

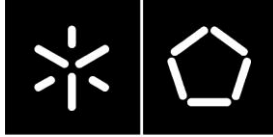


Universidade do Minho
Escola de Engenharia

Nuno Miguel Milhases da Silva

Suporte à Interoperabilidade entre o
Automation Studio e Sistemas SCADA:
Tradução de sinópticos de XAML para SVG

Outubro de 2013



Universidade do Minho
Escola de Engenharia

Nuno Miguel Milhases da Silva

Suporte à Interoperabilidade entre o
Automation Studio e Sistemas SCADA:
Tradução de sinópticos de XAML para SVG

Tese de Mestrado
Mestrado em Informática

Trabalho efectuado sob a orientação do:
Professor José Creissac Campos

Outubro de 2013

DECLARAÇÃO

Nome: Nuno Miguel Milhases da Silva

Endereço electrónico: pg13879@alunos.uminho.pt Telefone: 93 350 91 63

Número do Bilhete de Identidade: 11052786

Título dissertação

Suporte à Interoperabilidade entre o *Automation Studio* e Sistemas SCADA:

Tradução de sinópticos de XAML para SVG

Orientador: José Creissac Campos

Ano de conclusão: 2013

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:

Mestrado de Informática

Declaro que concedo à Universidade do Minho e aos seus agentes uma licença não-exclusiva para arquivar e tornar acessível, nomeadamente através do seu repositório institucional, nas condições abaixo indicadas, a minha tese ou dissertação, no todo ou em parte, em suporte digital.

Declaro que autorizo a Universidade do Minho a arquivar mais de uma cópia da tese ou dissertação e a, sem alterar o seu conteúdo, converter a tese ou dissertação entregue, para qualquer formato de ficheiro, meio ou suporte, para efeitos de preservação e acesso.

Retenho todos os direitos de autor relativos à tese ou dissertação, e o direito de a usar em trabalhos futuros (como artigos ou livros).

Concordo que a minha tese ou dissertação seja colocada no repositório da Universidade do Minho com o seguinte estatuto (assinale um):

Disponibilização do trabalho de acordo com o **Despacho RT-98/2010 c)** (embargo 3 anos)

Universidade do Minho, ___/___/_____

Assinatura: _____

Agradecimentos

Para o desenvolvimento desta tese de Mestrado foram vários os passos até que fosse possível começar a escrevê-la. O projecto que deu origem a esta monografia foi exigente com imensas etapas que ultrapassei, sempre seguindo os meus valores morais e éticos, e posso dizer que hoje estou mais preparado para enfrentar os problemas que surgem. Por isso, gostaria de reconhecer o mérito que algumas pessoas tiveram para que eu conseguisse atingir esta maturidade. Assim, quero agradecer

Ao professor e orientador José Creissac Campos pela sugestão do tema, pelo incentivo à criação de projectos inovadores e pelas observações técnicas, sempre pertinentes, durante o desenvolvimento da tese,

Ao meu orientador na EFACEC, engenheiro Rogério Dias Paulo,

À Marta Meira do CCG (Centro de Computação Gráfica) de Guimarães, pelo apoio com bolsa de investigação,

Ao Doutor Daniel Moreira, pelo apoio na elaboração de conteúdos,

Aos engenheiros da EFACEC Paulo Delfim Rodrigues e Cláudio Manuel Silva pela partilha de conhecimentos e ajuda no desenvolvimento do projecto,

À minha mãe Cândida e ao meu irmão André, por estarem sempre presentes de uma forma activa na minha vida, dando-me todo o amor e apoio nesta minha caminhada,

Aos meus amigos e familiares Daniel, Iris, Rafael, Mário, Anabela, Carlos, Cristina, Alexandre, Sofia, Nuno, Filipa, Lea, Diogo, Juliana e ao meu querido afilhado Gabriel, pela motivação e alento neste percurso da minha vida,

Ao meu mentor G.

Resumo

Em 2008 é posto em marcha um projecto de inovação e desenvolvimento denominado por InPact entre a EFACEC, a EDP Distribuição e a Universidade do Minho. O objectivo deste projecto é fornecer um conjunto de ferramentas de engenharia para programação dos sistemas de protecção, automação e controlo de sistemas de energia. Pretendia-se que as ferramentas suportassem a gestão completa dos sistemas da EFACEC, baseadas nas normas internacionais IEC 61850, IEC 61131-3, IEC 61499 e IEC 60870-5. Nesse âmbito, a EFACEC criou o *Automation Studio*, um ambiente de desenvolvimento integrado, desenvolvido em linguagem C# da *framework* .Net da Microsoft, sendo as ferramentas integradas nesse ambiente como *plugins*. Entre as ferramentas desenvolvidas, conta-se um editor de sinópticos. Tendo este sido desenvolvido em C# .Net, utiliza XAML para a descrição dos diagramas. No entanto, dos equipamentos produzidos pela EFACEC, mais concretamente, a plataforma para automação e controlo de sistemas de energia e gestor de sistemas SCADA UC 500, utiliza SVG para a visualização e interacção com os sinópticos. Assim, embora o editor permita criar os sinópticos para a plataforma UC 500, as linguagens utilizadas não são compatíveis. Para ultrapassar estes problemas de interoperabilidade, entre o editor e a plataforma, surgiu a necessidade de desenvolver um compilador XAML para SVG.

O objectivo do trabalho, desenvolvido no âmbito desta dissertação, foi então o desenvolvimento do referido compilador de XAML para SVG. Este deveria ser integrável no ambiente de edição do *Automation Studio* para, desta forma, permitir a configuração de diversos equipamentos da EFACEC, em particular da plataforma UC 500, a partir desse ambiente de desenvolvimento integrado. Após várias fases de testes e de melhoramentos, o compilador foi definitivamente integrado no editor *Automation Studio* na sua versão 2.0 e seguintes. O resultado positivo deste projecto é visível pela utilização actual em dois exemplos reais, um na subestação de Ermesinde e outro no *Bahrain*, ambos apresentados neste documento.

Abstract

In 2008 an innovation and development project, InPact, is set in motion between EFACEC, EDP Distribution and the University of Minho. The aim of this project is to provide a set of engineering tools for programming of power systems' protection systems, automation and control solutions. It was intended that the tools could support the complete management of EFACEC systems, based on international standards IEC 61850, IEC 61131-3, IEC 61499 and IEC 60870-5. In this context, EFACEC created Automation Studio, an integrated development environment, developed in the C# language using .Net technology. Different tools were integrated into this environment as plugins. Among the tools developed, there was a synoptics editor. This editor was developed in C# .NET, and XAML is used to describe the diagrams. However, the equipment produced by EFACEC, more specifically, the UC 500 platform for power systems automation and control and SCADA systems manager, uses SVG for the visualization and interaction with the synoptics. Thus, although the editor supports creating the synoptics for the UC 500 platform, the languages used are not compatible. To overcome these interoperability problems between the publisher and the platform, the need arose to develop a XAML to SVG compiler.

The aim of the work developed within this dissertation, then, was the development of the XAML to SVG compiler. This compiler should be integrated in the editing environment for Automation Studio, thus allowing configuration of several EFACEC devices, particularly the UC 500 platform, from this integrated development environment. After several phases of testing and improvement, the compiler was definitely integrated into the Automation Studio editor in its 2.0 version. The positive outcome of this project is visible in its current application in two concrete examples, one in the Ermesinde substation and another in Bahrain, both presented in this paper.

Keywords: XAML, WPF, SVG, SCADA, EFACEC

Índice

Agradecimentos	ii
Resumo	iv
Abstract	vi
Índice Tabelas	x
Índice Figuras	xii
Glossário	xiv
Capítulo 1 – Introdução	1
1.1. EFACEC	1
1.2. Projecto InPACT	2
1.3. <i>Automation Studio</i>	3
1.4. Exportador de XAML para SVG	3
1.5. Conclusão	4
Capítulo 2 – Sistemas SCADA	5
2.1. CLP 500	6
2.2. BCU 500	7
2.3. UC 500	8
2.3.1. Arquitectura Geral	9
2.3.2. Componentes nucleares (Base de Dados)	9
2.3.3. Controlos	11
2.3.4. Interface com o utilizador	11
2.4. Conclusão	14
Capítulo 3 – Conversão dos elementos disponibilizados pelo <i>Automation Studio</i> para SVG	15
3.1. XAML e SVG	15
3.2. Equivalência entre elementos	16
3.3. Atributos	18
3.3.1. Matrizes	19
3.3.2. Cores	20
3.3.3. Alinhamento	21
3.3.4. <i>Pen Line Cap</i>	22
3.4. Animações	22
3.4.1. Estudo da funcionalidade das animações suportadas pelo HMI 500	23
3.4.2. Tecnologias para a implementação das animações em SVG	24
3.5. Conclusão	25

Capítulo 4 – Ferramenta Desenvolvida	27
4.1. Arquitectura do SVG.....	27
4.2. Compilador	28
4.2.1. Ficheiro de definições	30
4.3. Processamento	30
4.3.1. Carregar sinóptico para memória.....	31
4.3.2. Gerar SVG inicial.....	32
4.3.3. Processar página.....	33
4.3.4. Gerar <i>MetaDados</i>	35
4.3.5. Adicionar <i>JavaScript</i>	36
4.4. Integração no <i>Automation Studio</i>	36
4.5. Conclusão.....	37
Capítulo 5 – Exemplos	39
5.1. Subestação de Ermesinde	39
5.2. Subestação do <i>Bahrain</i>	43
5.3. Conclusão.....	47
Capítulo 6 – Conclusões	49
Bibliografia.....	51

Índice Tabelas

Tabela 1 - Entidades disponíveis no CLP 500.....	9
Tabela 2 - Significado dos qualificadores das entidades	10
Tabela 3 - Lista de objectos do <i>Automation Studio</i> em XAML que são convertidos para SVG	17
Tabela 4 - Representação dum Matriz para transformadas em XAML	19
Tabela 5 - Representação dum Matriz para transformadas em SVG.....	19
Tabela 6 - Lista de animações suportadas pelo HMI 500	22
Tabela 7 - Número de elementos e animações necessário para traduzir as páginas em XAML	42
Tabela 8 - Número de elementos SVG e animações necessário para traduzir as páginas em XAML	46

Índice Figuras

Figura 1 - Arquitectura SCADA típica	5
Figura 2 - CLP 500	6
Figura 3 - BCU 500	7
Figura 4 - UC 500E.....	8
Figura 5 - Arquitectura de <i>software</i> da unidade central.....	8
Figura 6 - Interface típico para o CLP 500	12
Figura 7 - Arquitectura da visualização de Sinópticos	13
Figura 8 - Exemplos de sinópticos	13
Figura 9 - Exemplo duma declaração em XAML dum botão	15
Figura 10 - Exemplo duma declaração em SVG dum círculo	16
Figura 11 - Representação gráfica do botão e botão clássico	17
Figura 12 - Modelo UML das cores em WPF	20
Figura 13 - Diferentes visualizações para o numérico.....	23
Figura 14 - Estrutura dum ficheiro SVG no HMI 500	27
Figura 15 - Compilador para a geração de páginas SVG a correr no <i>Automation Studio</i>	28
Figura 16 - Ficheiro de definições típico utilizado pelo compilador.....	29
Figura 17 - Passos para a tradução de um ficheiro XAML, para um ficheiro SVG	30
Figura 18 - Modelo UML de uma página em XAML no <i>Automation Studio</i>	31
Figura 19 - Cabeçalho SVG para as páginas geradas pelo compilador	32
Figura 20 - Código SVG gerado para o botão clássico	33
Figura 21 - Código XAML para o botão clássico	34
Figura 22 - Estados para os botões clássicos	34
Figura 23 - Código SVG para animar o botão clássico	34
Figura 24 - Estrutura de <i>MetaDados</i> típico utilizado para as animações na página SVG	35
Figura 25 - Arquitectura do sistema montado na subestação de Ermesinde.....	39
Figura 26 - Página “Estado do sistema” no <i>Automation Studio</i>	40
Figura 27 - Página “Sistema 60” no <i>Automation Studio</i>	41
Figura 28 - Ermesinde no <i>Automation Studio</i>	41
Figura 29 - Sinóptico para a visão global da subestação.....	44
Figura 30 - Sinóptico da arquitectura da subestação.....	45
Figura 31 - Sinóptico das protecções para a subestação	45

Glossário

Automation Studio – Solução de *software* de edição gráfica da EFACEC para dar suporte a toda a gama de produtos da área de engenharia de automação da EFACEC.

Ajax – *Asynchronous JavaScript And XML*, é um conjunto de tecnologias de desenvolvimento *web* que estão relacionadas com o intuito de criar aplicações que enviam e recebem dados do servidor de modo assíncrono e sem interferir com a exibição e interacção com a página actual.

API - *Application Programming Interface*, é um conjunto de funções que estão acessíveis por meio de programação e que as aplicações e bibliotecas disponibilizam para se poder aceder aos serviços que ela fornece.

BCU 500 – É uma unidade da EFACEC utilizada para automação, controlo, supervisão e aquisição de dados no campo da distribuição e aplicação em redes de transmissão de energia, em especial no controlo e protecção de subestações e sistemas de comando.

CSS – *Cascading Style Sheets*, é uma linguagem de estilo utilizada para descrever o aspecto e a formatação dum documento escrito em linguagem de *markup* como o HTML ou XML, é utilizada para separar o conteúdo do aspecto do documento.

CLP 500 – Plataforma SCADA, desenvolvida pela EFACEC, de automação e controlo de energia composto por uma arquitectura de *hardware* e *software* distribuída.

DirectX – Conjunto de APIs da Microsoft para gráficos, que são utilizados para criar aplicações multimédia de elevado desempenho, como jogos de computador, com suporte para gráficos 2D e 3D.

ECMAScript – *European Computer Manufacturers Association Script*, linguagem criada para dar suporte ao *scripting* em navegadores *web*.

GDI/GDI+ - *Graphics Device Interface*, é uma API do *Windows* e fornece suporte para o desenho elementos gráficos em dispositivos como monitores e impressoras. O GDI+ foi introduzido depois do GDI, aquando da introdução do *Windows XP* e trouxe melhorias em relação ao sistema anterior como por exemplo coordenadas para vírgula flutuante, gradientes e suporte para ficheiros de gráficos como o JPEG e o PNG.

HTML – *HyperText Markup Language*, linguagem de *markup* utilizada para a criação de páginas *web* e que podem ser mostradas num navegador.

IEEE754 – *Institute of Electrical and Electronics Engineers 754*, é uma norma padrão para a representação de números binários de vírgula flutuante.

IED – *Intelligent Electronic Device*, trata-se dum dispositivo utilizado para controlar equipamento como disjuntores ou transformadores.

JavaScript – É uma linguagem de *scripting*, aberta e multiplataforma, orientada aos objectos, desenhada para criar aplicações dinâmicas *online*, sendo suportada pela maioria dos navegadores web. Apesar de o *JavaScript* ser orientado para aplicações *online*, é possível usá-lo para aplicações locais.

PLCs - *Programmable Logic Controllers*, trata-se dum microcomputador utilizado para a automação de processos mecânicos como por exemplo o controlo de maquinaria numa linha de montagem numa fábrica.

SCADA - *Supervisory Control and Data Acquisition*, é um pacote de *software* utilizado na supervisão de sistemas de *hardware*.

Sinóptico – Trata-se uma representação diagramática, geralmente associado à tecnologia 2D vectorial, que é tanto animada como interactiva em tempo-real, estando normalmente associados à supervisão, automação, controlo e/ou protecção de sistemas industriais, técnicos ou afins do tipo SCADA.

SVG – *Scalable Vector Graphics*, trata-se dum formato aberto baseado em XML utilizado para definir imagens vectoriais 2D desenvolvido pelo W3C, sendo suportado pela maioria dos navegadores modernos como *Internet Explorer* 9 e 10, *Firefox*, *Google Chrome*, *Opera* e *Safari*.

SMIL – *Synchronized Multimedia Integration Language*, é uma linguagem baseada em XML que permite escrever apresentações multimédia interactivas e permite definir como é que uma apresentação se comporta ao longo dum período de tempo.

SQL – *Structured Query Language*, é uma linguagem declarativa utilizada na pesquisa de dados em bases de dados.

Touch – Protótipo para interacção multitoque da EFACEC, para o sistema SCADA HMI 500, completamente configurável pelo *Automation Studio* (EFACEC, 2012).

InPACT - *Integrated Engineering Tools for Protection, Automation and Control Systems*, projecto de inovação e desenvolvimento da EFACEC cujo o objectivo é o desenvolvimento dum conjunto de ferramentas de engenharia avançadas que serão utilizadas para fazer uma gestão completa dos sistemas da EFACEC.

IEC 61850 – Norma padrão publicada pela IEC (*International Electrotechnical Commission's*) para descrever projectos de automação de subestações eléctricas, definindo protocolos de comunicação entre os dispositivos da subestação.

IEC 61131-3 – É uma norma publicada pela IEC utilizada para descrever/unificar todas as linguagens de programação utilizadas em controladores lógicos programáveis.

IEC 60870-5 – É uma norma publicada pela IEC utilizada para descrever, para sistemas de controlo de supervisão e aquisição de dados em engenharia electrónica e aplicações de automação em sistemas

de potência, no envio de mensagens de controlo entre dois sistemas que usam conexões de comunicação directa e permanentes.

IEC 61499 – É uma norma padrão, aberta, utilizada para o controlo distribuído e automação publicada pela IEC para descrever como as aplicações podem ser construídas usando FBD (*Function Block Diagrams*).

UC 500 – Unidade Central, é um produto da EFACEC utilizado para automação e controlo de sistemas de energia, podendo ser utilizado como gestor de sistemas SCADA ou simplesmente como concentrador de informação, é baseado num PC industrial suportado pelo sistema operativo *Windows XP*.

UC 500E – Trata-se dum produto semelhante à UC 500 em termos de funcionalidade, sendo que as diferenças encontram-se ao nível de *hardware*, utilizando componentes de maior fiabilidade sem partes móveis ou ventoinhas de refrigeração, ao nível do sistema operativo utiliza o *Windows XP Embedded*.

URT – Unidade Remota Terminal, é um dispositivo de interface entre o mundo físico e sistemas lógicos como os sistemas SCADA.

User32 – É um componente do sistema operativo *Windows* que fornece a aparência e comportamento do interface gráfico para o utilizador.

XML – *Extensible Markup Language*, linguagem de *markup* utilizada para a definição de modelos.

XAML – *Extensible Application Markup Language*, linguagem de *markup* declarativa que faz parte da arquitectura do WPF e permite separar a parte gráfica da parte lógica em aplicações.

W3C – *World Wide Web Consortium*, é um consórcio responsável pela padronização na indústria de tecnologias *web*.

WPF – *Windows Presentation Foundation*, é um sistema gráfico da Microsoft, cuja linguagem é baseada em XML, para aplicações .NET em *Windows*, usando *DirectX* e que vem substituir o anterior sistema gráfico baseado em User32 e GDI/GDI+.

Capítulo 1 – Introdução

Desde a introdução da linguagem XML (Jacobs, 2006), em 1996 pelo W3C, que surgiram várias tecnologias para tirar partido desta linguagem, desde linguagens para interacção multimédia, como o SMIL (W3C, 2008), a linguagens para representar elementos vectoriais 2D, como o SVG (Eisenberg, 2002) (W3C, 2003) e o XAML (Allen, 2006). Apesar destas linguagens serem todas baseadas em XML, cada uma delas apresenta peculiaridades inerentes, o que as torna incompatíveis umas com as outras, mesmo tratando-se de linguagens utilizadas para representar os mesmos objectos, como é o caso do SVG e do XAML.

No meio empresarial, várias empresas têm equipamentos a funcionar com as duas linguagens. É o caso da EFACEC, uma empresa que utiliza programas com ambas as linguagens e tem a necessidade de os interligar. A título de exemplo, ligar as suas plataformas de *hardware* ao *Automation Studio*, o ambiente de desenvolvimento, baseado em tecnologia .NET da Microsoft, que é utilizado para configurar essas plataformas. Enquanto o primeiro utiliza SVG nos seus sistemas de visualização, o segundo utiliza WPF (MacDonald, 2010) para criar as páginas que a plataforma de *hardware* irá apresentar.

Devido a estes problemas de interoperabilidade entre as diferentes tecnologias, surgiu a necessidade de criar um compilador para fazer a ponte entre essas duas linguagens gráficas e permitir que o *Automation Studio* pudesse ser utilizado para configurar o sistema SCADA da plataforma de *hardware*. Outra alternativa, obrigaria a alterar a programação de vários equipamentos em funcionamento no mercado, o que, pela sua complexidade, torna esta operação morosa e com custos elevados.

1.1. EFACEC

A EFACEC¹ nasceu em 1962, fruto de várias alterações de sociedades e áreas de actividade desde 1905, e é actualmente o maior grupo eléctrico nacional de capitais portugueses com um volume de negócios anual superior a 1000 milhões de euros, contando com mais de 4500 colaboradores e uma presença em mais de 65 países.

¹ <http://www.efacec.pt> (acedido pela última vez em 25 de Outubro de 2013)

O seu portfólio de actividades foi reorganizado em três áreas principais: Transportes e logística, engenharia e serviços onde se inclui automação, ambiente, engenharia e renováveis, e energia onde se incluem transformadores, *servicing* de energia e equipamentos de média e alta tensão.

É no departamento de automação que é necessária a utilização duma “ferramenta” que permita a comunicação entre as duas linguagens referidas anteriormente.

1.2. Projecto InPACT

Desde 2004 que a utilização da tecnologia baseada na norma IEC 61850 (Baigent, et al., 2004) por parte da indústria de energia tem vindo a crescer, sendo cada vez mais uma solução na automação de subestações de energia eléctrica e nos mais variados domínios de IT/Automação de sistemas eléctricos de energia. Com a introdução do IEC 61850 houve também uma alteração no mercado de automação de sistemas de energia, uma vez que esta norma definiu uma linguagem, em XML, permitindo que houvesse uma interoperabilidade entre diferentes fabricantes e uma evolução tecnológica dos IEDs (Dispositivo Electrónico Inteligente, do inglês, *Intelligent Electronic Device*) e consequentemente um aumento da complexidade das funções de protecção das funções de protecção existentes nos sistemas de automação. Actualmente, a aplicação desta norma está disponível em centenas de instalações e em milhares de dispositivos. É neste contexto que surge o projecto InPACT.

O projecto InPACT (Paulo, et al., 2009), com início em 2008, é um projecto de inovação e desenvolvimento da EFACEC tendo como objectivo o desenvolvimento dum conjunto de ferramentas de engenharia avançadas para aplicação a sistemas de protecção, automação e controlo de sistemas eléctricos de energia. Pretendeu-se que as ferramentas suportem uma gestão completa dos sistemas da EFACEC e que, ao mesmo tempo, incluam a máxima integração possível de produtos de terceiros de acordo com a normalização internacional aplicável. O projecto tem particular destaque na automação de subestações e em tecnologias de automação distribuída com base na norma IEC 61850, e normalização internacional associada, e na norma 61131-3 (Geneva & Schweiz, 2003). Ao mesmo tempo, tenta abranger conceitos mais recentes incluídos na norma IEC 61499 (Geneva & Schweiz, 2003) e abordar outras tecnologias comunicativas baseadas na norma IEC 60870-5 (Geneva & Schweiz, 2013). Neste sentido, foram desenvolvidas e concebidas ferramentas que foram implementadas como *plugins* para o *Automation Studio*, um ambiente de desenvolvimento integrado (IDE, do inglês *Integrated Development Environment*) da EFACEC e, como tal, integradas nas ferramentas de desenvolvimento da empresa.

1.3. *Automation Studio*

O *Automation Studio* é uma solução de *software* que compreende um conjunto de ferramentas integradas destinadas a dar suporte a várias actividades da engenharia de automação suportando toda a gama de produtos EFACEC, abrangendo produtos de terceiros. Essas ferramentas foram desenvolvidas tendo em conta um conjunto de factores:

- Fornecer uma interface amigável e rica para o utilizador com princípios de interacção e metáforas geralmente associados a utilizadores finais, onde se inclui a facilidade de visualização, análise e manipulação de definições e edição diagramática;
- Modelos de interacção unificado para operações similares e integração de funcionalidades para os diversos perfis de utilização (desenho, configuração e validação; análise e teste; operação e manutenção) de funcionalidades num único ambiente de ferramentas;
- No mesmo ambiente integrado, a gestão de toda a família de produtos EFACEC para os seguintes domínios: automação de subestações, comando e controlo de centrais electroprodutoras, automação de redes de distribuição, entre outros sistemas;
- Ser dirigido para as normas IEC 61850, IEC 61131-3, e outras normas internacionais, permitindo que as ferramentas possam ser utilizadas por equipamentos de múltiplos fabricantes;
- Ser orientado para abstrações típicas destes sistemas, e permitir uma utilização, a longo prazo, das ferramentas e das definições criadas, com ganhos crescentes e significativos na reutilização de definições, para a automatização de tarefas de engenharia de automação sem significativo valor acrescentado.

1.4. Exportador de XAML para SVG

Das várias ferramentas que foram desenvolvidas para o *Automation Studio*, uma delas consistiu num editor de modelos de sinópticos para programação gráfica dos equipamentos produzidos pela EFACEC, em particular, o equipamento UC 500, uma plataforma para automação e controlo de sistemas de energia, que pode ser utilizado como gestor de sistemas SCADA (Boyer, 2010), sendo que esta plataforma é controlada pelo CLP 500.

A parte gráfica da UC 500 é controlada pelo HMI 500. Este componente foi criado para fornecer suporte a funcionalidades de visualização e interacção com sinópticos (sendo que a interacção com os sinópticos é feita através de acesso remoto multiposto por navegação *Web*), fornecer uma nova

arquitetura de visualização de diagramas animados com o intuito de remover a antiga tecnologia de visualização e interacção com sinópticos.

O HMI 500 utiliza tecnologias *open source*, sendo que o SVG é utilizado para a visualização dos sinópticos juntamente com outras tecnologias *Web*, como o *Ajax* (Ullman & Dykes, 2007) que é utilizado para fornecer interacção nos sinópticos.

O editor de sinópticos do *Automation Studio*, foi desenvolvido na plataforma .NET da Microsoft utilizando a linguagem C#. Tem como modelo de visualização o WPF que, embora permita criar os sinópticos não é compatível com o HMI 500, uma vez que, como foi dito anteriormente, este utiliza SVG. Foi então criado o exportador XAML-SVG para fazer a ponte entre o editor de sinópticos e o HMI 500.

O compilador está inserido no *Automation Studio* como um *plugin*, estando acessível através dos menus de contexto do *Automation Studio* ou através de um executável a partir da linha de comandos. A descrição do processo de desenho e implementação deste exportador, bem como das tecnologias associadas, constitui o objectivo deste documento.

1.5. Conclusão

Este capítulo serviu para contextualizar o trabalho descrito neste documento. A utilização, em simultâneo, de produtos com diferentes linguagens pela EFACEC, criou a necessidade de desenvolver uma ferramenta que permita a comunicação entre os diferentes produtos, o compilador.

Os capítulos seguintes irão descrever as tecnologias utilizadas no *Automation Studio*, e no CLP 500, e a solução adoptada para criar o compilador. O Capítulo 2 descreve em mais profundidade a plataforma CLP 500. No Capítulo 3 são abordadas, mais detalhadamente, as linguagens XAML e SVG e as suas diferenças. O Capítulo 4 descreve a implementação do compilador. O Capítulo 5 apresenta exemplos de aplicação do compilador. Finalmente, no Capítulo 6, são apresentadas conclusões e algumas considerações para trabalho futuro.

Capítulo 2 – Sistemas SCADA

Um sistema SCADA tem como principal objectivo a supervisão dum sistema e controlo sobre o mesmo. Trata-se dum *software* que está posicionado acima do nível de *hardware* ao qual está ligado através dum interface, geralmente através de PLCs (Controladores Lógico Programáveis do inglês *Programmable Logic Controllers*) ou outro tipo de hardware comercial. Este tipo de *software* é utilizado no controlo de processos, como a produção de energia, fabricação dos mais variados componentes, entre outros.

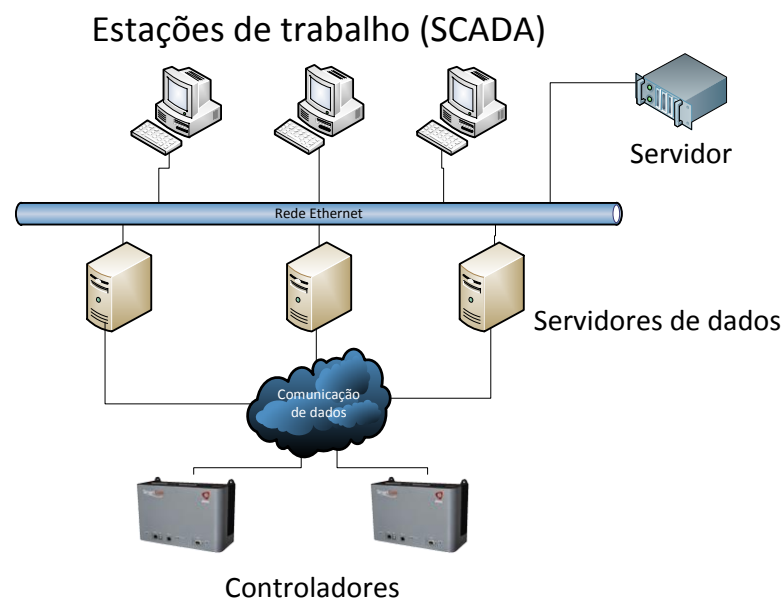


Figura 1 - Arquitectura SCADA típica

A evolução deste tipo de sistemas tem sido muito rápida e existem sistemas SCADA que controlam milhares de dispositivos de I/O (*Input/Output*). Tipicamente, um sistema SCADA está dividido em duas camadas: uma camada Cliente e uma camada Servidor (ver Figura 1).

A EFACEC desenvolve diversos produtos com aplicações específicas de sistemas SCADA. Para o contexto desta dissertação, irá ser abordado a plataforma SCADA CLP 500, juntamente com a sua unidade central, a UC 500 da qual faz parte o seu visualizador de sinópticos em SVG, o HMI 500.

2.1. CLP 500

O CLP 500 é uma plataforma para automação de sistemas de energia, desenvolvida pela EFACEC e representa uma nova geração de equipamentos de automação com integração de *hardware* e *software* numa arquitectura distribuída.



Figura 2 - CLP 500

A sua função primária desta plataforma (ver Figura 2), consiste em adquirir dados do equipamento ao qual está ligado e de executar comandos através de sistemas industriais interligando diversos IED, tal como a BCU 500 (ver Figura 3) e a UC 500 (ver Figura 4). Com uma mudança do *software* de aplicação e sem alterar o *hardware* da plataforma, o CLP 500 pode funcionar como unidade remota terminal ou como uma unidade de automação ou como um sistema de protecção - controlo e de comando de subestações.

Esta flexibilidade, por parte da plataforma, foi alcançada devido ao seguinte conjunto de objectivos adoptados aquando do seu desenvolvimento e que agora fazem parte das suas características:

- Adopção duma arquitectura modular e escalável;
- Utilização duma arquitectura multiprocessador distribuída;
- Utilizações de componentes industriais normalizados;
- Suporte de capacidades de comunicação poderosas.

Tais características oferecem vantagens significativas a qualquer cliente, como por exemplo:

- Suporte de desenvolvimento do próprio cliente;
- Utilização do mesmo *hardware* em diferentes aplicações, reduzindo custos de manutenção;
- Solução escalável com elevada capacidade evolutiva, protegendo desta maneira o investimento inicial;
- Verdadeiro sistema aberto, onde aplicações e/ou equipamentos de terceiros podem ser suportados;
- Interface Humana Máquina opcional, dispondo dum conjunto de ferramentas gráficas amigáveis.

2.2. BCU 500

O BCU 500 é uma unidade para automação, controlo, supervisão e aquisição de dados. Possui um painel configurável e programável através do *Automation Studio*, e uma arquitectura multiprocessador desenvolvida especificamente para a execução de diferentes funções. Devido à sua elevada fiabilidade e robustez, é utilizado no campo da distribuição e aplicação em redes de transmissão de energia, em especial, no controlo e protecção de subestações e sistemas de comando.



Figura 3 - BCU 500

A configuração da BCU (ver Figura 3) é feita de acordo com a norma IEC 61850 e parte de automação é feita segundo a norma IEC 61131-3, sendo totalmente programável por qualquer computador, desde que na mesma rede que o BCU 500. O visor gráfico (ver Figura 3) do BCU 500

pode disponibilizar várias informações tais como, medidas analógicas, informação estática, descrição de alarmes e informação acerca do estado dos aparelhos, sob a forma de sinópticos.

2.3. UC 500

No topo da hierarquia da plataforma CLP 500 surge uma Unidade Central, denominada por UC 500 ou UC 500E, trata-se dum produto vocacionado para automação e controlo de sistemas de energia, que pode ser utilizado como gestor de sistemas SCADA ou simplesmente como concentrador de informação.



Figura 4 - UC 500E

Ambas as unidades são baseadas num computador industrial e suportadas pelo sistema operativo *Windows XP*. A unidade UC 500E (ver Figura 4) não contém partes móveis e a sua unidade de CPU não utiliza qualquer disco de memória rígido ou ventoinhas de refrigeração, ao contrário da unidade UC 500.

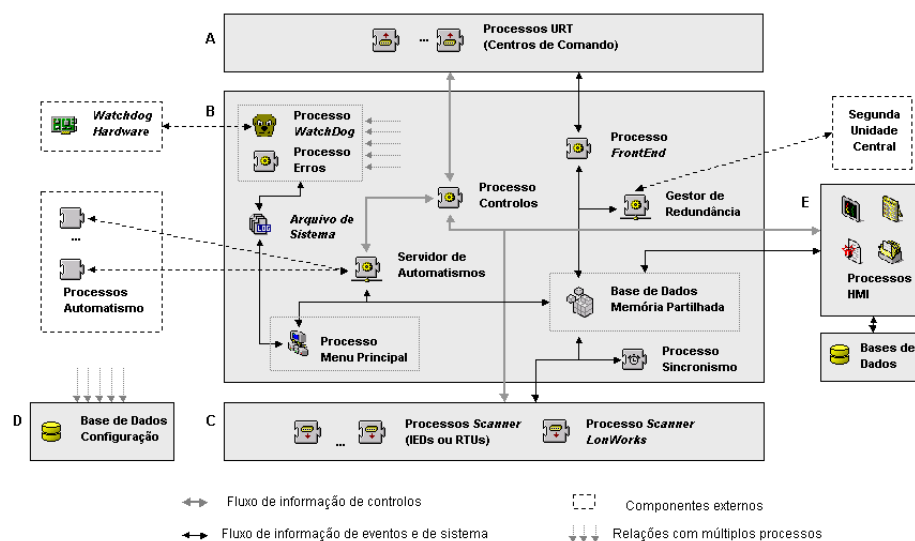


Figura 5 - Arquitectura de *software* da unidade central

2.3.1. Arquitectura Geral

O conjunto de *software* da Unidade Central está subdividido em 5 grupos (ver Figura 5): Componentes de comunicação com centros de comando, ou processo de comunicações URT; componentes nucleares; componentes de comunicação com IEDs, URTs e unidades de aquisição e controlo ou *scanners*; base de dados de configuração; componentes de interface com o utilizador.

2.3.2. Componentes nucleares (Base de Dados)

O meio de comunicação entre os vários processos da Unidade Central é a memória partilhada, que contém uma imagem do estado actual de todas as entidades de telecontrolo além do estado da própria Unidade Central. Este componente nuclear é chamado de base de dados em tempo real.

Tabela 1 - Entidades disponíveis no CLP 500

Entidade	Tipo	Subtipo	Valor	Estados
Digital	Interno, Telemetria ou Derivado	Simple, Duplo ou Enumerado de 2, 3 ou 4 <i>bits</i>	Inteiro nos intervalos [0,1], [0,3], [0,7] ou [0,15] consoante o subtipo	Um estado definido para cada valor possível.
Medida		Não aplicável	Virgula flutuante de 32 bits (formato IEEE 754) ²	Muito baixo, baixo, normal, alto e muito alto
Contador				Normal, alto e muito alto

As entidades de telecontrolo (ver Tabela 1) permitem representar características duma instalação ou sistema que se pretende monitorizar e controlar. O estado actual ou as alterações de estado das entidades são normalmente adquiridos pelos *scanners*, processados pelos processos nucleares e automatismos e enviados para centros de comando através de comunicações URT.

As entidades Digitais representam estados discretos com significado individual, como por exemplo: o estado dum disjuntor, o estado duma válvula, a posição da tomada dum transformador. As entidades Medida representam grandezas contínuas como: valores de tensão, pressão ou caudal.

² Os valores, quando correspondentes a uma grandeza física, têm associada uma unidade (KW, m³/s, m, A, etc.).

As entidades Contador correspondem a valores de grandezas integrados ao longo do tempo, como: contadores de energia, contagens discretas do número de manobras dum determinado órgão.

O tipo dum entidade (ver Tabela 1) está associado à fonte de aquisição, sendo que as entidades do tipo Interna correspondem ao estado da própria unidade central: validade da sincronização, percentagem da utilização de memória, estado de comunicação com um determinado centro de comando.

Tabela 2 - Significado dos qualificadores das entidades

Qualificador	Significado
<i>Offscan manual</i>	O utilizador da unidade inibiu as comunicações do <i>scanner</i> que adquire a entidade ou efectuou uma qualquer outra acção que inibe o refrescamento da entidade. O valor da entidade é o último valor válido conhecido. Este atributo é um atributo de qualidade da entidade.
<i>Offscan automático</i>	O <i>scanner</i> que adquire a entidade não consegue estabelecer comunicações com o módulo periférico ou o módulo periférico não é capaz de enviar um ou mais atributos da entidade. O valor da entidade é o último valor válido conhecido. Este atributo é um atributo de qualidade da entidade.
Inálido	O estado de invalidade provém, normalmente, dos módulos periféricos e significa que o valor da entidade não é válido por razões não determinadas, nomeadamente mau funcionamento do equipamento de aquisição. Quando uma entidade está no estado de invalidade o seu valor é irrelevante uma vez que é desconhecido. Este atributo é, na realidade, um estado adicional possível para uma entidade.
<i>Overflow</i>	O valor da medida ou contador é superior ao máximo ou inferior ao mínimo valor que o equipamento permite adquirir ou aquele teoricamente válido para o sistema, equipamento ou grandeza representada pela unidade. O valor dum entidade em <i>overflow</i> continua válido e actual. Este atributo é um atributo do valor da entidade (apenas para medidas e contadores).
Alarme	O estado actual da entidade corresponde a um estado de alarme. Este atributo é um atributo do estado da entidade.
Qualificadores de estado adicional	Estão disponíveis seis qualificadores para serem utilizados em aplicações específicas com significado não definido.

As entidades do tipo Telemetria são entidades que são adquiridas através dum *scanner*.

As entidades do tipo Derivada são entidades que resultam do processamento de dados realizados por funções automáticas ou automatismos. Qualquer que seja a entidade, ela possui 5 atributos: o valor que corresponde à leitura da grandeza física ou lógica associada; o estado que corresponde à tradução discreta do valor e seu significado; os estados possíveis exclusivos; a datação que é feita relativamente à hora de relógio da unidade Central; os qualificadores binários (ver Tabela 2).

2.3.3. Controlos

Os controlos permitem aos utilizadores, ou instalação, agir sobre uma instalação e existem dois tipos elementares de controlos: comandos e *set-points*. Os comandos são controlos aos quais não existem valores associados. Um comando só por si é suficiente para definir a acção desejada sobre o sistema, como por exemplo, ordem de abertura duma válvula ou ordem de fecho dum disjuntor. Os *set-points* são controlos aos quais existem valores associados, como por exemplo, a alteração de posição de tomada dum transformador ou envio de parâmetro para unidade remota. Os valores são vírgula flutuante de 24 bits de resolução no formato IEE754.

Os controlos podem ser internos ou sequências de controlos. No caso de serem internos, provocam alterações em entidades digitais, medidas ou contadores derivados. Se forem sequências de controlos, a sua função implica a execução duma sequência de grupos de controlos individuais, com interrupção de sequência opcional, em caso de falha de controlos individuais.

2.3.4. Interface com o utilizador

A interface da Unidade Central é do tipo monoposto, com opção multiutilizador ou utilizador único, e é constituída por um conjunto de aplicações separadas mas que estão integradas no ambiente de trabalho (ver Figura 6). Dependendo do conjunto de funções SCADA utilizadas na aplicação e configurações específicas, algumas dessas aplicações e funcionalidades poderão estar, ou não, disponíveis. Como tal, a interface apresentada ao utilizador muda conforme a função da unidade no sistema de automação.

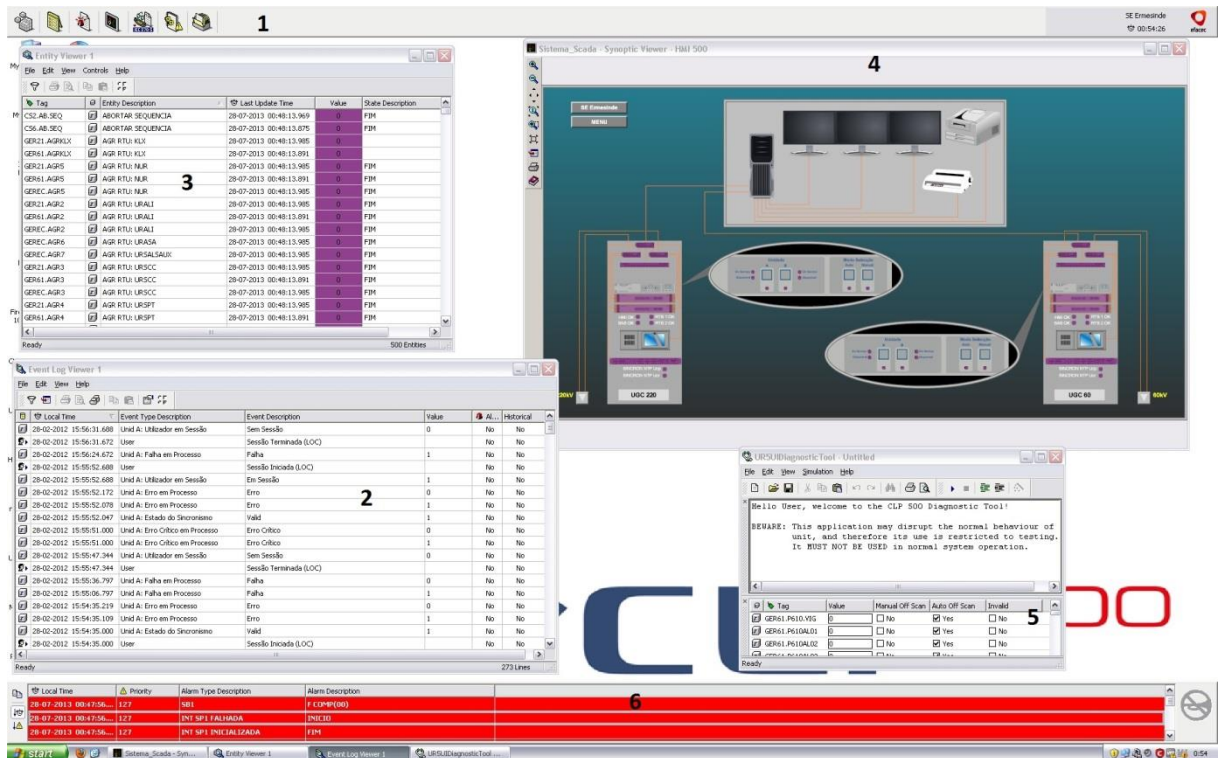


Figura 6 - Interface típico para o CLP 500

A Figura 6 mostra uma interface possível do CLP 500, que permite controlar a plataforma e equipamentos ligados à mesma. Nesta interface existe:

- Um menu principal do CLP 500 (ver Figura 6 - #1), que dá acesso às aplicações que fazem parte do CLP 500 e aos sinópticos. Este menu é totalmente configurável a partir do *Automation Studio*;
- Uma aplicação para resumir eventos (ver Figura 6 - #2) onde se pode obter uma lista com a hora e a data de todos os eventos que aconteçam no sistema, tais como o registo de erros ou indicação da hora e data em que o utilizador acedeu ao sistema;
- Um visualizador de entidades (ver Figura 6 - #3) que permite ter uma lista de entidades filtradas por tipo e onde se podem obter informações acerca de quando foi a última vez que essa entidade foi manipulada, descritivos e qual o valor que está associado à mesma, entre outras informações;
- O visualizador de sinópticos (HMI 500) (ver Figura 6 - #4) onde o utilizador tem, duma maneira gráfica, acesso à informação em tempo real, sendo que pode ter múltiplas aplicações de visualização a correr ao mesmo tempo;
- Uma aplicação para fazer diagnóstico, como por exemplo testar se a informação contida no sinóptico está correcta (ver Figura 6 - #5);
- Uma aplicação que permite ver os alarmes que foram activados no sistema (ver Figura 6 - #6), com informação da data e hora a que foram activados, e qual o motivo.

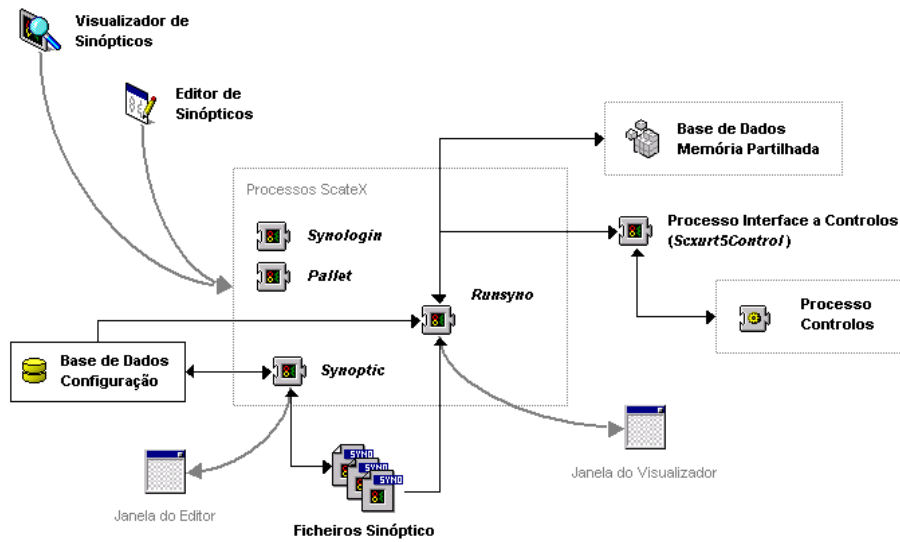


Figura 7 - Arquitectura da visualização de Sinópticos



Figura 8 - Exemplos de sinópticos

Uma das aplicações que está contida no interface do utilizador é o visualizador de sinópticos (ver Figura 7). Um sinóptico (ver Figura 8) consiste numa representação diagramática animada e interactiva em tempo-real, geralmente recorrendo a gráficos 2D vectoriais disponibilizados por linguagens como o SVG ou WPF, de um processo controlado através dum operador.

Estão normalmente associados à supervisão, automação, controlo e/ou protecção de sistemas industriais, técnicos ou afins do tipo SCADA.

2.4. Conclusão

Este capítulo apresentou a plataforma CLP 500, uma solução para automação de sistemas de energia da EFACEC e que utiliza sistemas de *hardware* SCADA como a UC 500. Um dos problemas que a empresa enfrenta é a diferença entre as diferentes linguagens vectoriais utilizadas, por um lado, na plataforma, que utiliza uma linguagem SVG nos seus sistemas de visualização, por outro, no *Automation Studio*, que utiliza a linguagem XAML nos seus sistemas de visualização.

Embora o SVG e o XAML sejam ambas baseadas em XML, são linguagens diferentes. Apesar de ambas serem utilizadas para representar vectorialmente objectos em 2D, têm formas diferentes de representar o mesmo objecto, fazendo com que as duas linguagens não sejam compatíveis e haja necessidade de criar o compilador. No próximo capítulo, serão analisadas as diferenças entre as duas linguagens e a forma como as interligar.

Capítulo 3 – Conversão dos elementos disponibilizados pelo *Automation Studio* para SVG

A linguagem utilizada pelo *Automation Studio* para representar um sinóptico é o WPF. Como o HMI 500 utiliza SVG para representar o mesmo documento, torna-se necessário traduzir a linguagem WPF em SVG. O documento SVG gerado dessa “tradução”, deverá ser compatível com a maioria dos navegadores modernos e com o HMI 500.

Para implementar o compilador pretendido, foi necessário realizar um estudo comparativo entre o WPF e o SVG, tendo em vista determinar quais as diferenças entre ambos e, de que forma, se poderá traduzir uma linguagem noutra. Para este estudo foram considerados os elementos disponibilizados pelo *Automation Studio* que teriam de ser traduzidos para SVG (os atributos que podem ser utilizados na definição dos elementos, como a cor, os tipos de letra e outros atributos, e por fim, as animações a que cada elemento em WPF pode estar sujeito) e de que forma será traduzida para SVG.

3.1. XAML e SVG

XAML é uma linguagem declarativa baseada em XML. Esta linguagem faz parte da arquitectura do WPF que, por sua vez, faz parte da arquitectura .NET. Permite simplificar a criação de interfaces gráficas em aplicações, ao separar a parte gráfica da parte lógica. Uma das características do XAML é a separação dos elementos, em que a junção das duas partes é feita através da definição de classes parciais.

WPF é um sistema gráfico para aplicações .NET, que vem substituir o anterior sistema baseado em User32 e GDI/GDI+ (que tem um custo elevado quer em termos de complexidade, quer em termos de desempenho) para ser baseado em DirectX (Nathan, 2007), aproveitando assim todo o desempenho disponível nas placas gráficas. Como o WPF utiliza um motor de *rendering* vectorial, o XAML, ele garante que as aplicações gráficas se comportam sempre da mesma maneira, independentemente do sistema operativo utilizado.

```
<Button Height="72" Width="160" Content="Click Me" />
```

Figura 9 - Exemplo duma declaração em XAML dum botão

A Figura 9, mostra a declaração dum botão em XAML, onde os atributos *Height* e *Width* representam a altura e a largura respectivamente, e o atributo *Content*, o texto que aparece no próprio botão. Embora o WPF permita definir os elementos gráficos programaticamente e, portanto, ser completamente independente do XAML, uma abordagem declarativa vai permitir que a parte gráfica da aplicação fique completamente separada do código, podendo ser completamente manipulada por *designers* sem necessidade de intervenção a nível programático.

```
<circle cx="100" cy="50" r="40" fill="red" />
```

Figura 10 - Exemplo duma declaração em SVG dum círculo

SVG é uma linguagem não proprietária, que serve para representar gráficos 2D em XML, permitindo que sejam representados três tipos de objectos gráficos: vectoriais, imagens e texto. A Figura 10, mostra um exemplo da declaração de um círculo em SVG, onde os atributos *cx* e *cy* representam a posição onde o centro está colocado na página, o *r* representa o raio e o *fill* a cor de fundo.

Criado pela W3C como um padrão para plataformas móveis e para a interactividade gráfica na *World Wide Web* (WWW), o SVG está disponível na maioria dos navegadores modernos como o *Internet Explorer*, *Firefox* e o *Google Chrome*, assim como em inúmeros dispositivos móveis, podendo ser integrado em tecnologias com linguagens *Web* como o *AJAX* entre outras.

A cada um dos elementos, quer em XAML quer em SVG, podem ser aplicados filtros de efeitos, máscaras, transformadas, como transformadas de escala ou rotação, entre outros, ou interactividade - ambas as linguagens fornecem uma vasta API para eventos como *click* do rato.

3.2. Equivalência entre elementos

Como dito em cima, para este estudo, foram considerados os elementos disponibilizados pelo *Automation Studio*, que teriam de ser traduzidos para SVG. Na análise dos elementos, foram tidos em consideração os atributos a que os elementos podem estar sujeitos e a sua definição (atributos como a cor, os tipos de letra, entre outros). Por fim, também foram analisadas as animações a que cada elemento pode estar sujeito e de que forma será traduzida para SVG.

Tabela 3 - Lista de objectos do *Automation Studio* em XAML que são convertidos para SVG

Elemento XAML	Conversão SVG Directa	Conversão SVG Indirecta
Rectângulo <RectangleGeometry>	<rect>	
Elipse <EllipseGeometry>	<ellipse>	
Linha <LineGeometry>	<line>	
Polígono <Polygon>	<polygon>	
Polilinha <Polyline>	<polyline>	
Arco <Arc>		<path>
Conector <StraightConnector>		<path>
Botão <Button>		<rect> + <clipPath> + <text>
Botão Clássico <ClassicButton>		2x <path> + <rect> + <clipPath> + <text>
Canvas <Canvas>		<g>
Formas Complexas <PathGeometry>	<path>	
Layer <PageLayer>		<g>
Imagem <Image>	<image>	
Texto <TextArea>		<rect> + <text>
Texto Limitado por Área <TextBlock>		<rect> + <text>
Texto Numérico <NumericText>		<rect> + <text>
Texto Numérico Limitado por Área <NumericTextBlock>		<clipPath> + <rect> + <text>
Etiqueta <Label>		<rect> + <clipPath> + <text>
Etiqueta Numérica <NumericLabel>		<rect> + <clipPath> + <text>
Grupo de Formas Complexas <GeometryGroup>	<path>	
MultiState <MultiState>		*
Símbolo <Symbol>		*

A Tabela 3 mostra o resultado do estudo da conversão dos elementos XAML para elementos SVG.



Figura 11 - Representação gráfica do botão e botão clássico

A primeira coluna mostra os elementos utilizados no editor, e que terão de ser convertidos para SVG, a coluna também mostra qual o *tag* utilizado para representar o elemento em XAML.

Ainda relativamente à primeira coluna, verifica-se que existem dois tipos de botões, o botão e o botão clássico, sendo que o botão representa um botão com aspecto mais moderno (ver Figura 11 – Botão branco) e o botão clássico representa um botão com o aspecto dos botões utilizados pelo *Windows 95* (ver Figura 11 – Botão cinzento). A segunda e terceira coluna mostram quais as *tags*, em SVG, que foram utilizadas para representar os elementos WPF, sendo que a segunda coluna representa os elementos para os quais existe um elemento equivalente e a terceira coluna representa os elementos para os quais não existe um elemento equivalente.

Os objectos que não tiveram tradução directa, os da terceira coluna, foram traduzidos através da utilização de objectos similares já existentes em SVG ou da combinação dos mesmos, como é o caso, por exemplo, do Texto (*TextArea*) que foi traduzido utilizando um rectângulo juntamente com o texto.

O elemento *Symbol*, um elemento XAML específico do *Automation Studio*, permite representar qualquer elemento que está descrito na Tabela 3, incluindo páginas com múltiplas camadas e objectos, este elemento pode ser replicado inúmeras vezes por múltiplas páginas, permitindo dessa maneira ser reutilizável.

O elemento *MultiState* é um elemento que existe no HMI 500. Trata-se de um elemento XAML, específico do *Automation Studio*. Permite representar um dado estado, para um dado valor, de uma variável. Trata-se de um elemento que contém múltiplas camadas, ou estados, onde cada camada pode conter qualquer elemento descrito na Tabela 3, sendo que cada camada só é visível para um dado valor de uma variável.

3.3. Atributos

Cada um dos elementos em XAML está sujeito a inúmeros atributos, que determinam a forma como o objecto é visualizado no ecrã. Os atributos determinam desde a cor que um objecto tem, ao tipo de letra utilizado para texto, entre outros parâmetros, sendo que, o *Automation Studio* permite alterar algumas características desses atributos.

Estes atributos tiveram de ser analisados e traduzidos para SVG, tendo em conta as diferenças e limitações inerentes a cada uma das linguagens. Devido ao enorme número de atributos que cada elemento XAML e SVG possuem, nesta dissertação, apenas serão referidos os atributos mais relevantes.

3.3.1. Matrizes

O *Automation Studio* permite manipular completamente os objectos, isto é, podem ser rodados, inclinados, redimensionados e reposicionados. Estas transformações podem ser aplicadas a um objecto ou a grupos de objectos.

As manipulações dos objectos são muito importantes, pois permitem que um dado objecto, como por exemplo um rectângulo, possa assumir outras formas, que não a sua forma inicial e regular, permitindo assim, obter um leque de objectos muito mais vasto que o inicial.

Essa manipulação, quer em XAML, quer em SVG, pode ser feita de duas maneiras: ou através de atributos que especificam qual a acção a executar, como por exemplo rodar um objecto, ou através da utilização de matrizes, que podem definir um ou mais tipos de manipulação. No caso da tradução de XAML para SVG, optou-se por utilizar matrizes, uma vez que se trata de um processo mais simples de aplicar. Durante este processo, o programador apenas se preocupa com o ponto de partida e o resultado final, a matriz processa os passos intermédios necessários até atingir esse resultado.

Tabela 4 - Representação duma Matriz para transformadas em XAML

a	b	0
c	d	0
trx	try	1

Tabela 5 - Representação duma Matriz para transformadas em SVG

a	c	trx
b	d	try
0	0	1

As matrizes utilizadas quer pelo XAML, quer pelo SVG, são matrizes 3x3 (3 linhas e 3 colunas) e ambas representam os valores de maneira diferente (como se pode ver nas Tabela 4 e Tabela 5, as matrizes são transpostas de uma linguagem para a outra). A sua representação, nas duas linguagens, é feita através de um vector [a, b, c, d, trx, try], sendo a representação desse vector igual para ambas as linguagens (a linha/coluna [0, 0, 1], por ser constante, é apenas para representada na forma matricial). Apesar da sua representação vectorial ser a mesma, na sua tradução de XAML para SVG, os valores utilizados para a translação (movimento), respectivamente os valores trx e try, não são os

Capítulo 3 – Conversão dos elementos disponibilizados pelo *Automation Studio* para SVG mesmos, já que em XAML os valores utilizados são relativos, ou seja, o valor é relativo ao controlo onde está inserido, enquanto em SVG, por uma decisão de desenho, o seu valor é absoluto, ou seja, o seu valor é sempre relativo à origem da página onde o objecto está inserido.

Apesar de não se conseguir saber quais as manipulações que foram aplicadas a um objecto a partir de uma matriz, pode-se utilizar a mesma matriz para aplicar um certo tipo de manipulação ao objecto. Por exemplo, ao aplicar o vector $[1, 0, 0, 1, \text{trx}, \text{try}]$, está a ser efectuada uma transformada de translação (movimento), com trx e try a representar o valor da translação no eixo dos XX e no eixo dos YY respectivamente. Utilizando-se o vector $[\cos(y), \sin(y), -\sin(y), \cos(y), 0, 0]$, temos uma transformada de rotação, onde y indica o ângulo de rotação. Com o vector $[a, 0, 0, d, 0, 0]$ obtemos uma transformada de escala, com a a representar o valor da escala a aplicar no eixo dos XX e d o valor da escala a aplicar no eixo dos YY. Utilizando o vector $[1, 0, \tan(y), 1, 0, 0]$, inclinámos o objecto no eixo dos XX e com $[1, \tan(y), 0, 1, 0, 0]$, inclinámos o objecto no eixo dos YY, onde o y representa o ângulo de inclinação.

3.3.2. Cores

O *Automation Studio* permite atribuir um conjunto de cores a todos os objectos, sendo possível atribuir cores de fundo, cor da linha, cor determinado texto, entre outros, com as opções de cores sólidas ou gradientes, lineares e radiais.

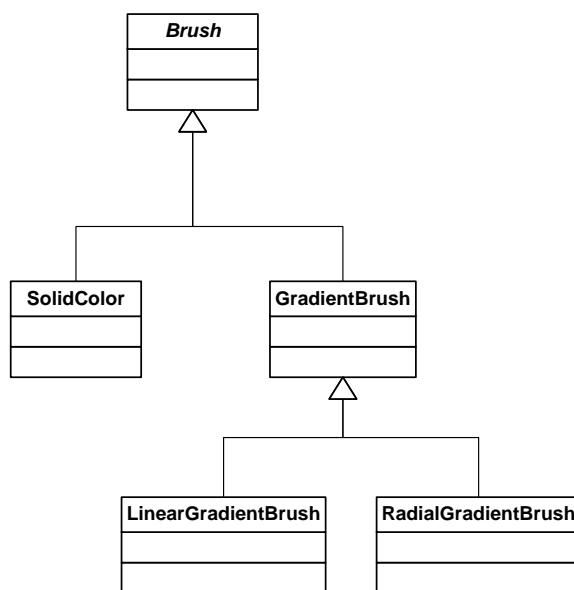


Figura 12 - Modelo UML das cores em WPF

Em XAML, a cor sólida é do tipo ARGB (*Alpha, Red, Green, Blue*), onde *Alpha* determina a transparência da cor e RGB a cor propriamente dita, sendo que o valor utilizado para cada um dos parâmetros é em hexadecimal. Em SVG, a cor também é representada pelos mesmos valores e parâmetros, mas ao contrário do XAML o parâmetro A e os parâmetros RGB são representados separadamente, sendo que os valores para RGB são representados em hexadecimal, mas o valor de A é representado por um valor numérico entre 0 e 1, como tal, para se representar uma cor de XAML para SVG são necessários dois atributos, um para a cor e outro para a transparência.

Os gradientes (ver Figura 12) estão divididos em dois tipos: lineares e radiais. Num gradiente linear, as cores que fazem parte do gradiente mudam suavemente ao longo duma linha, independentemente da sua direcção, enquanto nos gradientes radiais as cores mudam a partir do centro para a periferia, em forma de círculo ou elipse.

Em XAML, tal como em SVG, quer no gradiente linear quer no radial, a direcção pela qual as cores são dispostas pode ser afectada por uma transformada.

No gradiente radial em XAML é possível definir qual o valor do raio do gradiente para o eixo dos XX e para o eixo dos YY, sendo assim possível definir além de círculos, elipses. Em SVG não existe tal possibilidade, o valor do raio em SVG afecta ambos os eixos, ou seja, só é possível definir círculos. Como tal, na conversão optou-se por calcular o valor médio dos raios em WPF, sendo esta uma diferença entre o modelo XAML e o SVG.

3.3.3. Alinhamento

É possível definir o alinhamento horizontal e vertical dum elemento. O alinhamento horizontal permite especificar como é que o texto num objecto está alinhado em relação às margens desse objecto, podendo o texto ter vários tipos de alinhamento: alinhado à esquerda, à direita, centrado e justificado. De todos os alinhamentos suportados pelo WPF o alinhamento justificado não é suportado pelo SVG.

O alinhamento vertical permite especificar como é que um elemento dentro dum objecto é posicionado. O elemento pode ter vários tipos de alinhamento: alinhado em baixo, em cima, topo e esticado. De todos os alinhamentos suportados pelo WPF o alinhamento esticado não é suportado pelo SVG.

3.3.4. *Pen Line Cap*

O *Pen Line Cap* permite definir a forma geométrica no final duma linha ou segmento. Existem 4 tipos diferentes de formas que o fim da linha pode ter: liso, redondo, quadrado e triângulo. De todas as formas geométricas que podem ser definidas em WPF o fim em triângulo não é suportado em SVG.

3.4. Animações

As animações são um elemento bastante importante na interacção dos sistemas SCADA. O *Automation Studio*, dispõe de uma série de animações que permite essa interacção podendo ser associadas a um ou mais objectos XAML. A cada objecto só pode ser atribuída uma animação de cada tipo e para cada animação está associado o nome de uma variável, que corresponde a uma entidade da base de dados, e cujo valor irá ser utilizado para determinar qual o comportamento da animação.

Tabela 6 - Lista de animações suportadas pelo HMI 500

Animação\Conversão SVG	Suportada	Não Suportada
Rotação	X	
<i>Standard</i>	X	
Visibilidade	X	
Escala	X	
Navegação (Acção de Interactividade)	X	
Estilo	X	
Escrita (Acção de Interactividade)		X
Preenchimento		X
<i>Zoom</i>	X	
<i>Declutter</i>	X	
Execução (Acção de Interactividade)	X	
Movimento	X	
Numérico	X	
<i>Linear Slider</i>		X
<i>Rotation Slider</i>		X

O *Automation Studio* dispõe de vários módulos especializados para poder configurar os equipamentos da EFACEC e de terceiros, como por exemplo o módulo de simulação que permite testar

Capítulo 3 – Conversão dos elementos disponibilizados pelo *Automation Studio* para SVG a configuração de certos equipamentos antes de essa configuração ser enviada para os mesmos ou o módulo *Touch* que foi criado para dar suporte ao protótipo *multi-touch* da EFACEC. Como tal, algumas das animações disponíveis na Tabela 6 foram criadas para dar suporte a esses módulos e não estão disponíveis para o HMI 500. Assim sendo, a animação de Escrita só está disponível para o módulo de simulação do *Automation Studio*, as animações *Linear* e *Rotation Slider* só estão disponíveis para o módulo *Touch* do *Automation Studio* e, por uma decisão de desenho, não foram efectuadas as respectivas traduções para SVG.

À altura do desenvolvimento do compilador, a animação de preenchimento não era suportada pelo HMI 500, e, como tal, a tradução para SVG ficou para trabalho futuro, para quando o HMI 500 pudesse suportar a animação.

Cada uma das animações suportadas pelo HMI 500 (ver Tabela 6), têm funções específicas, pelo que foi necessário estudar cada uma delas para entender qual o seu funcionamento, e a sua implementação em SVG.

3.4.1. Estudo da funcionalidade das animações suportadas pelo HMI 500

As animações que são suportadas pelo HMI 500 têm características e funcionalidades específicas de cada uma delas, pelo que, antes de se poder implementar cada uma destas características para SVG, foi necessário determinar qual seria o comportamento esperado de cada uma delas.

Assim, a animação de rotação deve rodar um objecto a partir de um dado centro, a de escala deve redimensionar (escalar) um objecto a partir de um dado centro e, a de movimento permite mudar um objecto de posição, coordenadas (x, y), para outra. Estas animações têm três modos de operação: *Status* (estado), modo em que a animação depende do valor da variável que a anima e *Restartable* (Reinicializável), modo em que a animação repete do início após chegar ao fim; *Reversible* (Reversível) modo em que a animação se faz no sentido inverso quando chega ao fim. As animações Reinicializável e Reversível são feitas de forma automática, durante um período de tempo.

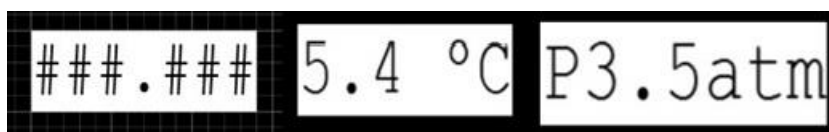


Figura 13 - Diferentes visualizações para o numérico

Existem ainda outros tipos de animação. A animação *standard* é uma animação que pertence ao HMI 500 e permite determinar o estado de uma variável. A animação de visibilidade tem dois modos de funcionamento, *Status* (estado) em que dependendo do valor da variável que o anima, o objecto está ou não visível, e *Blink* (Pisca) em que dependendo do valor da variável o objecto pisca ou não. A animação de navegação permite navegar para outros sinópticos ou para uma determinada área do sinóptico ou ainda abrir páginas *web*. A animação de estilo permite mudar certos atributos de um objecto para um determinado valor da variável, por exemplo, para um determinado valor o objecto pode ter cor de fundo verde e para outro vermelho. A animação de execução permite executar controlos. A animação de *declutter* permite que os objectos só estejam visíveis para certos limites de ampliação. O *Zoom* permite definir em percentagem o limite mínimo e máximo de *zoom* (ampliação) para o qual o sinóptico está visível. O Numérico permite visualizar o valor de uma variável, sendo possível formatar e adicionar informação ao valor visualizado (ver Figura 13). O *Zoom* e o Numérico são elementos (atributo e objecto respectivamente) que, apesar de não serem considerados animações, têm de ser animados para desempenhar a função específica para a qual foram desenvolvidas. Como existe a necessidade de animar estes elementos, estes constam da Tabela 6 (como animações), para ressaltar a interactividade que têm com o utilizador.

3.4.2. Tecnologias para a implementação das animações em SVG

Depois de analisar o que cada animação deveria fazer, foi necessário determinar qual a melhor forma das animações serem traduzidas para SVG e terem o comportamento esperado.

Como foi referido anteriormente, as animações têm um comportamento específico e podem ter vários modos de operação, além de terem de ser integradas para trabalhar com o HMI 500. Para proceder a esta integração foi utilizado o *JavaScript* (Wilton & McPeak, 2007), uma linguagem que é interpretada em vez de ser compilada, e que pode ser adicionada ao SVG, para se obter o comportamento esperado das animações. Além do *JavaScript* foi utilizado o SMIL (*Synchronized Multimedia Integration Language*) (Anon., 2013), uma linguagem utilizada para apresentações multimédia. Embora todas as animações pudessem ser programadas utilizando *JavaScript*, verificou-se que, para se programar as animações com comportamento automático, que são dependentes de um intervalo de tempo, não era a melhor solução, enquanto o SMIL demonstrou ser a linguagem perfeita para este tipo de situações.

3.5. Conclusão

Neste capítulo, foram analisadas as semelhanças entre as linguagens SVG e XAML, assim como as respectivas diferenças. Depois de identificados os elementos divergentes e as suas representações em XAML, desenvolveu-se um estudo que permite a equivalência entre estes elementos e a sua representação em linguagem SVG.

No próximo capítulo, vamos analisar como foi desenvolvido o compilador da linguagem XAML em SVG e a sua integração no *Automation Studio*, bem como o controlo das plataformas da EFACEC que utilizam linguagem SVG.

Capítulo 4 – Ferramenta Desenvolvida

Este capítulo, analisa o processo de tradução do modelo em XAML, para o modelo em SVG, realizado pelo *plugin* desenvolvido. Irá também ser descrito todo o processo de desenvolvimento, nomeadamente, as características da aplicação desenvolvida e utilizada pela EFACEC.

4.1. Arquitectura do SVG

Para desenvolver a ferramenta foi necessário identificar quais as informações que têm de ser traduzidas de XAML para SVG. Para esse estudo, agrupamos as informações em grupos que respeitassem a arquitectura do HMI 500 e que são necessárias para que o ficheiro que vai ser traduzido possa ser correctamente interpretado pelo equipamento.

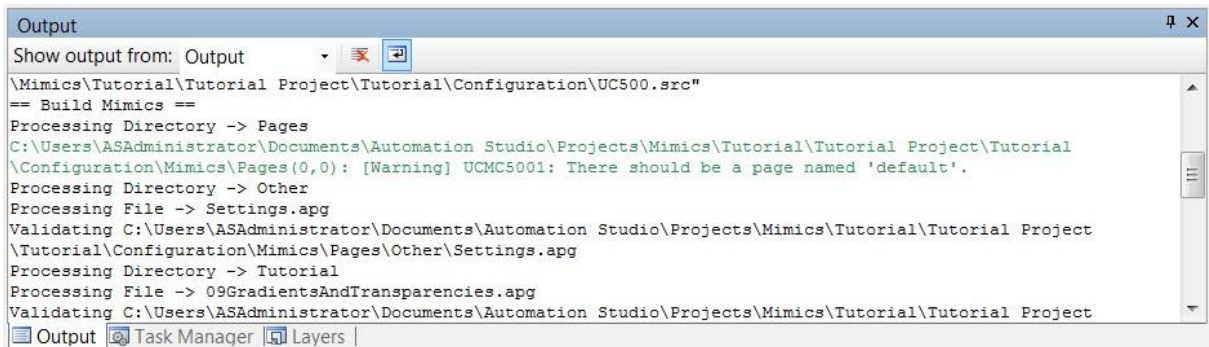
SVG Header
Title
Defs
Animation Layer
Unresizable Layer
Metadata
Scripts de animação

Figura 14 - Estrutura dum ficheiro SVG no HMI 500

Na Figura 14, estão referidos quais os grupos de informações que têm de ser preenchidos e a respectiva ordem pela qual são apresentados (no sentido decrescente) para que exista compatibilidade com o modelo HMI 500. Dentro de cada grupo é reunida a informação necessária para a compatibilidade dos ficheiros, mas é importante destacar o “Animation Layer”, o “Metadata” e os “Scripts de animação” já que é dentro destes grupos que existe o maior processamento de informação por parte do compilador. Nos restantes grupos, o código gerado é necessário para garantir a interoperabilidade entre o HMI 500 e a página (isto permite que o utilizador possa interagir com a página e obter os valores do equipamento que está a monitorizar). Com excepção da alteração de alguns atributos, a informação gerada é praticamente estática.

4.2. Compilador

O compilador XAML-SVG desenvolvido, foi inserido no *Automation Studio* como um *plugin*. Os *plugins* no *Automation Studio* estão acessíveis através de um executável, permitindo que o *plugin* possa ser executado de forma independente do *Automation Studio*. Isto permite, também, que existam diferentes versões do IDE, em que um determinado *plugin* pode ou não estar disponível nessa versão do *Automation Studio*.



```

Output
Show output from: Output
\\Mimics\\Tutorial\\Tutorial Project\\Tutorial\\Configuration\\UC500.src
== Build Mimics ==
Processing Directory -> Pages
C:\\Users\\ASAdministrator\\Documents\\Automation Studio\\Projects\\Mimics\\Tutorial\\Tutorial Project\\Tutorial\\Configuration\\Mimics\\Pages(0,0): [Warning] UCMC5001: There should be a page named 'default'.
Processing Directory -> Other
Processing File -> Settings.apg
Validating C:\\Users\\ASAdministrator\\Documents\\Automation Studio\\Projects\\Mimics\\Tutorial\\Tutorial Project\\Tutorial\\Configuration\\Mimics\\Pages\\Other\\Settings.apg
Processing Directory -> Tutorial
Processing File -> 09GradientsAndTransparencies.apg
Validating C:\\Users\\ASAdministrator\\Documents\\Automation Studio\\Projects\\Mimics\\Tutorial\\Tutorial Project

```

Figura 15 - Compilador para a geração de páginas SVG a correr no *Automation Studio*

O *plugin* XAML-SVG foi criado como um compilador (ver Figura 15), sendo baseado num já existente anteriormente criado pela EFACEC. A sua utilização deveu-se ao facto de a equipa de desenvolvimento da EFACEC utilizar e desenvolver vários compiladores para criar diversas aplicações ou *plugins* para o *Automation Studio*. O compilador foi desenvolvido em C#, utilizando a ferramenta de edição da Microsoft, o *Visual Studio*, e integrado num projecto C# já existente da EFACEC. Utilizou-se inicialmente a *Framework .Net 3*, e mais tarde, actualizado para a *Framework .Net 4*.

O compilador desenvolvido é responsável por converter páginas em XAML para SVG, garantindo que as páginas SVG geradas, são interpretadas correctamente pelo HMI 500. As páginas geradas podem ainda ser comprimidas, ocupando, dessa maneira, menos espaço físico em disco, sendo esta a opção, por omissão, do *Automation Studio*. O compilador, é capaz de determinar se houve ou não alguma mudança na página XAML que vai gerar, e desta maneira saltar a geração da página se tal não for necessário.

Durante a geração da página SVG, o compilador faz tratamento de erros, enviando mensagens de erro para o utilizador, quando existe algum atributo em XAML que não é capaz de converter para SVG ou quando alguma animação não está correcta. Para que uma animação seja correcta, ela precisa de ter um conjunto de parâmetros correctamente preenchidos pelo utilizador no *Automation Studio*. Quando esses parâmetros não são preenchidos, ou estão incorrectos, a animação é ignorada pelo

compilador, uma vez que não é possível ter a animação a funcionar, por esses parâmetros não estarem correctos.

Além de converter a página em SVG, o compilador também é responsável por garantir que as páginas SVG são criadas no local correcto e que qualquer imagem (ficheiros JPEG e outros) é copiada para o destino da página SVG, numa única directoria. Isto permite garantir que a leitura dessas imagens, e da própria página, seja feita a nível local. Com efeito, por decisão da equipa de desenvolvimento da EFACEC, o acesso remoto às páginas, imagens e restante informação, não foi equacionado para evitar sobrecarga da rede e problemas de segurança. Adicionalmente, a opção por manter as imagens locais, apesar de aumentar o tamanho da directoria gerada, permite evitar erros de leitura das imagens. Ou seja, garante que, independentemente do equipamento utilizado, a leitura das imagens, por parte da página SVG, é feita sempre da mesma forma.

Para que o compilador possa executar as operações atrás descritas e outras intermédias, como por exemplo, verificar se a entidade que vai fazer parte da animação existe na base de dados, são passadas informações ao compilador através de vários parâmetros.

```
<?xml version="1.0" encoding="utf-8"?>
<MimicsDefinition version="1.0">
  <Authoring>
    <Tool>Automation Studio Designer Edition 2.6.402.2</Tool>
    <UserName>Nuno</UserName>
    <MachineName>ASUS</MachineName>
    <Date>2013-10-08 06:02:28.0372329</Date>
  </Authoring>
  <References />
  <IncludeSource>true</IncludeSource>
  <Precision>3</Precision>
  <Compress>>false</Compress>
  <CompressionLevel>4</CompressionLevel>
  <ValidateBeforeBuild>true</ValidateBeforeBuild>
  <FadeIn>>false</FadeIn>
  <PagesOrder />
  <CurrentLanguage>en</CurrentLanguage>
  <AvailableLanguages>en</AvailableLanguages>
</MimicsDefinition>
```

Figura 16 - Ficheiro de definições típico utilizado pelo compilador

Esses parâmetros vão desde a localização dos ficheiros XAML, localização para onde os ficheiros SVG são gerados, o modelo do dispositivo onde as páginas estão a ser utilizadas, ficheiro com

definições (ver Figura 16), lista de cores, utilizadas para mostrar a qualidade duma variável nas animações do tipo *standard*, entre outros.

4.2.1. Ficheiro de definições

O ficheiro de definições da Figura 16 contém alguma da informação utilizada pelo compilador no decorrer da tradução, sendo que: O atributo *References* dá o caminho para as bibliotecas, se existirem, caso contrário essa informação é ignorada. A secção *IncludeSource* determina se o directório onde estão as páginas XAML é, ou não, incluído no directório de destino das páginas SVG. *Precision* diz qual o número de casas decimais é utilizado por alguns dos atributos dos elementos SVG, como por exemplo o valor da largura (*width*), ou os valores utilizados pelas matrizes, enquanto *Compress* e *CompressionLevel* dizem respeito às páginas SVG deverem ser comprimidas e qual o nível de compressão utilizado. *ValidateBeforeBuild* indica que deverá ser verificado se a página original contém erros antes de ser gerada, e o *FadeIn* determina se as animações do tipo Pisca (do inglês *Blink*), piscam ou se têm o efeito de desvanecimento. *CurrentLanguage* e *AvailableLanguages* determinam qual a língua e linguagens disponíveis. Neste caso, estes dois atributos são aplicados para os elementos que contenham texto em XAML, o que faz com que uma página possa ser gerada para SVG em múltiplas línguas. Os atributos *Authoring* e *PagesOrder* são ignorados pelo compilador, pelo que não são aqui abordados.

4.3. Processamento

Para efectuar a tradução de XAML para SVG, o compilador processa toda a página em memória, e no fim escreve o resultado para um ficheiro.

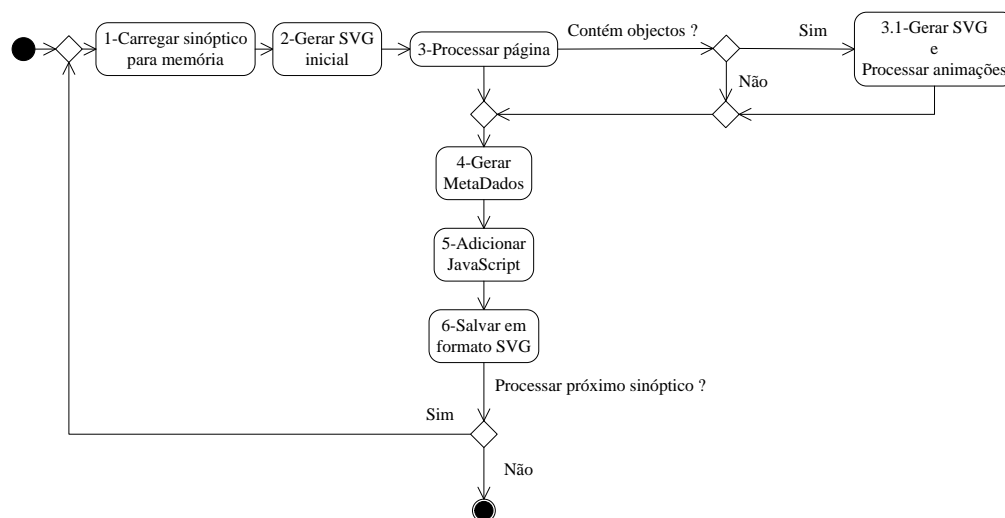


Figura 17 - Passos para a tradução de um ficheiro XAML, para um ficheiro SVG

Conforme podemos analisar na Figura 17, para cada página em XAML são percorridos os passos que serão descritos de seguida e, após executar a tradução de todas as páginas individualmente, o compilador escreve o ficheiro SVG para disco, correspondente a todo o ficheiro XAML.

4.3.1. Carregar sinóptico para memória

O primeiro passo do compilador, passa por carregar o ficheiro XAML para memória para, dessa forma, ter acesso a todo o modelo de mímicos do *Automation Studio*. Assim, são reunidas todas as informações necessárias, relativamente a cada um dos objectos XAML, desde posição que ocupa no ecrã, animações a que possa, ou não, estar sujeito, entre outras informações.

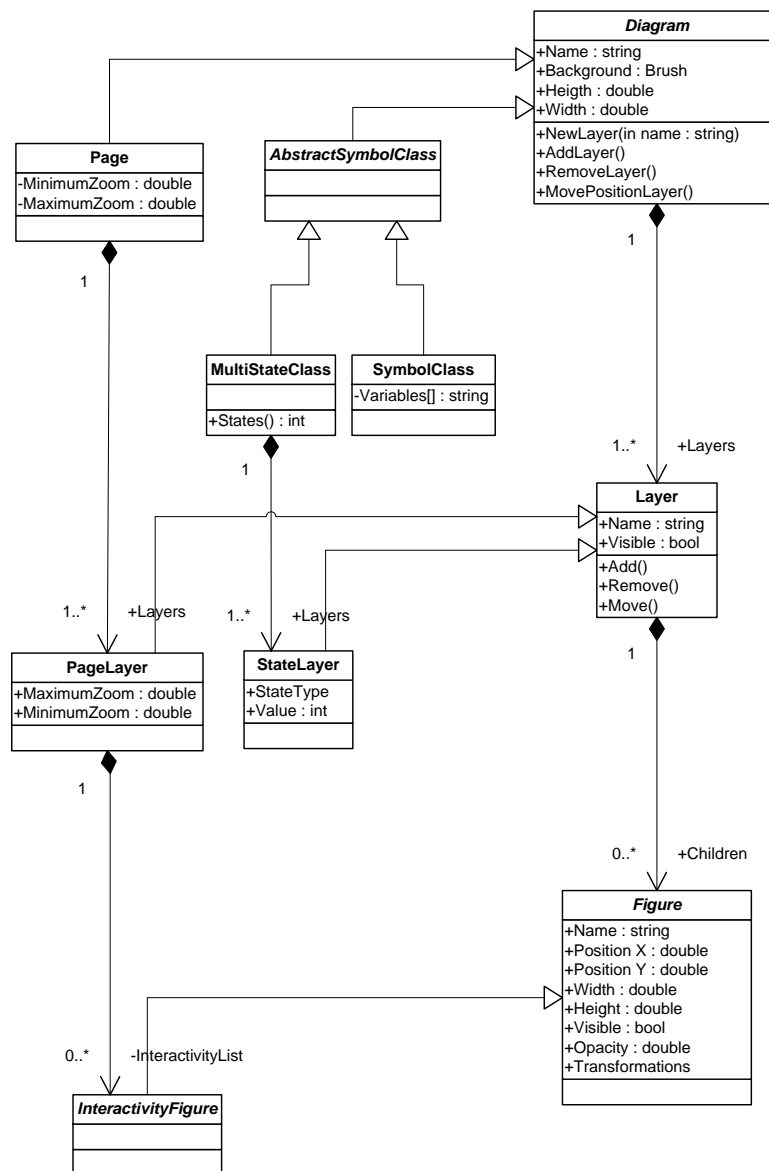


Figura 18 - Modelo UML de uma página em XAML no *Automation Studio*

Analisando o modelo UML da Figura 18, ficamos com uma noção de como os diferentes elementos XAML estão organizados. O compilador percorre, sequencialmente, este modelo para poder traduzir os diferentes elementos da página XAML em SVG. Começa pelo *Diagram* que representa a raiz do modelo, e a partir dele começa a processar a página (*Page*), percorrendo as várias camadas (*Layer*) que a constituem, traduzindo os objectos (*Figure*) e respectivas animações (*InteractivityFigure*) para SVG. Existem ainda os *MultiState* (*MultiStateClass*) e os símbolos (*SymbolClass*), que apesar de fazerem parte da página, são tratados como diagramas (*Diagram*), já que cada um deles pode conter múltiplos objectos, tal como as páginas contém. Durante este processo, só é feita uma recolha de informação, ainda não é efectuada nenhuma tradução de XAML para SVG.

4.3.2. Gerar SVG inicial

Neste ponto, antes de se começar a processar a tradução da página XAML, é iniciada a escrita do ficheiro SVG. A arquitectura do ficheiro SVG tem de respeitar a estrutura referida na Figura 14, em que, os primeiros 3 grupos de informação (*SVG Header*, *Title*, *Defs*), garantem a compatibilidade com o HMI 500. Estes grupos de informação são comuns em todas as páginas SVG traduzidas.

```
<svg xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:synoptic="http://www.efacec.pt/synoptics/" version="1.2" width="100%"
height="100%" preserveAspectRatio="xMidYMid meet" viewBox="0 0 1545 800"
onload="Initialize(evt)" font-family="Arial, Helvetica, Tahoma, sans-serif" font-
size="11" xml:space="preserve" xmlns="http://www.w3.org/2000/svg">
```

Figura 19 - Cabeçalho SVG para as páginas geradas pelo compilador

No grupo *SVG Header* (ver Figura 19), que corresponde ao cabeçalho de qualquer documento SVG. Também são definidos alguns parâmetros para serem utilizados de forma global pelo documento. Por exemplo, o tamanho e os tipos de letra a serem utilizados nos elementos de texto, e uma variável global (*synoptic*, ver Figura 19), para aceder a funcionalidades criadas pela EFACEC para o HMI 500.

O *Title* corresponde, neste caso, ao nome da página que está a ser gerada. Aparece na janela onde a página está a ser visualizada e ajuda, a que mais facilmente, o utilizador consiga identificar as diversas páginas que podem estar abertas no HMI 500.

Em *Defs*, é adicionada informação visual, estilos CSS (*Cascading Style Sheets*) e gradientes, que depois são utilizados pelo HMI 500, para as animações *Standard*. É neste ponto que a lista de cores (referida no ponto 4.2), passada para o compilador como um parâmetro, é utilizada pelo mesmo.

4.3.3. Processar página

Após inserir o código SVG inicial no documento, o compilador começa a analisar e a processar a página. Uma página pode conter uma ou mais camadas (*Layer*), que pode conter vários objectos XAML (ver Figura 18). O compilador processa a página, camada a camada, processando uma camada, e seus objectos, de cada vez. Os objectos são processados individualmente, convertidos para SVG (segundo a Tabela 3), e os diversos atributos que o compõem, são processados, de forma a serem compatíveis com o SVG.

Na linguagem SVG é possível adicionar um identificador aos objectos, e esta funcionalidade permite que um determinado elemento possa ser encontrado na página. O valor do identificador tem de ser único, isto para que, um dado objecto, possa referenciar outro, e o compilador garante esta situação. O identificador é utilizado pelo compilador para objectos com animações. Também será utilizado mais tarde para fazer a ligação aos *MetaDados* (variáveis da base de dados), bem como para identificar gradientes e delimitadores de área (*clip-path*), que depois serão utilizados pelo elemento SVG que dele necessite.

Como já foi explicado, a tradução de vários objectos XAML para SVG não é directa. Este, é mais um dos motivos pelo qual, durante o processamento da tradução, o compilador passa todo o conteúdo do ficheiro XAML para memória e, só depois, efectua a escrita em disco do ficheiro SVG. Durante a leitura de cada elemento XAML o compilador procura a sua correspondência em SVG e as “adaptações” que serão necessárias para não alterar as suas características. Vamos analisar a quantidade de código necessário para representar um objecto, como um botão clássico, nas duas linguagens.

```
<path d="M224,80 L416,80 L406,90 L234,90 L234,150 L224,160 z" fill="#B2B2B2" id-
dummy="LightColor" fill-dummy="#B2B2B2" fill-opacity-dummy="1" />
<path d="M234,150 L406,150 L406,90 L416,80 L416,160 L224,160 z" fill="#E5E5E5" id-
dummy="DarkColor" fill-dummy="#E5E5E5" fill-opacity-dummy="1" />
<rect x="234" y="90" width="172" height="60" opacity="1" fill="#FFFFFF" fill-
opacity="1" />
<clipPath id="ID2">
  <path d="M 234 90 L 406 90 L 406 150 L 234 150 z" />
</clipPath>
<text text-anchor="middle" x="320" y="127.855" font-family="Courier New" font-
size="24" font-style="normal" font-weight="400" font-stretch="normal"
fill="#000000" fill-opacity="1" clip-path="url(#ID2)">OK</text>
```

Figura 20 - Código SVG gerado para o botão clássico

```
<ClassicButton BevelWidth="10" Background="#FFFFFF" Foreground="#FF000000"
FontFamily="Courier New" FontSize="24" HorizontalContentAlignment="Center"
Width="192" Height="80" av:Canvas.Left="224" av:Canvas.Top="80">OK</ClassicButton>
```

Figura 21 - Código XAML para o botão clássico

A Figura 21 representa o código XAML original criado pelo *Automation Studio*, enquanto a Figura 20 representa o mesmo objecto traduzido para SVG. Como podemos observar, existem mais elementos SVG do que XAML para representar o mesmo objecto, isto porque, não existem botões clássicos nativos em SVG, e como tal, é necessário desenhar todos os aspectos do botão para que tenha a mesma aparência que em XAML.

Os dois elementos *path* do SVG, representam o efeito 3D do botão clássico. Para se obter esse efeito, o compilador teve de calcular, a partir da cor original, uma cor mais clara e outra mais escura, para criar o efeito de interação com o utilizador de que o botão estava ou não activado. Essas cores foram obtidas ao serem manipuladas algumas características, como a intensidade do brilho a partir da cor original. Mais concretamente este efeito foi obtido convertendo a cor RGB em HSL (Tonalidade/Saturação/Luminosidade, do inglês *Hue/Saturation/Lightness*), em que se aplica uma luminosidade de +/- 30% (o mesmo valor utilizado pelo objecto em XAML) e convertendo o resultado novamente para RGB, para poder ser interpretado correctamente em SVG. No código SVG pode-se ainda observar a utilização de identificadores, neste caso, para a utilização de um delimitador de área (*clip-path*), que é utilizado pelo texto, para garantir que não ultrapassa a área do botão.

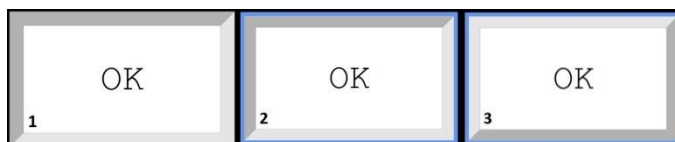


Figura 22 - Estados para os botões clássicos

```
<g id="ID2">
...
<a cursor="pointer" onclick="evt.preventDefault()">
<rect x="319.5" y="159.5" width="161" height="81" fill="#FFFFFF" fill-opacity="0"
stroke="#FFFFFF" stroke-opacity="0" onmouseout="StdButtonAnim(evt, false, false,
false);ClassButtonAnim(evt, 'ID2', false);"
onmouseup="ClassButtonAnim(evt, 'ID2', false);navigate(evt, 'P211.svg',1,1, false)"
onmouseover="StdButtonAnim(evt, true, false, false);"
onmousedown="ClassButtonAnim(evt, 'ID2', true);" />
</a>
</g>
```

Figura 23 - Código SVG para animar o botão clássico

O compilador ao processar os botões, caso estes tenham animações (ver Capítulo 3.4) *standard* ou *execute* ou acções de navegação, adiciona animações visuais, para mostrar aos utilizadores que existe uma acção associada ao botão. Como tal, foi adicionado código extra (ver Figura 23), ao código SVG já existente (ver Figura 20). Esse código extra permitiu adicionar uma animação de selecção (ver Figura 22), mudar o tipo de cursor do rato, quando este está por cima do elemento e dar a ilusão que o botão está a ser carregado quando é accionado. Na Figura 22, podemos ver quais os estados possíveis do botão:

1. Representa o botão não seleccionado;
2. Representa o botão seleccionado;
3. Representa o botão seleccionado e carregado.

4.3.4. Gerar *MetaDados*

O compilador, após concluir o processamento da parte visual da página, adiciona os *MetaDados* à página SVG a ser gerada. Os *MetaDados* são uma estrutura que contém todas as referências das animações, e dos objectos animados, e permitem fazer a ponte entre o objecto a ser animado, a variável na base de dados de acordo com a qual o comportamento da animação varia (como foi explicado no capítulo 3.4) e o *JavaScript* utilizado para fazer a animação.

```
<metadata>
<devices>
<device graphicsId="STDEXEC2">
<entities />
<controls>
<control logicalId="O$execute" silent="false" value="30" />
</controls>
<observers />
</device>
<device graphicsId="ANIMA3">
<entities formula="''+ translateAnimation(50,200,300,159,$1)+ ''">
<entity logicalId="A$movement" />
</entities>
<controls />
<observers>
<observer type="attribute" targetTag="g" targetAttribute="transform" />
</observers>
</device>
</devices> <decluttering /> </metadata>
```

Figura 24 - Estrutura de *MetaDados* típica utilizado para as animações na página SVG

O código apresentado na Figura 24, representa um exemplo de utilização dos *MetaDados*. Neste caso em particular, *graphicsId="STDEXEC2"* representa uma animação de execução, e *graphicsId="ANIMA3"* representa uma animação de movimento. O valor do atributo *graphicsId*, corresponde ao identificador que foi utilizado pelo compilador para animação durante o processamento da página. O valor do atributo *logicalId* corresponde ao nome da variável utilizada na base de dados, cujo valor vai ser utilizado para determinar qual o comportamento da animação.

Como foi referido, *ANIMA3* representa uma animação de movimento, em que o valor do atributo *formula* corresponde a uma acção que irá proceder à animação, utilizando linguagem *JavaScript*. O mesmo processo é utilizado para as animações de rotação, escala e de estilo.

4.3.5. Adicionar *JavaScript*

O comportamento das animações nativas em SVG é sempre igual, independentemente do valor da variável a que estejam associadas, mas esse comportamento, por vezes, não responde às necessidades específicas de cada objecto. Para resolver esta questão, é necessário adicionar código *JavaScript*, à página SVG gerada, que permite que cada objecto tenha determinado comportamento, em função do valor da variável. Com isto, cada objecto pode ter diversas animações e comportamentos, de acordo com os valores das variáveis associadas.

4.4. Integração no *Automation Studio*

Tal como já referido, o objectivo desta ferramenta é a sua integração no *Automation Studio* como um compilador automático de ficheiros XAML para SVG. Para garantir o sucesso desta integração, foram efectuados diversos testes de tradução utilizando ficheiros XAML de projectos reais, ficheiros XAML criados para efeitos de teste e ficheiros SVG já existentes (o *Automation Studio* tem uma ferramenta que traduz os ficheiros SVG, gerados por uma ferramenta de edição da EFACEC, para XAML), neste caso, o ficheiro SVG traduzido teria de ser igual ao SVG inicial.

Em primeiro lugar, foram traduzidos ficheiros XAML criados no *Automation Studio* apenas para testar a eficácia da tabela de equivalências dos elementos e animações. Após este processo e constatado o sucesso do mapa de equivalências, foram utilizados ficheiros criados aquando do desenvolvimento da plataforma HMI 500, pela equipa de desenvolvimento da EFACEC. Numa fase seguinte, foram utilizados ficheiros de projectos reais que, em conjunto com a equipa de testes da

EFACEC, eram traduzidos para SVG, sendo o seu funcionamento testado nos equipamentos da EFACEC. Em simultâneo com estes procedimentos, foi utilizada uma ferramenta do *Automation Studio* que traduz os ficheiros de SVG, de outra plataforma da edição gráfica da EFACEC, para XAML e comparados os respectivos resultados.

Quando a ferramenta desenvolvida comprovou, em vários níveis de teste, a sua eficácia foi integrada no *Automation Studio*. Ou seja, durante a fase de testes, o *plugin* funcionou como uma outra funcionalidade do *Automation Studio* numa versão Beta (de testes). Numa versão final, foi assumido como uma funcionalidade real do *Automation Studio* e comercialmente disponível na versão *Automation Studio 2.0*, mantendo-se como uma das funcionalidades disponíveis nas versões seguintes.

4.5. Conclusão

Neste capítulo, ficou descrito o processo de desenvolvimento do *plugin* que permite a tradução de XAML para SVG. Esta ferramenta, funciona como uma parte integrante do *Automation Studio*, aplicação da EFACEC que configura e gere vários dos seus equipamentos, resolvendo a questão da interoperabilidade entre as tecnologias que utilizam diferentes linguagens (XAML e SVG).

No próximo capítulo, serão apresentados dois exemplos de utilização real da ferramenta, bem como as vantagens da utilização do ficheiro traduzido.

Capítulo 5 – Exemplos

Neste capítulo, iremos apresentar exemplos da utilização da ferramenta criada. Serão apresentados dois casos de utilização real. Um, que está a ser utilizado na subestação de Ermesinde e outro, na *Electricity and Water Authority Utility* do *Bahrain*. As páginas SCADA criadas no *Automation Studio* (em linguagem XAML) são utilizadas pelas plataformas das duas subestações em análise, através do processo de tradução da ferramenta desenvolvida (de XAML para SVG).

5.1. Subestação de Ermesinde

A subestação de Ermesinde, pertencente às Redes Energéticas Nacionais (REN), fica localizada na freguesia de Ermesinde, concelho de Valongo, distrito de Porto. É uma instalação que contém equipamentos de distribuição, protecção e controlo de energia eléctrica de alta potência.

Recentemente a REN procedeu a uma actualização da subestação, para a qual a EFACEC contribuiu com a implementação de sistemas de controlo e protecção, para além da implementação da plataforma de automação, o CLP 500SAS (EFACEC, 2012).

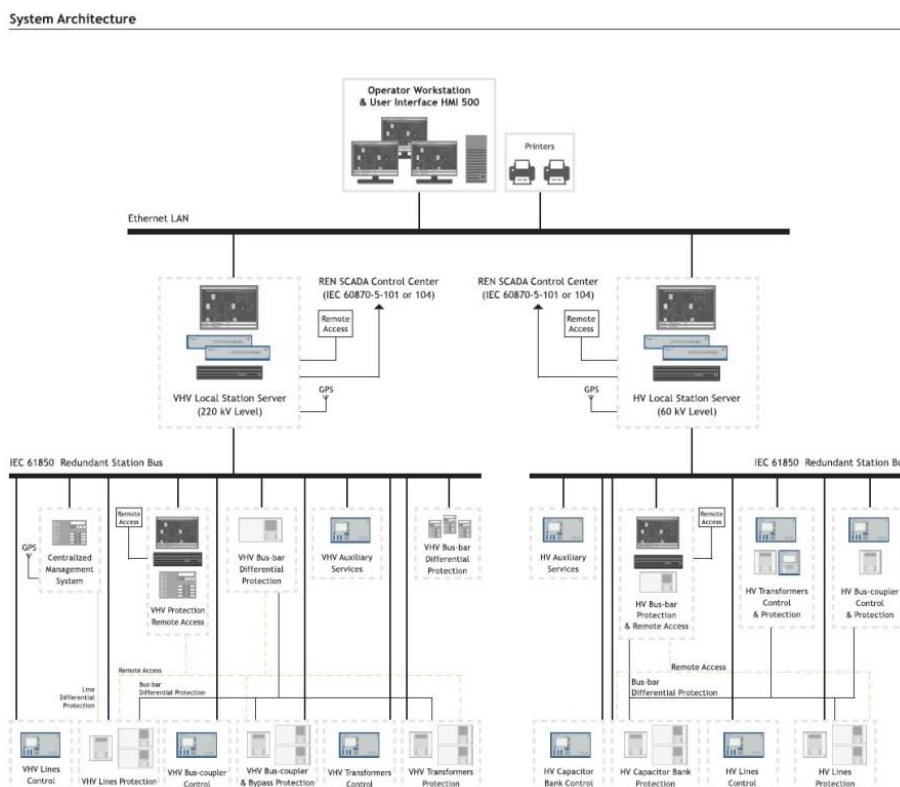


Figura 25 - Arquitectura do sistema montado na subestação de Ermesinde

Além da plataforma CLP 500SAS, a subestação (ver Figura 25) inclui para controlo e protecção, 18 baías, incluindo linhas, transformadores e barras de cobre para níveis de voltagem e bancos de baías de condensadores utilizados para a correcção do factor de potência (EFACEC, 2011).

A implementação, configuração e teste da plataforma CLP 500SAS foi feita pela equipa da EFACEC automação, recorrendo à utilização do *Automation Studio*, que permitiu configurar, não só, cada um dos dispositivos, mas também toda a interface HMI e interacção com utilizador através dos sinópticos.

Devido ao elevado número de dispositivos instalados (ver Figura 25) os sinópticos criados para os monitorizar são muito complexos, não só no número de elementos que compõem cada página, como também no número de animações e entidades utilizadas para essas mesmas animações.

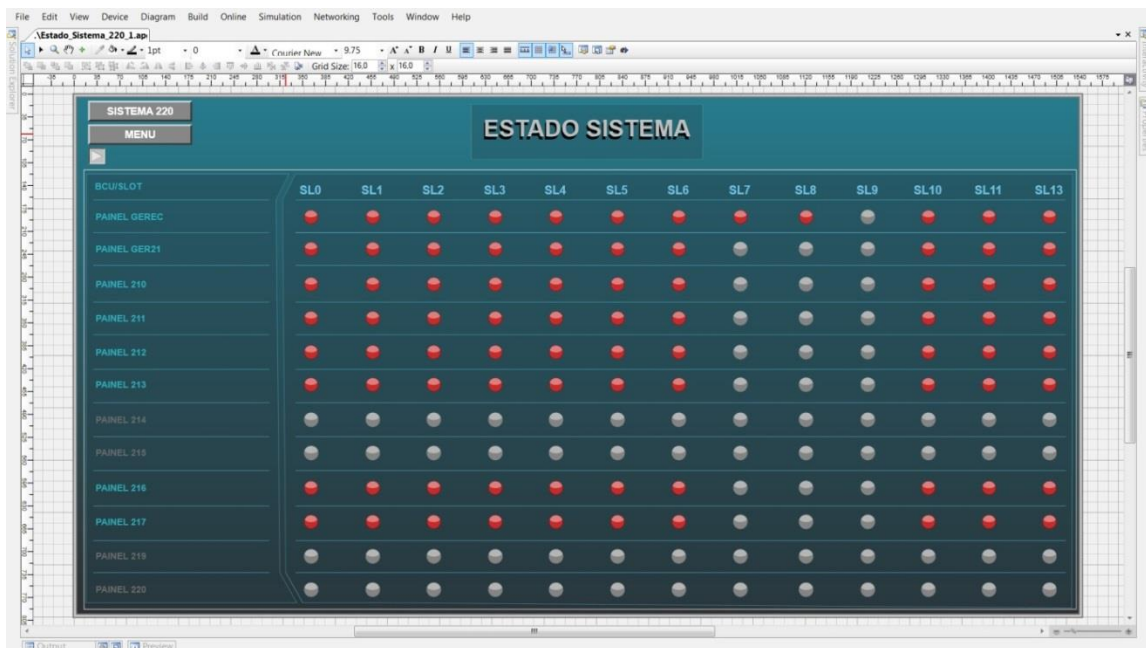


Figura 26 - Página “Estado do sistema” no *Automation Studio*

O projecto criado para a subestação utiliza múltiplas bibliotecas de objectos, que depois são utilizadas nas diferentes páginas. É composto por cerca de 30 páginas, com diferentes complexidades, que vão desde 26 a mais de 7000 elementos XAML numa única página. Os elementos XAML dessas páginas são dos mais variados, sendo que a página mais complexa contém 2161 rectângulos, 2280 elipses, entre outros objectos, para a compor.

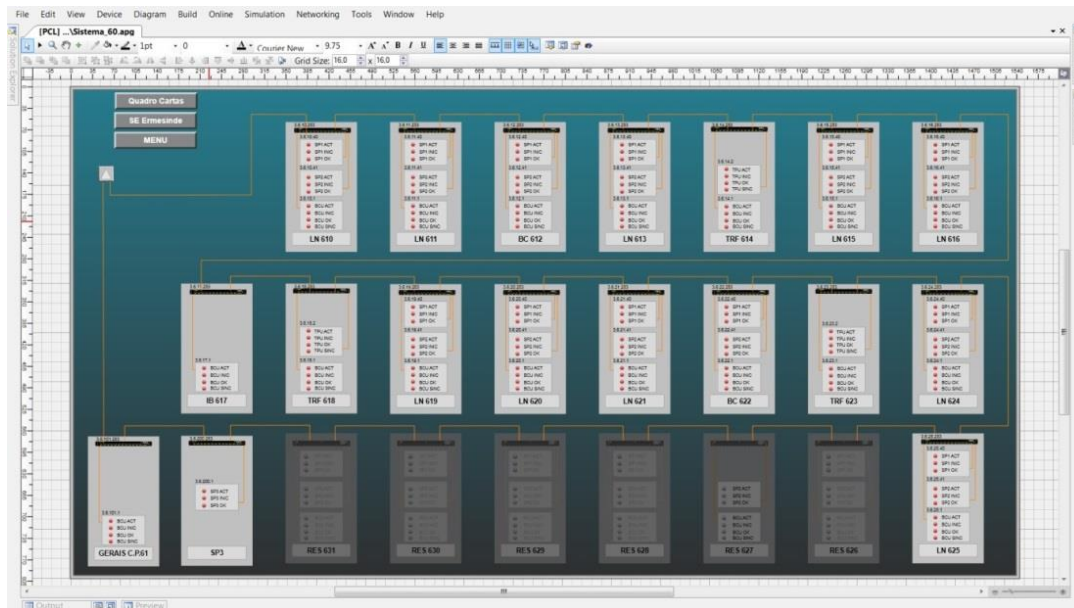


Figura 27 - Página “Sistema 60” no *Automation Studio*

Para além dos elementos geométricos que compõem as páginas, as páginas são interactivas e, como tal, muitos desses elementos são animados. O número de animações por página vão desde 4 a mais de 1500, com centenas de entidades diferentes (o projecto utiliza mais de 6700 entidades, sendo que nem todas são utilizadas para as animações nas páginas) utilizadas para animar os elementos XAML e interagir com os dispositivos.

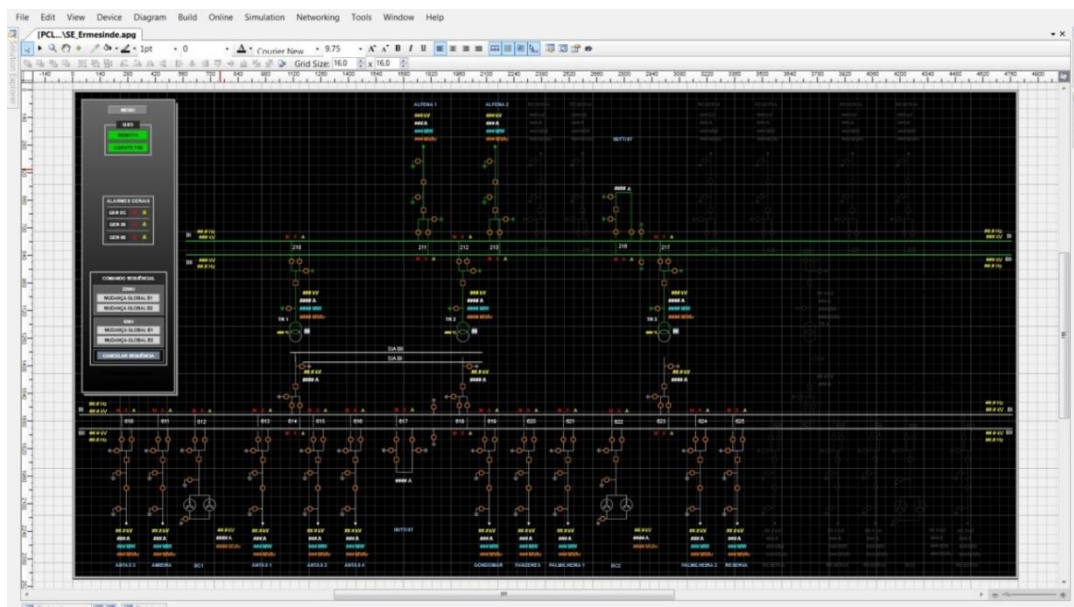


Figura 28 - Ermesinde no *Automation Studio*

Das 30 páginas que compõem o projecto, iremos analisar três delas (ver Figura 26, Figura 27 e Figura 28), e assim ter uma ideia, não só, da complexidade de cada página, como também, do número e tipo de elementos necessários para criar a mesma página em SVG. Cada uma destas

páginas permite ter uma visão global do estado dos equipamentos da subestação em tempo real, permitindo assim obtenção mais específica sobre o estado de determinados equipamentos que monitorizam.

Tabela 7 - Número de elementos e animações necessário para traduzir as páginas em XAML

Elemento\Página	Figura 26		Figura 27		Figura 28	
	XAML	SVG	XAML	SVG	XAML	SVG
Rectângulos	7	312	2161	4176	38	2769
Elipses	74	238	2280	2590	87	562
<i>Paths</i>	74	419	72	2160	0	1145
Linhas	27	35	479	487	912	914
Polígonos	0	3	24	27	4	4
Polilinha	2	2	673	673	18	22
Gradientes do tipo linear	1	4	1	4	1	4
Matrizes	159	172	270	271	118	248
Grupos	150	744	263	1415	289	4725
Texto	30	34	1382	1386	512	612
Arcos	0	0	192	0	0	0
Total de elementos	524	1963	7797	13189	1979	110005
Animação de Navegação	3	7	21	90	141	677
Animação de Estilo	0	0	0	0	0	0
Animação MultiState	82	85	267	173	1543	525
Animação <i>Standard</i>	0	0	0	0	0	0
Animação de Execução	0	0	0	0	0	0
Animação Numérica	0	0	0	0	0	0
Animação de Visibilidade	0	0	0	0	0	0
Total de animações	85	267	173	1543	525	
Número de entidades	82	85	267	173	1543	525

A Tabela 7 apresenta alguns dados sobre qual o número de elementos SVG e animações que são necessárias para traduzir cada uma das três páginas para SVG. A página “Sistema 60” (ver Figura 27) contém 7797 elementos XAML, para os quais o compilador precisou de gerar 13189 elementos SVG, além de 267 animações, que são geridas por 173 entidades diferentes (não é necessário existir o mesmo número de entidades para o mesmo número de animações) para a interactividade da página. A página “Ermesinde” apresenta uma complexidade superior, já que para “apenas” 1979 elementos XAML, foram precisos 110005 elementos SVG, e foi preciso animar 1543 animações geridas por 525

entidades. Mesmo a página com menos elementos, “Estado do Sistema” (ver Figura 26) precisou de 1963 elementos SVG para 524 elementos XAML, com cerca de 85 animações.

Como foi referido anteriormente, este projecto da subestação é composto por mais de 30 páginas SCADA desenvolvidas pelo *Automation Studio*, e cada página composta por milhares de elementos, cada um com o seu grau de complexidade. Apesar dessa complexidade, o compilador foi capaz de gerar todas as páginas para SVG em apenas 56 segundos, o que dá uma média de menos de 2 segundos por página. Uma tradução manual das páginas, demoraria incomparavelmente mais tempo. Com este cenário, podemos comprovar a necessidade e a importância de uma ferramenta como a que foi desenvolvida, capaz de traduzir linguagem XAML em SVG. A eficácia que o *plugin* demonstrou ao fazer a tradução dos ficheiros XAML para SVG com a equipa de testes e desenvolvimento da EFACEC, permitiu que a configuração do equipamento fosse realizada de forma bastante rápida.

O compilador, para além de gerar as páginas, funciona como um corrector de “erros” de configuração dessas páginas, já que, se os dados não respeitarem os valores definidos, lança alertas para que o utilizador possa corrigir os campos assinalados. Existe um aumento do número de objectos do ficheiro XAML para o ficheiro traduzido SVG (conforme explicado no ponto 3.2 da tabela de equivalência de elementos) que é difícil de quantificar porque depende directamente do tipo de elementos que são utilizados na criação das páginas SCADA no *Automation Studio*. Este aumento, que se pode observar nos resultados apresentados na Tabela 7, tem como consequência directa, o aumento do tempo de carregamento da página pela plataforma (neste caso pelo HMI 500). No entanto, não restam dúvidas que este “senão” é completamente aceitável, tendo em conta as suas funções e mais-valias.

5.2. Subestação do *Bahrain*

A EFACEC fornece tecnologia para a expansão da rede eléctrica do *Bahrain*. Um conjunto de 50 subestações, todas elas equipadas com sistemas integrados de automação, protecção e controlo da EFACEC.

Os sistemas são baseados na plataforma CLP 500 e, exactamente como na subestação de Ermesinde, a configuração dos dispositivos foi feita através da utilização do *Automation Studio*, assim como a configuração da interface e interacção HMI com o utilizador, através dos sinópticos.

Trata-se dum projecto de grande envergadura, com um número elevado de subestações e com um número elevado de dispositivos para monitorizar. O projecto para a subestação contém cerca de 60 páginas e foi traduzido pela ferramenta desenvolvida para o *Automation Studio*, em cerca de 58 segundos. Como no exemplo anterior, os sinópticos são muito complexos, com páginas que contêm desde 31 a mais de 3000 elementos XAML. As páginas geradas contêm um elevado número de elementos SVG que vão desde 100 a mais de 8000, e animações, com páginas que contêm mais de 1200 animações para a interacção com o utilizador, utilizando centenas de entidades. Este projecto em particular, tem mais de 3000 entidades, para as animações e configurações dos dispositivos.

Deste projecto massivo de 60 páginas, iremos fazer uma análise de três delas (ver Figura 29, Figura 30 e Figura 31). Analisaremos a complexidade de cada uma das páginas, e qual o número de elementos XAML e SVG que as compõem. As páginas criadas permitem monitorizar e ter uma visão global sobre o estado da subestação e, em particular, os dispositivos que a controlam.

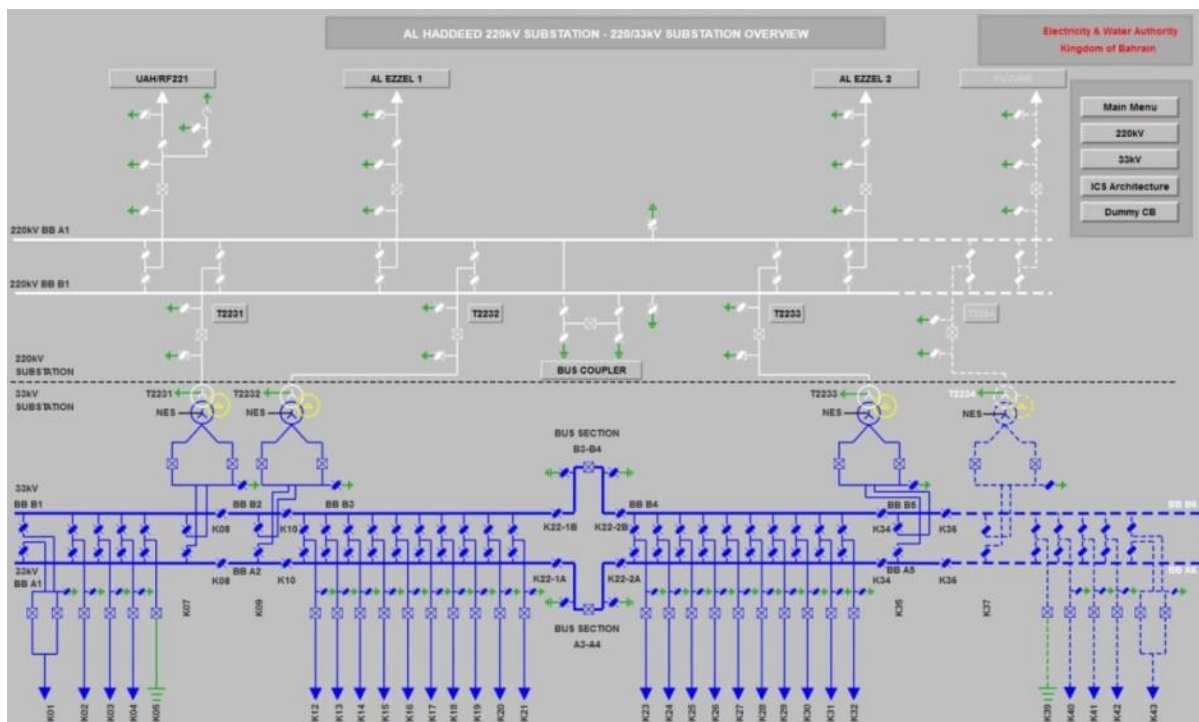


Figura 29 - Sinóptico para a visão global da subestação

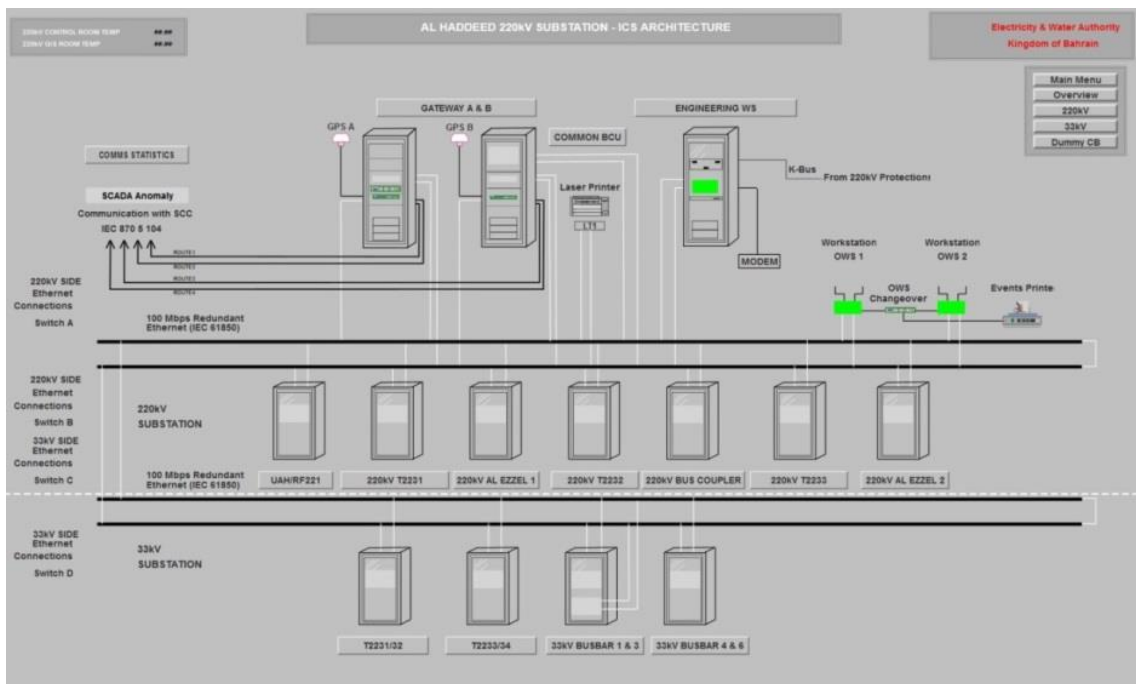


Figura 30 - Sinóptico da arquitectura da subestação

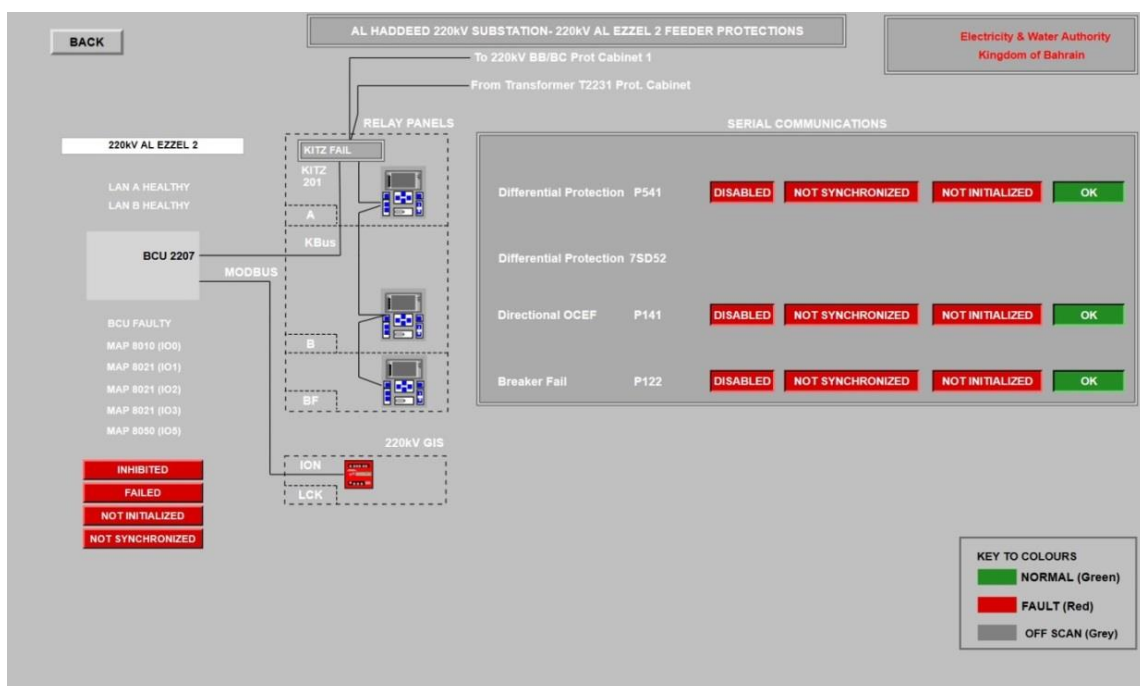


Figura 31 - Sinóptico das protecções para a subestação

A página “Sinóptico para a visão global da subestação” (ver Figura 29), dá-nos uma visão global de uma parte da subestação, e o utilizador, através deste sinóptico, tem acesso à informação indispensável sobre todo o funcionamento da subestação.

Tabela 8 - Número de elementos SVG e animações necessário para traduzir as páginas em XAML

Elemento\Página	Figura 29		Figura 30		Figura 31	
	XAML	SVG	XAML	SVG	XAML	SVG
Rectângulos	95	2215	92	639	103	334
Elipses	12	12	60	64	36	60
<i>Paths</i>	1	634	0	241	0	182
Linhas	1017	1511	320	336	56	58
Polígonos	32	161	0	0	0	0
Polilinha	0	43	0	13	6	8
Imagens	1	1	9	55	29	31
Gradientes do tipo linear	0	3	0	3	0	3
Matrizes	390	630	27	92	44	49
Grupos	1432	3262	477	799	124	298
Texto	93	97	76	84	63	67
Arcos	2	0	0	0	0	0
Total de elementos	3075	8569	1061	2326	461	1090
Animação de navegação	12		20		1	
Animação MultiState	178		27		2	
Animação <i>Standard</i>	137		99		36	
Animação de execução	0		0		0	
Animação de visibilidade	274		0		18	
Animação numérica	0		2		0	
Animação de estilo	0		97		16	
Número de animações	601		245		73	
Número de entidades	315		76		26	

Analisando a Tabela 8, para a Figura 29, podemos ver que se trata de um sinóptico, contendo 3075 elementos XAML, os quais foram traduzidos para 8569 elementos SVG. Foram ainda criadas 601 animações, geridas por 315 entidades, para interacção com o utilizador.

Para a traduzir a página “Sinóptico da arquitectura da subestação” (ver Figura 30), para os 1061 elementos XAML (ver Tabela 8) que compõem o sinóptico, foram precisos 2326 elementos SVG, bem como 245 animações geridas por 76 entidades. Esta página mostra-nos informação com um carácter mais visual, já que no sinóptico foi desenhado um esboço da plataforma CLP 500, e o utilizador tem assim, uma visão mais aproximada acerca do dispositivo ou dispositivos que está a monitorizar pois além da identificação através do código do equipamento tem informação visual. Além

dessa informação visual, os sinópticos ainda contêm acções de navegação para a respectiva plataforma, o que dá uma visão mais geral do dispositivo e permite uma navegação simples até equipamentos ligados entre si. Existe ainda uma animação numérica que dá informação acerca da temperatura da sala onde estão inseridos os dispositivos. Apesar desta informação da temperatura da sala não ser das principais no que diz respeito ao controlo e configuração dos equipamentos e dispositivos, foi acrescentada de forma a ajudar a gerir a subestação.

O sinóptico da Figura 31, “Sinóptico das protecções para a subestação”, contém cerca de 461 elementos XAML (ver Tabela 8), sendo necessários 1090 elementos para recriar a página em SVG, auxiliado por 73 animações, geridas por 26 entidades. Esta página mostra a utilização de botões clássicos, um elemento XAML, que não existe em SVG e como tal foi necessário não só recriá-lo, utilizando outros elementos SVG (ver Tabela 3), como também foi necessário animá-lo, para que o seu comportamento fosse semelhante ao dos botões que são utilizados pelo *Windows 95*.

Com este exemplo, podemos comprovar que não existem limites para o desenvolvimento das páginas SCADA e a sua tradução para SVG. A ferramenta permite a correta e quase imediata tradução (cerca de 58 segundos para os 60 sinópticos corresponde a menos de 1 segundo por página) dos elementos XAML para SVG deste projecto, com a única desvantagem a ser a quantidade de elementos SVG que são necessários para representar todos os elementos XAML.

5.3. Conclusão

Neste capítulo, foram descritas duas situações reais onde a ferramenta desenvolvida, aplicada como *plugin* do *Automation Studio*, é utilizada pela EFACEC como uma solução de sucesso para a tradução da linguagem XAML em SVG. Ficaram demonstradas as vantagens da integração deste componente no *Automation Studio* e as múltiplas aplicações possíveis na gestão de equipamentos que utilizem as linguagens XAML e SVG sem prejuízo da sua eficácia, complexidade e dimensão. Isto foi conseguido, com custos irrisórios e a possibilidade de alargar a incorporação de equipamentos de outras empresas que utilizem linguagens semelhantes, em sistemas geridos pela EFACEC.

Capítulo 6 – Conclusões

O trabalho desenvolvido nesta dissertação enquadra-se no âmbito do projecto InPACT, um projecto de inovação e desenvolvimento, promovido pela EFACEC em parceria com a EDP Distribuição e a Universidade do Minho. O objectivo do projecto foi o desenvolvimento dum conjunto de ferramentas de engenharia avançadas (um ambiente de desenvolvimento integrado – IDE – o *Automation Studio*) para aplicação a sistemas de protecção, automação e controlo de sistemas eléctricos de energia, tendo em vista uma gestão completa dos sistemas da EFACEC. Uma das ferramentas do IDE é um editor de sinópticos, desenvolvido em C# e XAML, que é utilizado para programar os sistemas SCADA da EFACEC. Como os sistemas SCADA utilizam SVG para visualizar os sinópticos, e essa linguagem é incompatível com o XAML, foi necessário criar uma ponte entre o editor do *Automation Studio* e os sistemas SCADA. O trabalho desenvolvido passou então (ver Capítulos 3 e 4) pelo desenvolvimento de uma ferramenta que fizesse a conversão de ficheiros XAML em SVG, de forma a permitir a interoperabilidade entre o *Automation Studio* e os equipamentos SCADA da EFACEC.

Nesse sentido, a ferramenta desenvolvida teve de ser concebida e implementada como um *plugin* para o IDE *Automation Studio*, de forma a poder ser integrada nas ferramentas de desenvolvimento da empresa. Após a integração definitiva do *plugin* no *Automation Studio*, este foi utilizado com sucesso em diversos projectos reais desenvolvidos pela empresa, com destaque para as subestações de Ermesinde e do *Bahrain*, conforme descrito no Capítulo 5 – Exemplos.

Apesar de existirem diferenças entre as linguagens XAML e SVG, a solução desenvolvida conseguiu colmatar essas diferenças, e fazer com que não existisse diferenças visíveis entre a página criada e a página traduzida. Conseguiu ainda, dotar a página criada, com elementos interactivos, que são característicos dos sistemas SCADA. Os sinópticos criados pelo compilador são ligeiramente mais pesados, em termos de tamanho, que os ficheiros XAML criados pelo *Automation Studio*. Os sinópticos criados podem ser melhorados, quer em termos de tamanho, quer em termos de qualidade dos desenhos criados. Note-se no entanto que estas melhorias são, em grande medida, incompatíveis entre si. Se se aumentar a qualidade dos desenhos, aproximando-se da forma real do equipamento que estão a descrever, o ficheiro criado irá aumentar em tamanho. Se se pretender diminuir o tamanho dos ficheiros, isso poderá ser conseguido através de desenhos mais simples, o que irá implicar uma diminuição da qualidade dos sinópticos.

Numa versão futura do compilador terá de ser decidida qual o principal objectivo da aplicação. Privilegiar a componente técnica do desempenho do HMI 500, com preocupações ao nível do peso do ficheiro SVG para beneficiar e aumentar o desempenho do equipamento, ou privilegiar a interacção com o utilizador, ao desenvolver sinópticos mais completos com desenhos próximos do real do equipamento. Qualquer uma destas opções é viável, mas altera, a abordagem e o desenvolvimento das versões seguintes do compilador.

Bibliografia

Allen, K. S., 2006. *Programming Windows Workflow Foundation: Practical WPF Techniques and Examples using XAML and C#*. s.l.:PACKT.

Anon., 2013. W3C. [Online]

Available at: <http://www.w3.org/AudioVideo/>

Baigent, D., Adamiak, M. & Mackiewicz, R., 2004. *IEC 61850: Communication networks and systems in substations: An Overview for Users*. s.l.:SIPSEP.

Boyer, S. A., 2010. *SCADA: Supervisory Control and Data Acquisition 4th Edition*. s.l.:Iliad Development Inc..

EFACEC, 2011. *Transmission Substation - Automation : Ermesind 220/60 Kv Substation [Brochura]*. s.l.:Automation Business Unit.

EFACEC, 2012. *CLP 500SAS*. [Online]

Available at:

http://www.efacec.pt/PresentationLayer/ResourcesUser/Cat%C3%A1logos%202013/Automa%C3%A7%C3%A3o/CS74P1302A1_CLP_500SAS_PT.pdf

[Accessed 25 10 2013].

EFACEC, 2012. *HMI 500 - Multi-touch*. [Online]

Available at: http://www.youtube.com/watch?v=J7Q_7LFhd5w

Eisenberg, J. D., 2002. *SVG Essentials*. s.l.:O' Reilly.

Geneva & Schweiz, 2003. *IEC 61131-3 Programmable controllers – Part 3 - Programming Languages*. s.l.:International Electrotechnical Commission.

Geneva & Schweiz, 2003. *IEC 61499: Function blocks for industrial-process measurement and control systems*. s.l.:International Electrotechnical Commission.

Geneva & Schweiz, 2013. *IEC 60870-5: Telecontrol equipment and systems – Part 5: Transmission protocols*. s.l.:International Electrotechnical Commission.

Jacobs, S., 2006. *Beginning XML with DOM and Ajax, From novice to Professional*. s.l.:Apress.

MacDonald, M., 2010. *Pro WPF in C# 2010*. s.l.:Apress.

Marketing, E., 2013. *HMI 500 - Multi-touch*. [Online]

Available at:

http://www.youtube.com/watch?v=J7Q_7LFhd5w&feature=share&list=PL5F3E984115DF32BC

Nathan, A., 2007. *Windows Presentation Foundation Unleashed*. s.l.:SAMS.

Paulo, R. et al., 2009. *Advanced Engineering Tools for Next Generation Substation Automation Systems: The Added Value of IEC 61850 and the InPACT Project*. In *20th International Conference and Exhibition on Electricity Distribution (CIRED 2009)*, volume PEP0550Z of IET Conference. [Online]
Available at: <http://www3.di.uminho.pt/~jfc/publications/PauloCLBC09.pdf>

Ullman, C. & Dykes, L., 2007. *Beginning Ajax*. s.l.:Wrox, Wiley Publishing, Inc.

W3C, 2003. *Scalable Vector Graphics (SVG) 1.1 Specification*. s.l.:W3C.

W3C, 2008. *Synchronized Multimedia Integration Language (SMIL 3.0)*. [Online]
Available at: <http://www.w3.org/TR/smil/>

Wilton, P. & McPeak, J., 2007. *Beginning JavaScript*. s.l.:WROX.