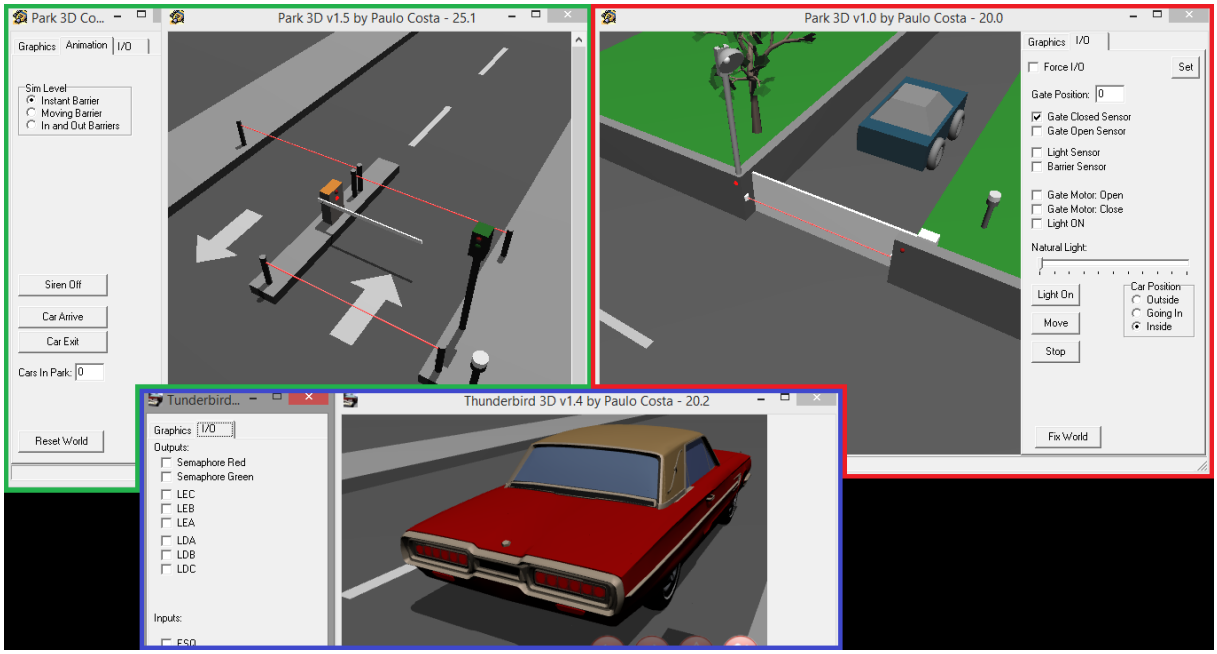



**Figure 3 - FEUPAutom simulators**

## 3.4 Added value to the course

As seen this software is both programmable in ST or Grafcet, what is very good in terms of teaching because it enhances the ability to create Problem-Based Learning (PBL) exercises, leading to better results and letting the students learn much faster than by the usual teaching. This methodology combined with the siblings labs work, allows every class, in each lab, to have 24 students divided 12 groups of two persons maximizing the time that the students are with a teacher in the lab and improving their skills by learning while solving problems.

# 4 RESULTS AND SURVEYS

For this tool to be approved and validated the grades had to be good and it was necessary to know what the students thought about the tool both as a development tool and as compared to the industrial software used, PL7.

## 4.1 Surveys

For better understand the usefulness of the tool created, in the year of 2013/2014 surveys were conducted. All the students answered in lab time to the survey that was composed by 18 questions of interest to the teaching group. Of the 18 questions, 3 relating to FEUPAutom were chosen to present in this article.

The answers ranged from 1 to 5 meaning: 1=Fully Disagree, 2=Disagree, 3=Neutral, 4= Agree, 5=Fully Agree. There were made graphics of the sum of each number and they will be presented next.

As it can be seen in Figure 4 the first question was: “FEUPAutom was useful to my learning?”. As the results show there is a good approval of FEUPAutom by the students and they believe it help them learn the matters being discussed (ST and Grafcet), leading to better results in the course.

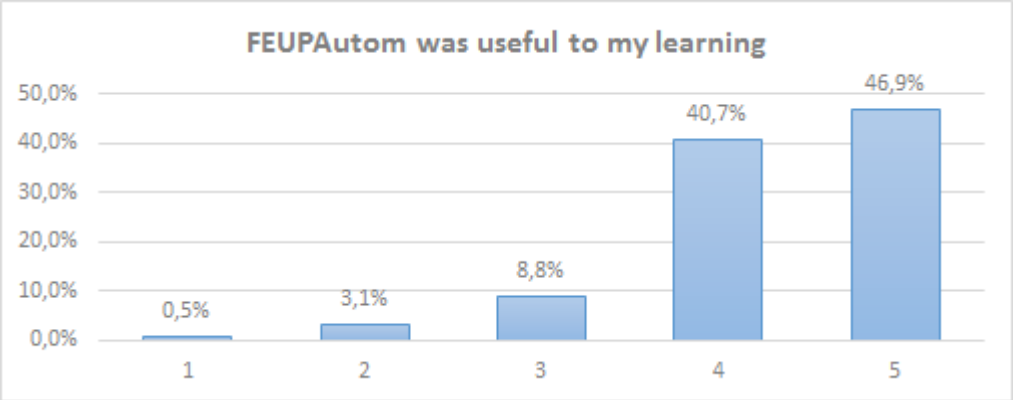



**Figure 4 - Survey question: "Was FEUPAutom useful to my learning?"**

When asked what interface, FEUPAutom or PL7, was better for learning and programming the majority of the students agreed that FEUPAutom interface was better than its industrial counterpart PL7 (Figure 5). As said in chapter 2 the course is given in too sibling labs and while half the class work with PL7 the other half is working with FEUPAutom in the same problem, after 2 weeks they change and the process repeats. So this question is an important one because it allows to take the conclusion that there is no need for PL7 in the future if FEUPAutom starts compiling for PLCs.

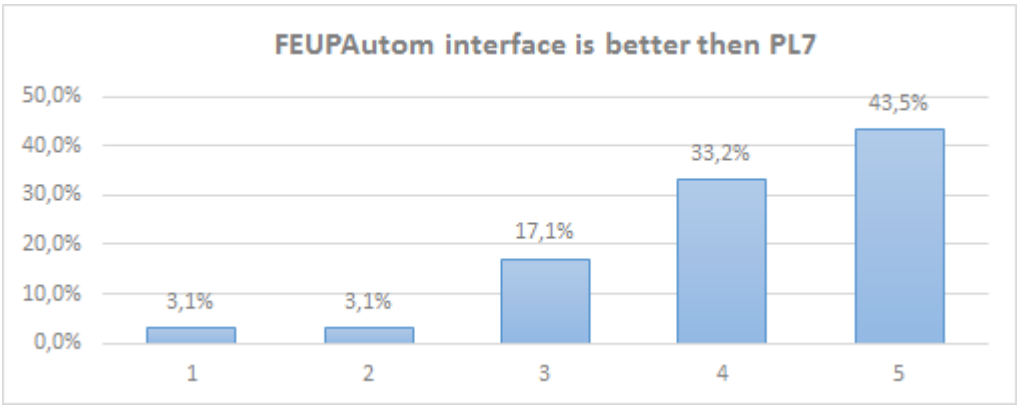



**Figure 5 - Survey question: "FEUPAutom interface is better than PL7?"**

At last it was asked if they enjoyed seeing their program interact with a simulator, and as seen in Figure 6 more than half the students (62.4% representing 121 of 194 students) fully agree that the simulators are a great way to learn, because it allows them to interact and see what are the real consequences of their programming.

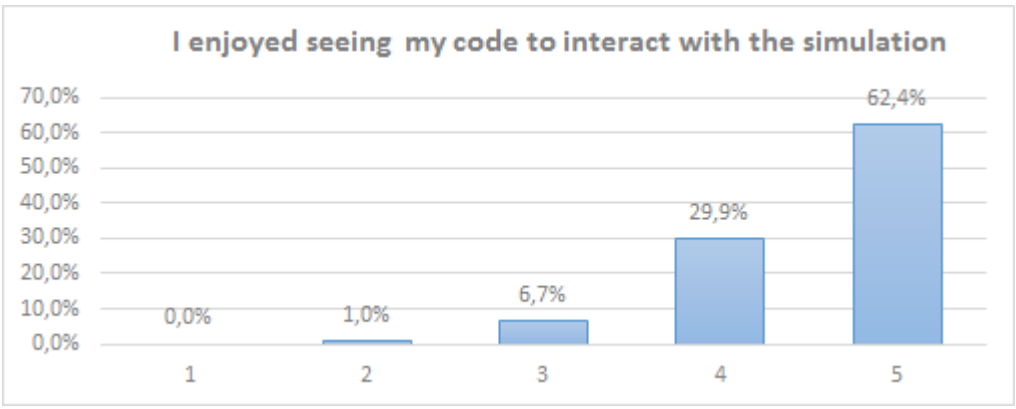



**Figure 6 -Survey question "I enjoyed seeing my code to interact with the simulation"**

In the end of the survey it was asked for the opinions of the students about the strengths and weaknesses of the course and FEUPAutom, some of the answers were:

**Strengths:**

• “Simplicity of programming FEUPAutom and very useful simulations for exercises.”

• “I think it is on the Grafcet that FEUPAutom stands out. It really is much easier to work with FEUPAutom Grafcet than with the PL7 one.”

• “The FEUPAutom is an integral part of the approach ‘student friendly’. It is easy to use and you can learn a lot without resorting to laboratories”

• “The part of C and ST language of the FEUPAutom with temporal traces, I/O's, customizable simulations and temporal strokes helped a lot.”

• “Good program that takes up little space, allowing the student to continue is study at home with a certain ease.”

**Weaknesses:**

• “FEUPAutom only run in Windows OS”.

• “Wish it had more real systems simulations and above it, that was possible to simulate a system designed by me.”

• “One teacher to so many stands during practices classes is insufficient.”

## 5 CONCLUSIONS

The creation of this tool as an alternative to the costly acquisition of new hardware and commercial software proves the strengths of creating specific software for classes. The work involved is justified by the inadequacy of the debugging features that further enhance the learning of the students. This additional learning is even greater because the sense of proximity allows students to try to understand how the software works and maybe even be close or a part of the development of new / improved features, thus involving the students in their own learning processes.

Course grades are regularly not low (average of last 3 editions is around 14 out of 20), thus demonstrating the interest the hand-on approach of the lab classes and the usage of the mentioned FEUPAutom software. Students’ perception of the interest of using the software tailored for their needs is clear by quiz answers where 91 out of 194 students fully agree that the tool is useful for their learning, even more so when compared with the other commercial tool used (that is admittedly out of date).

The FEUPAutom software tool is freely available for download and student can use it at home, in other courses and is even available to other schools and can control simulated or real environments. Such advantages are still to be even further explored.

## 6 FUTURE WORK

The FEUPAutom software was tested under Linux and OS X via the Wine tool but some effort is being made to make it truly cross platform and only using open-source software in it development.

As for features the main objectives are to implement ‘Step by Step’ control so that the students have one more powerful tool to debug their code and algorithms.

Work is being done in order to complement FEUPAutom by allowing the students to create their own simulations such as semaphore, elevator, wind shield wipers… This new software should be designed to be truly portable, easy to use, and generic so that the students could create a great variety of simulation from their own imagination.

## 7 ACKNOWLEDGEMENTS

This work is partially financed by the ERDF – European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project «FCOMP-01-0124-FEDER-037281»

## 8 REFERENCES

[1] Armando Jorge Miranda de Sousa. FEUPAutom. URL: http://paginas.fe.up.pt/~asousa/wiki/doku.php?id=proj:feupautom

[2] International Electrotechnical Commission. IEC 61131-3 Programmable controllers - Part 3: Programming languages. 2nd edition, 2003.

[3] International Electrotechnical Commission. IEC 60848 Ed. 2 Specification language GRAFCET for sequential function charts. 2002.

# Bibliografia


[1] D. Hebert, "Software: Simulation Saves Time and Money," 2014. [Online]. Disponível: http://www.controldesign.com/articles/2015/simulation-saves-time-and-money [Acedido a: 2015-02-10]

[2] P. Costa, "paco/Wiki | Main / SimTwo," 2015. [Online]. Disponível: http://paginas.fe.up.pt/~paco/wiki/index.php?n=Main.SimTwo [Acedido a: 2015-02-01]

[3] P. Costa, J. Gonçalves, J. Lima, e P. Malheiros, "Simtwo realistic simulator: A tool for the development and validation of robot software," *Theory and Applications of Mathematics & Computer Science*, vol. 1, pp. 17–33, 2011.

[4] M. Lischke, E. Grange, e C. Ulrich, "GLSceneSite: NEWS," 2011. [Online]. Disponível: http://glscene.sourceforge.net/wikka/HomePage [Acedido a: 2015-01-06]

[5] Realgames, "Simulation Software for Automation Training - Real Games." [Online]. Disponível: http://www.realgames.pt/factory-io/ [Acedido a: 2015-02-01]

[6] A. del Pozo, J. Escano, e C. Bordons, "Simulator for control and automation using an interactive and configurable 3D virtual environment," *SICE Annual Conference (SICE), 2012 Proceedings of*, pp. 2268–2273, 2012.

[7] Realgames, "Factory I/O - Parts Essentials," 2014. [Online]. Disponível: http://www.realgames.pt/downloads/factoryio/software_files/factoryio_parts_essentials.pdf

[8] Realgames, "Fatory I/O - User Guide," 2014. [Online]. Disponível: http://www.realgames.pt/downloads/factoryio/software_files/factoryio_user_guide_en.pdf

[9] A. Daneels e W. Salter, "What Is Scada?" *Access*, pp. 339–343, 1999.

[10] S. Boyer, "SCADA: Supervisory Control and Data Acquisition," em *Scada: Supervisory Control And Data Acquisition*. ISA, 1999, pp. 1–24.

[11] Citect Corporation, "CitectSCADA."

[12] Sielco Sistemi, "SCADA Software | HMI Software | Supervisory Software for Data Acquisition - Remote Control." [Online]. Disponível: http://www.sielcosistemi.com/en/products/winlog_scada_hmi/index.html [Acedido a: 2015-02-15]

[13] Ecava, "IntegraXor HMI/SCADA System • Web SCADA with fully functional & free SCADA development tools." [Online]. Disponível: http://www.integraxor.com/ [Acedido a: 2015-02-14]

[14] Bosch Rexroth Group |, "PC vs. PLC: key factors in comparing control options," 2011.

[15] A. Sousa, "FEUPAutom," 2014. [Online]. Disponível: http://paginas.fe.up.pt/~asousa/wiki/doku.php?id=proj:feupautom [Acedido a: 2015-01-30]

[16] Beremiz, "Home page of Beremiz." [Online]. Disponível: http://www.beremiz.org/ [Acedido a: 2015-01-15]

[17] ICS Triplex ISaGRAF, "ICS Triplex ISaGRAF Inc. - leading IEC 61131 and IEC 61499 software." [Online]. Disponível: http://www.isagraf.com/index.htm [Acedido a: 2015-02-16]

[18] Omron, "Machine Automation Controllers | Software | Sysmac Studio." [Online]. Disponível: http://industrial.omron.eu/en/products/catalogue/motion_and_drives/machine_automation_controllers/software/sysmac_studio/default.html [Acedido a: 2015-01-15]

[19] International Electrotechnical Commission, *IEC 60848 Ed. 2 Specification language GRAFCET for sequential function charts*, 2002.

[20] International Electrotechnical Commission, *Programmable controllers - Part 3: Programming languages*, 2nd ed., 2003.

[21] K.-H. John e M. Tiegelkamp, *IEC 61131–3: Programming Industrial Automation Systems: Concepts And Programming Languages, Requirements for Programming Systems, AIDS to Decision-making Tools*, 2001.

[22] J. Provost, J. Roussel, e J. Faure, "A formal semantics for Grafcet specifications," *IEEE International Conference on Automation Science and Engineering*, pp. 488–494, 2011.

[23] Lazarus and Free Pascal Team, "Lazarus Homepage," 2015. [Online]. Disponível: http://www.lazarus.freepascal.org/ [Acedido a: 2015-02-16]

[24] Free Pascal Team, "Free Pascal - Advanced open source Pascal compiler for Pascal and Object Pascal," 2010. [Online]. Disponível: http://www.freepascal.org/ [Acedido a: 2015-06-12]

[25] M. Lischke, E. Grange, e C. Ulrich, "GLScene - Gallery - Users," 2011. [Online]. Disponível: http://glscene.sourceforge.net/oldsite/gallery_users.htm [Acedido a: 2015-06-12]

[26] Modbus Organization, "Modbus," p. 2015. [Online]. Disponível: http://www.modbus.org/ [Acedido a: 2015-04-06]

[27] Modbus-IDA, *Modbus Messaging on Tcp / Ip Implementation Guide*. Modbus Organization, 2006.

[28] W3Schools, "XML," 2015. [Online]. Disponível: http://www.w3schools.com/xml/ [Acedido a: 2015-06-12]

[29] D. Repici e D. Burton, "CSV Comma Separated Value File Format," 2010. [Online]. Disponível: http://creativyst.com/Doc/Articles/CSV/CSV01.htm [Acedido a: 2015-06-26]

[30] J. Rubio, "Simulador de escenarios robóticos con capacidad multirobot : Memoria," Projeto Fim de Carreira, Universidad de Zaragoza, 2012.

[31] M. Cantu, *Mastering Delphi 7*. Sybex, 2003.

[32] M. A. Azeem, *Start programming using Object Pascal*, P. Anderson e J. Hackney, Eds., 2012.