Ricardo Jorge Nogueira Fernandes

Interfaces Web para Aplicações SCADA



Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
2003

Ricardo Jorge Nogueira Fernandes

Interfaces Web para Aplicações SCADA



Tese submetida à Faculdade de Ciências da Universidade do Porto para obtenção do grau de Mestre em Informática

Dissertação realizada sob a supervisão do
Professor Doutor João Correia Lopes,
Professor Auxiliar da
Faculdade de Engenharia da Universidade do Porto

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
2003

Porto, 11 de Agosto de 2004

Nelma Moreira

Aos meus pais.

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao meu orientador, o professor João Correia Lopes, pela sua disponibilidade constante, pela seu empenho na realização deste trabalho, pelas preciosas correcções ao texto final e, sobretudo, por ter sabido manter-me sempre com os "pés assentes na terra" e concentrado no objectivo final.

À EFACEC Sistemas de Electrónica S.A., nas pessoas do Eng. António Carrapatoso e do Eng. Alberto Rodrigues, por terem avalizado a realização deste trabalho e por terem a convicção de que é possível juntar os meios académico e empresarial com resultados interessantes para ambos.

Ao Dave, por ter marcado profundamente a minha forma de trabalhar e pensar *software*.

Ao Pedro, ao Rui e ao Viegas e às preciosas trocas de opinião sobre as questões que abordei neste trabalho. Sem eles este seria, certamente, mais pobre.

Ao Miguel, pelo seu entusiasmo contagiante e por ter sido, durante os últimos dois anos, o meu "companheiro de carteira" e parceiro de discussões, tornando mais agradável a realização deste mestrado.

Aos amigos e colegas da EFACEC Sistemas de Electrónica S.A., principalmente ao Hugo e ao Carlos, pelo interesse que sempre demostraram, pelo apoio incondicional e, fundamentalmente, por contribuirem para criar, todos os dias, um excelente ambiente de trabalho.

À minha família (agora maior), pelo apoio, pelo carinho e por ter suportado a minha "ausência" durante os anos do mestrado.

E finalmente à Sofia, pelo amor, pelo carinho, pela compreensão, pela paciência e, sobretudo, por ter aceitado partilhar os nossos primeiros meses de casamento com esta tese. Por tudo isto, este trabalho também é teu!

Abstract

SCADA systems are widely used in industry for supervisory, control and data acquisition of industrial processes. These systems allow operators to act on the underlying processes by means of very rich and interactive graphical user interfaces accessible at a central facility.

The trends in industry has been to integrate these kind of systems with other corporate IT functions, focusing both on reducing maintenance costs and on achieving powerful systems which may also be accessed for people outside of the control room.

Taking SCADA to the Web is a way to provide more people with wider access to SCADA information, by means of a common and ubiquitous element: the Web Browser. However, the complexity of SCADA interfaces may present some challenges when trying to develop SCADA Web based applications, due to the browser's natural disability in providing a rich and interactive user experience.

In this work, we propose a strategy for providing SCADA Web based interfaces through a set of Java custom graphical components, which can be integrated into applets that run inside a browser.

The work consisted in the definition of a collaborating components architecture, based on the publish/subscribe paradigm for the exchange of information. The arquitecture was implemented as a component framework. In addition, a SCADA specific tools library was developed in order to contribute for the rapid implementation of graphical custom components. The core element of the library is a custom component used for displaying synoptic diagrams for the SCADA system.

An evaluation of this work was performed, which concluded that the strategy, as initially proposed, fulfills its purposes. It also proved the usefulness of the tools that were developed.

Resumo

Sistemas SCADA são sistemas amplamente divulgados na indústria para a supervisão, controlo e aquisição de informação de processos industriais. Estes sistemas garantem a operação dos processos pela disponibilização de interfaces gráficos ricos e muito interactivos, acessíveis apenas aos operadores residentes num centro de comando.

A tendência crescente de integração destes sistemas com outro tipo de sistemas de informação empresarial, tem visado a obtenção de sistemas mais interoperantes e menos dependentes de equipas de manutenção separadas, permitindo a redução gradual de custos, e a obtenção de ambientes de operação mais completos que tendem a ultrapassar os limites do centro de comando.

O alargamento dos sistemas SCADA à *Web* aparece, assim, como um meio interessante de disponibilização de funcionalidades a um leque mais abrangente de intervenientes, por via de um meio comum e presente em qualquer lugar: o navegador. Contudo, o elevado grau de dinamismo e interactividade próprias das aplicações SCADA são características pouco comuns ao tipo de interfaces *Web* normalmente utilizados, sendo por isso imperativo o estudo de mecanismos que possibilitem uma tal implementação.

Neste trabalho apresenta-se uma abordagem para a disponibilização de interfaces SCADA na *Web*, baseada na utilização de componentes gráficos Java especializados, passíveis de serem integrados em *applets* para execução local num navegador.

A execução do trabalho consistiu na definição de uma arquitectura de componentes colaborantes baseada no princípio de publicação e subscrição de tópicos para a troca de informação, e na disponibilização de uma biblioteca de funcionalidades SCADA, com o intuito de permitir o rápido desenvolvimento daqueles componentes gráficos especializados. O elemento fundamental desta biblioteca é um componente especializado para a visualização de diagramas sinópticos. A arquitectura de componentes materializou-se na implementação de um *framework* de componentes.

Os testes realizados permitiram validar o sucesso da abordagem proposta, bem como a utilidade das ferramentas desenvolvidas.

Conteúdo

Abstract vi

Resumo vii

Índice de Tabelas xv

Índice de Figuras xix

1 Introdução 1

1.1 Sistemas SCADA 1

1.1.1 Sistemas de Tempo-Real de Alta Disponibilidade 2

1.1.2 Tendência: Integração com Outros Sistemas 2

1.1.3 Área de Interesse para Este Trabalho 3

1.2 O Impacto da Web 3

1.3 Motivação e Objectivo 3

1.4 Estrutura da Dissertação 4

2 Sistemas SCADA 6

2.1 Conceito 6

2.1.1 Supervisão 7

2.1.2 Controlo 7

2.1.3 Aquisição de Dados 7

2.2 Arquitectura 8

2.2.1 Estações Remotas 8

2.2.2 Centro de Comando 9
2.2.3 Comunicações 9
2.3 Sistema de Tempo-Real de Alta Disponibilidade 10
2.4 Software de Gestão Central 11
2.4.1 Interface Física 12
2.4.2 Módulos Principais 12
2.4.3 Módulos de Utilizador 15
2.5 SCADA em Sistemas Eléctricos de Energia 20
2.6 SCADA na EFACEC Sistemas de Electrónica, S.A. 22
2.6.1 Controlo de Subestações 22
2.6.2 Centro de Comando 23
2.7 Integração SCADA / SI 26
2.8 SCADA na Web 27
2.9 Sumário 27

3 Aplicações na World Wide Web **29**

3.1 Um Pouco de História 29
3.2 Sistema Universal 30
3.2.1 Navegador — O cliente universal 31
3.2.2 Servidor Web — O servidor universal 31
3.2.3 HTML — A linguagem universal 32
3.2.4 HTTP — O protocolo universal 33
3.2.5 URI — O identificador universal 33
3.3 Tecnologias para Aplicações na Web 34
3.3.1 Common Gateway Interface 34
3.3.2 FastCGI 35
3.3.3 Módulos de Scripting 35
3.3.4 API de Extensões ao Servidor 36
3.3.5 Java Servlets 37

3.3.6 Scripting no Servidor 38
3.3.7 eXtensible Markup Language (XML) 39
3.3.8 Serviços Web 42
3.3.9 Plug-ins 44
3.3.10 Código no Cliente 45
3.3.11 Plataformas Empresariais 47
3.4 Modelo de Programação Distribuído 49
3.4.1 Modelo Cliente-Servidor 50
3.4.2 Modelo de 3-Camadas 51
3.4.3 Modelo de N-Camadas 52
3.5 Aplicações SCADA na Web 53
3.5.1 Rockwell Automation 53
3.5.2 Axeda Systems Inc. 54
3.5.3 Serk Controls 54
3.6 Sumário 55

4 Análise do Problema e Definição da Tese **56**
4.1 Contextualização do Trabalho 56
4.1.1 Revisão dos Requisitos 56
4.1.2 Caracterização do Ambiente 58
4.2 Abordagens Analisadas 59
4.2.1 Clientes Simples 59
4.2.2 Clientes Complexos 62
4.2.3 Escolha da Solução e Definição da Tese 64
4.3 Arquitectura de Software 65
4.3.1 Arquitectura de Componentes 65
4.3.2 Nota de Implementação 67
4.4 Biblioteca de Funcionalidades SCADA 68
4.4.1 Funcionalidades Genéricas 68

4.4.2 Funcionalidades Gráficas 68
4.5 Sumário 69

5 Framework de Componentes **71**
5.1 Arquitectura ou Framework? 71
5.1.1 Framework 72
5.1.2 Arquitectura de Software 72
5.1.3 Distinção 72
5.2 Estrutura dos Pacotes de Software 73
5.3 Framework para Aplicações 73
5.3.1 Contexto 73
5.3.2 Descrição do Problema 74
5.3.3 Descrição Funcional 74
5.4 Modelo de Componentes 75
5.4.1 Contexto 75
5.4.2 Descrição do Problema 76
5.4.3 Eventos, Pedidos de Dados e XML 76
5.4.4 Descrição Funcional 79
5.5 Sumário 82

6 Biblioteca de Funcionalidades SCADA **84**
6.1 Descrição Geral dos Pacotes 84
6.2 Modelo de Atributos 86
6.2.1 Contexto 86
6.2.2 Descrição do Problema 86
6.2.3 Descrição Funcional 87
6.2.4 Voltando aos Eventos 88
6.3 Modelo de Expressões 88
6.3.1 Contexto 88

6.3.2 Descrição do Problema 89
6.3.3 Descrição Funcional 89
6.3.4 Definição de Expressões em XML 92
6.4 Modelo de Regras 97
6.4.1 Contexto 97
6.4.2 Descrição do Problema 97
6.4.3 Descrição Funcional 98
6.4.4 Definição de Regras em XML 99
6.5 Modelo de Componentes Gráficos 100
6.5.1 Contexto 100
6.5.2 Descrição do Problema 102
6.5.3 Descrição Funcional 103
6.6 Gestão Espacial 106
6.6.1 Contexto 106
6.6.2 Descrição do Problema 108
6.6.3 Descrição Funcional 108
6.7 Implementações Standard 109
6.7.1 Contexto 110
6.7.2 Descrição do Problema 110
6.7.3 Descrição Funcional 111
6.8 Modelo de Simbologia 113
6.8.1 Contexto 114
6.8.2 Descrição do Problema 114
6.8.3 Descrição Funcional 115
6.8.4 Gramáticas XML para Motor de Simbologia 118
6.9 Sumário 121

7 Protótipo — ScadaEngine **122**

7.1 Contexto 122

7.2 Descrição do Problema 123
7.3 Descrição Funcional 124
7.4 Configurações da Aplicação 126
7.4.1 Metamodelo de Eventos 126
7.4.2 Configuração de Diagramas 127
7.4.3 Apresentação em HTML 128
7.4.4 Utilização noutros Ambientes 129
7.5 Configurações Gerais 130
7.6 Sumário 131

8 Testes e Análise de Resultados **132**
8.1 Métricas de Avaliação 132
8.2 Ambiente de Testes 133
8.3 Testes à Biblioteca de Funcionalidades SCADA 134
8.3.1 Modelo de Expressões 134
8.3.2 Modelo de Regras 138
8.3.3 Processamento Gráfico 143
8.4 Testes ao Protótipo 145
8.4.1 Descrição dos Testes 145
8.4.2 Análise dos Resultados 148
8.5 Sumário 149

9 Conclusões **151**
9.1 Objectivos Iniciais 151
9.2 Escolha da Solução e Principais Dificuldades 152
9.3 Cumprimento dos Objectivos? 153
9.4 Oportunidades de Melhoria 154
9.5 Sumário 157

A Acrónimos **158**

B Esquema XML para o Motor de Expressões **160**

B.1 Esquema expressions.xsd 160

C Esquema XML para o Motor de Regras **167**

C.1 Esquema rules.xsd 167

D Esquemas XML para o Motor de Simbologia **169**

D.1 Esquema symbology.xsd 169

D.2 Esquema multilayer.xsd 172

E Configurações do Protótipo – ScadaEngine **173**

E.1 Definição de Simbologia — symbology.xml 173

E.2 Definição de Estilos — styles.xml 174

E.3 Definição de um Diagrama da Rede de Metropolitano — metrodiagram.xml 177

E.4 Definição do Diagrama Unifilar de uma Subestação — substationdiagram.xml 179

Referências **181**

Lista de Tabelas

8.1 Parâmetros de ensaios ao modelo de expressões 136
8.2 Resultado dos ensaios ao modelo de expressões 136
8.3 Parâmetros de ensaios ao modelo de regras — Cenário 1 139
8.4 Parâmetros de ensaios ao modelo de regras — Cenário 2 140
8.5 Resultado dos ensaios ao modelo de regras — Cenário 1 140
8.6 Resultado dos ensaios ao modelo de regras — Cenário 2 141
8.7 Parâmetros dos ensaios ao processamento gráfico 144

9.1 Verificação dos Requisitos 155

Lista de Figuras

2.1 Arquitectura física de um sistema SCADA 8

2.2 Diagrama dos blocos funcionais de um possível Software de Gestão Central SCADA 13

2.3 Exemplo de aplicações SCADA gráficas 16

2.4 Estrutura de um SCADA/DMS 21

2.5 BUS — Arquitectura de publicação/subscrição para componentes colaborantes 25

3.1 Elementos fundamentais da Web 30

3.2 Ciclo de processamento de um CGI 35

3.3 Ciclo de processamento de um FastCGI 36

3.4 APIs de extensões ao servidor 37

3.5 Java Servlets 37

3.6 Processador XSLT 41

3.7 Acesso a um serviço Web através da internet 42

3.8 Serviço Web para a correcção ortográfica de documentos 44

3.9 Pilha de tecnologias do J2EE 48

3.10 Plataforma .Net 49

3.11 Modelo de programação cliente-servidor 50

3.12 Modelo de programação de 3-camadas 51

3.13 Modelo de programação de n-camadas 52

4.1 Características da arquitectura de software 66

4.2 Biblioteca de funcionalidades SCADA 68

5.1 Estrutura de pacotes do framework de componentes 73
5.2 Diagrama de classes do pacote frk 75
5.3 Estrutura de um evento (event.dtd) 76
5.4 Estrutura de um pedido de dados (data_request.dtd) 77
5.5 Estrutura de eventos e pedidos de dados para transmissão (events.dtd) 77
5.6 Exemplo de um documento XML com eventos e pedidos de dados (events.xml) 78
5.7 Estrutura do metamodelo de eventos (metamodel.dtd) 78
5.8 Exemplo de um documento XML contendo um metamodelo de eventos (metamodel.xml) 79
5.9 Diagrama de classes afectas ao processamento de eventos 80
5.10 Diagrama de classes do pacote cmp 81

6.1 Estrutura geral de pacotes da biblioteca de funcionalidades SCADA . . 85
6.2 Diagrama de classes do pacote efa.types 87
6.3 Diagrama de classes do pacote efa 87
6.4 Diagrama de classes do pacote efa.exp — expressões 90
6.5 Diagrama de classes do pacote efa.exp — operandos 91
6.6 Diagrama de classes do pacote efa.exp — operadores 92
6.7 Esquema XML de expressões — expressão condicional 93
6.8 Esquema XML de expressões — expressão booleana 94
6.9 Esquema XML de expressões — expressão relacional 94
6.10 Esquema XML de expressões — expressão aritmética 95
6.11 Exemplo de uma expressão em XML 96
6.12 Diagrama de classes do pacote efa.rules — regras 98
6.13 Representação gráfica do esquema XML para o modelo de regras 99
6.14 Definição de uma regra em XML 101
6.15 Diagrama de classes do pacote efa.base — Componente Opaco 103
6.16 Diagrama de classes do pacote efa.base — mapeamento de coordenadas 105
6.17 Diagrama de classes do pacote efa.base — componente visualizador . . 106

6.18 Diagrama de classes do pacote toolkit.tools.spatial — quadtrees 109
6.19 Diagrama de classes do pacote toolkit.standard — ferramentas 111
6.20 Diagrama de classes do pacote toolkit.standard — figuras 112
6.21 Diagrama de classes do pacote toolkit.standard — camadas 113
6.22 Diagrama de classes do pacote toolkit.symbol — estrutura MVC 115
6.23 Diagrama de classes do pacote toolkit.symbol — símbolos 117
6.24 Diagrama de classes do pacote toolkit.symbol — gestores 118
6.25 Diagrama de classes do pacote toolkit.symbol — componente de simbologia 119
6.26 Representação gráfica do esquema XML de simbologia 120
6.27 Representação gráfica do esquema XML de diagramas 120

7.1 Contexto de utilização do ScadaEngine 122
7.2 Estrutura de pacotes do ScadaEngine 123
7.3 Diagrama de classes do ScadaEngine 124
7.4 Diagrama de classes do ScadaEngine — modelos 126
7.5 Metamodelo de eventos para o ScadaEngine 127
7.6 Informação para carregamento de diagrama 128
7.7 Página HTML para o ScadaEngine 128
7.8 ScadaEngine — applet 129
7.9 ScadaEngine — aplicação stand-alone 130

8.1 Tempo de avaliação de expressões — extremo positivo 138
8.2 Tempo de avaliação de expressões — extremo negativo 139
8.3 Tempo de avaliação de regras — Cenário 1 141
8.4 Tempo de avaliação de regras — Cenário 2 142
8.5 Resultados dos ensaios ao processamento gráfico — Teste 1 146
8.6 Resultados dos ensaios ao processamento gráfico — Teste 2 146
8.7 Resultados dos ensaios ao processamento gráfico — Teste 3 147
8.8 Resultados dos ensaios ao processamento gráfico — Teste 4 147

8.9 Resultados dos ensaios ao processamento gráfico — Teste 5 148

8.10 Resultados dos ensaios ao protótipo 149

B.1 Representação gráfica do esquema XML de expressões 166

Capítulo 1

Introdução

Este capítulo apresenta uma visão global do trabalho pela abordagem dos temas principais nele discutidos, e pela descrição da motivação e objectivos que levaram à sua concretização.

No final descreve-se de forma sumária a estrutura da presente dissertação.

1.1 Sistemas SCADA

As tarefas relativas à gestão operacional de processos industriais constitui uma problemática comum a diversas áreas de negócio actuais, principalmente àquelas cujos processos se apresentam dispersos por áreas geograficamente distintas, tais como os sistemas de distribuição de energia eléctrica, gás, água, redes de telecomunicações, entre muitos outros. Nestes sistemas, a possibilidade de concentração de informação relativa aos seus processos, em conjunto com a possibilidade de intervenção remota nesses mesmos processos, constituem ferramentas imprescindíveis ao garante da sua operação contínua, e à manutenção de níveis de qualidade elevados. Estas actividades são asseguradas por sistemas altamente especializados, os sistemas SCADA.

Sistemas SCADA são, assim, sistemas de supervisão, controlo e aquisição de informação de processos industriais [DS99]. Estas características fazem dos sistemas SCADA estruturas complexas que integram conceitos e tecnologias diversas: a sua implementação requer a utilização de equipamentos especializados (sensores, actuadores, unidades de aquisição, etc), capacidade de comunicação rápida por utilização de vários protocolos e a existência de um sistema central capaz de receber, gerir e apresentar a informação de forma simples, permitindo a intervenção de operadores sempre que necessário.

Os operadores contactam com o sistema através do seu interface, normalmente constituído por aplicações gráficas, cujo objectivo é o de apresentar a informação numa forma simples e intuitiva. A evolução das capacidades gráficas de computadores e terminais, permitiu a evolução das interfaces SCADA de tal modo que os produtos actualmente existentes primam pela disponibilização de interfaces gráficos evoluídos, que são utilizados para a representação dos processos, por via da incorporação de símbolos e animações, e onde se aplicam metáforas gráficas que facilitam a sua operação: ampliações, arrastamentos, *point-and-click*, *popups*, etc [DS99, Boy93].

1.1.1 Sistemas de Tempo-Real de Alta Disponibilidade

Sendo ferramentas de supervisão, os sistemas SCADA devem garantir que a propagação de alterações ocorridas nos processos sejam efectuadas no menor tempo possível, idealmente zero [Boy93], permitindo a tradução fiel do estado dos processos físicos que controlam ao mesmo tempo que contribuem para o aumento da confiança dos operadores na sua utilização.

Devido à natureza de operação contínua dos processos que controla, um sistema SCADA deve garantir taxas de serviço próximas dos 100%, o que leva à utilização de equipamentos menos sujeitos a falhas (ex. servidores UNIX) e à implementação de mecanismos redundantes de operação e estratégias de recuperação e comutação de funcionamento: servidores duplicados, *watch-dogs*, capacidade de comutações entre servidores, mecanismos de *backup* e recuperação de dados, etc.

Estas características fazem dos SCADA sistemas de tempo-real de alta disponibilidade.

1.1.2 Tendência: Integração com Outros Sistemas

A utilização de sistemas SCADA tem sido confinada à implementação de redes autónomas que funcionam isoladas do resto dos sistemas de informação das empresas que os utilizam. Tal acontece em parte devido à especificidade deste tipo de sistemas, em parte devido à importância que constituem uma vez que se apresentam como o sistema de informação mais próximo das actividades ligadas às áreas de negócio dessas empresas.

Contudo, a necessidade de diminuição de custos inerentes à gestão separada de vários sistemas, aliada à crescente consciencialização dos benefícios provenientes da interligação de sistemas SCADA com outros sistemas de informação, abre perspectivas interessantes para a sua expansão, nomeadamente para a *Web*.

1.1.3 Área de Interesse para Este Trabalho

A área de interesse para o presente trabalho restringe-se ao estudo do *software* de gestão central, nomeadamente às aplicações que promovem o interface com os operadores, e à possibilidade da sua expansão para um ambiente *Web*.

1.2 O Impacto da Web

A *World Wide Web*, um sistema de navegação por hipertexto desenvolvido por Tim Berners-Lee, mudou a forma de lidar com a informação. Embora a *internet* tivesse já permitido a interligação de redes, foi com a *Web* que se definiu um mecanismo universalmente aceite para a disponibilização global da informação nelas contida. Fê-lo pela definição de uma arquitectura simples e baseada em princípios que se revelaram independentes de plataformas e tipo de informação existentes: um servidor universal (o servidor HTTP), um protocolo de comunicação universal (o HTTP), uma linguagem universal (o HTML), um meio de identificação universal de recursos (o URI) e um cliente universal (o navegador) [CH98].

Embora o conceito original da *Web* fosse o da apresentação de documentos HTML estáticos interligados por ligações de hipertexto, desde logo se avançou para o desenvolvimento de técnicas que permitiram a implementação de sistemas cada vez mais complexos capazes da execução de processos especializados que culminavam com a geração dinâmica de informação. Este tipo de capacidades motivou o desenvolvimento de um novo tipo de aplicações, as aplicações *Web*, que vieram beneficiar das suas características de acesso universal.

Como veremos mais adiante, a implementação de aplicações *Web* conduziu ao desenvolvimento de modelos de programação específicos, que permitem reutilizar a lógica de negócio implementada para sistemas existentes e com isso reduzir o desenvolvimento de aplicações *Web* à implementação de interfaces nela baseados.

1.3 Motivação e Objectivo

O alto grau de especificidade dos sistemas SCADA fazem da sua instalação e manutenção tarefas exigentes e complexas, que requerem a existência de pessoal técnico especializado e em constante vigilância.

Se a interligação entre sistemas permite, por um lado, a diminuição de custos oriundos da gestão de várias redes separadamente, introduz, por outro, a possibilidade de acesso aos sistemas até ao alcance das redes que os servem. Este benefício pode, e deve ser aproveitado, mas apresenta dificuldades decorrentes da necessidade de instalação de *software* específico nos vários equipamentos de acesso, de modo a permitir a sua

disponibilização generalizada à empresa.

Disponibilizar aplicações SCADA na *Web* é, antes de mais, utilizar a sua vantagem natural: acesso global. Isto significa a disponibilização de serviços especializados através de um meio comum e presente em qualquer lugar: o navegador.

Esta abordagem permite a concentração de esforços de instalação e configuração de aplicações SCADA apenas aos servidores do sistema, ao mesmo tempo que apresenta possibilidades de acesso ilimitadas, garantidas pelas infra-estruturas actuais de acesso, sejam as redes locais ou a própria *internet*.

Neste contexto, o objectivo do presente trabalho é o estudo e demonstração da possibilidade de apresentação dos interfaces gráficos tipicamente disponibilizados por aplicações SCADA, num ambiente único como é a *Web*.

Este estudo levar-nos-á à análise de algumas soluções e á escolha de opção considerada mais apropriada para a implementação deste tipo de aplicações, e com isso, à definição final da tese que se defende.

Numa forma mais detalhada, o trabalho que ora se apresenta propõe-se a:

- Identificar os tipos de aplicações típicas de sistemas SCADA, analisar as metáforas gráficas mais comuns e caracterizar o tipo de informação que circula entre aplicações;
- Apresentar uma arquitectura apropriada para o desenvolvimento de interfaces gráficos SCADA na *Web*;
- Provar a sua viabilidade pela disponibilização de interfaces *Web* que apresentem as mesmas características das aplicações gráficas SCADA convencionais.

1.4 Estrutura da Dissertação

A dissertação estrutura-se em 9 capítulos que apresentam o trabalho desenvolvido e os conceitos fundamentais à sua compreensão:

- Capítulo 2 – **Sistemas Scada**
  Neste capítulo descreve-se o conceito e a arquitectura típica de sistemas SCADA, com especial atenção ao *software* de gestão central e às características gráficas das aplicações que promovem o interface com o utilizador. Será com base nestas características que se definirão os requisitos gráficos que se pretendem transportar para a *Web*.
- Capítulo 3 – **Aplicações na *World Wide Web***
  Aqui é apresentada uma visão geral do que é a *World Wide Web* e das principais tecnologias que permitem o desenvolvimento de aplicações nela baseados. Este

capítulo apresenta ainda o resultado da pesquisa de soluções SCADA para a *Web* disponibilizadas por alguns fabricantes.

- Capitulo 4 – **Análise do Problema e Definição da Tese**
  Neste capítulo contextualiza-se o problema em análise pela definição do ambiente em estudo e pela apresentação dos requisitos que serviram de base à escolha da solução implementada. Esta escolha resultou da análise de quatro abordagens ligadas à disponibilização de aplicações gráficas no navegador e conduziu à definição final da tese, que é aqui enunciada. A apresentação da tese permite ainda enquadrar o trabalho a desenvolver pela definição das tarefas a realizar.

- Capítulo 5 – **Framework de Componentes**
  Este capítulo descreve todo o projecto de *software* levado a cabo para a implementação do *framework* de componentes colaborantes em Java, que constitui a materialização dos princípios da arquitectura definida no capítulo anterior. O projecto socorre-se da utilização de padrões de desenho de *software* e utiliza-se a notação UML como forma de apresentação dos diagramas de classes nele discutidos.

- Capítulo 6 – **Biblioteca de Funcionalidades SCADA**
  Neste capítulo apresenta-se o projecto de *software* referente à biblioteca de funcionalidades SCADA, onde se descrevem os vários pacotes que a constituem, culminando com a descrição de um componente gráfico especializado para a visualização de diagramas sinópticos. Também aqui se utiliza a notação UML como meio de apresentação dos diagramas de classes discutidos e indicam-se os padrões de desenho de *software* utilizados.

- Capítulo 7 – **Protótipo – ScadaEngine**
  A apresentação do ScadaEngine, um protótipo que valida o trabalho desenvolvido, é efectuada neste capítulo. Aqui se descreve todo o projecto de *software* inerente à sua implementação e se discutem algumas questões relativas à configuração do ambiente de utilização. Mais uma vez utiliza-se a notação UML para a apresentação do projecto de *software*.

- Capítulo 8 – **Testes a Análise dos Resultados**
  Neste capítulo apresentam-se os ensaios realizados às funcionalidades implementadas, e a sua análise em função de métricas vulgarmente utilizadas na indústria do *software*, para a avaliação do grau de interactividade das aplicações.

- Capítulo 9 – **Conclusão**
  Neste capítulo avalia-se o resultado do trabalho e apresentam-se propostas para a sua evolução.

Capítulo 2

Sistemas SCADA

Este capítulo introduz o conceito de sistemas SCADA apresentando uma descrição geral da sua arquitectura física. A preocupação principal é a análise do *software* de gestão central do sistema, com especial ênfase para as aplicações que promovem o interface com o operador.

Introduziremos depois as especificidades deste sistemas relativas à sua utilização em sistemas eléctricos de energia (DMS) e, com essa deixa, apresentaremos a EFACEC Sistemas de Electrónica, S.A. (EFACEC SE), mais concretamente, a unidade de negócio para Automação e Sistemas de Energia (EFACEC SE/AS) e as suas actividades no âmbito dos sistemas SCADA e SCADA/DMS: unidades de aquisição e controlo de subestações e *software* para centros de comando.

No final, efectua-se uma pequena análise da tendência de integração dos sistemas SCADA com outros sistemas de informação empresarial, com especial atenção para a sua expansão ao ambiente *Web*.

2.1 Conceito

SCADA é o acrónimo de *Supervisory Control And Data Acquisition* (Supervisão Controlo e Aquisição de Dados) e trata-se de um sistema que permite a monitorização e controlo de processos geograficamente dispersos [Boy93]. A utilização de sistemas SCADA é vulgar para a gestão de processos industriais, sistemas de distribuição de gás, água e energia, redes de telecomunicações, redes de transporte e outros. Um sistema SCADA permite a comunicação entre estações remotas e um centro de comando, fornecendo informação necessária para o controlo do processo.

2.1.1 Supervisão

Num SCADA, os operadores do centro de comando encontram-se em constante vigilância ao estado do sistema que controlam, através da análise da informação por ele recolhida e apresentada nos seus postos de operação. A informação é normalmente apresentada num modo simples e intuitivo, permitindo a sua rápida assimilação. É por isso comum a existência de diagramas animados que representam o sistema físico controlado. A ocorrência de alterações físicas no sistema é imediatamente reproduzida, graficamente, nos terminais de operação. É também comum que a ocorrência de determinadas condições possam levar ao disparo de alarmes que visam chamar a atenção do operador para a existência de uma situação excepcional e que, por isso, deve ser analisada com maior atenção. A ocorrência de um alarme pode traduzir-se numa alteração gráfica do terminal de operação ou, em situações mais graves, pode levar ao accionamento de um sinal sonoro.

Como exemplo, a alteração da temperatura de uma caldeira poderá ser representada graficamente por um termómetro que mimetiza o funcionamento de um termómetro físico. No caso da temperatura exceder um valor pré-determinado, o sistema informará o utilizador através do disparo de um alarme gráfico e/ou sonoro.

2.1.2 Controlo

Os operadores de um sistema SCADA têm a possibilidade de actuar remotamente sobre o processo físico, através de execução de controlos, a partir dos seus postos de operação. Um controlo, consiste no accionamento remoto de um equipamento pelo envio de uma "ordem electrónica" através do sistema, sendo por isso conhecido como telecontrolo.

A execução de telecontrolos apresenta duas vantagens importantes: permite a actuação directa no sistema a partir de um lugar centralizado e permite que a actuação ocorra rapidamente, fazendo com que a correcção de problemas aconteça no menor tempo possível.

2.1.3 Aquisição de Dados

Uma característica importante de um sistema SCADA é a sua capacidade de colectar informação referente ao sistema físico que controla. A informação recolhida pode assim ser enviada para o centro de comando, tanto para visualização imediata através das aplicações de supervisão, como para arquivo em registo histórico para posterior tratamento.

Os dados de um sistema SCADA constituem uma peça de informação valiosa para técnicos e gestores da empresa. A sua análise permite a compreensão mais aprofundada

dos problemas ocorridos no passado e permite efectuar projecções e previsões sobre o modo de agir no futuro.

2.2 Arquitectura

Um SCADA é um sistema complexo que abrange um conjunto diverso de tecnologias. A figura 2.1 ilustra a arquitectura física de um sistema SCADA de onde se destacam três elementos fundamentais:

- Estações Remotas
- Centro de Comando
- Comunicações

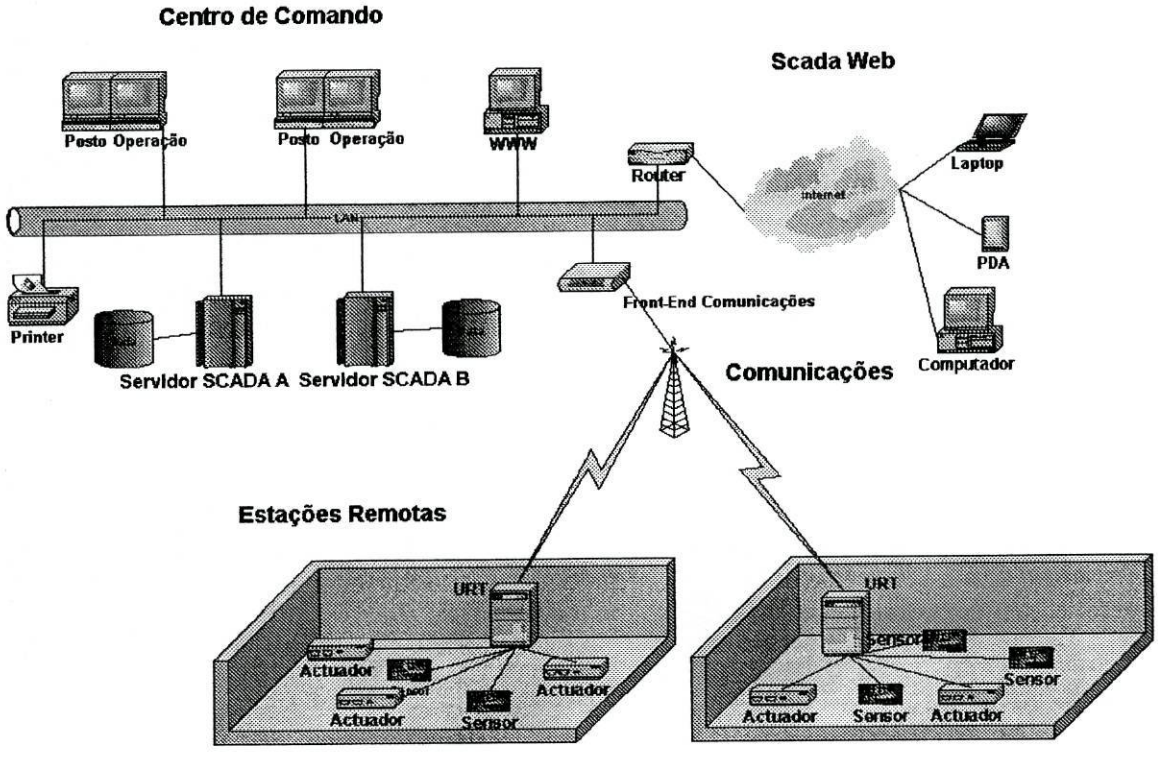


Figura 2.1: Arquitectura física de um sistema SCADA

2.2.1 Estações Remotas

As estações remotas constituem os pontos pelos quais se distribui o processo físico controlado. Num sistema eléctrico de energia, por exemplo, estas estações correspondem a subestações eléctricas dispersas geograficamente.

Em cada estação encontra-se uma Unidade Remota Terminal (URT) responsável pela gestão local dos equipamentos da rede de sensores e actuadores e pela comunicação com o centro de comando. Um sensor é um dispositivo instalado no processo, capaz de traduzir grandezas físicas em impulsos eléctricos (termómetros, barómetros, sensores de corrente, etc). Em oposição, um actuador permite o accionamento remoto de equipamentos por tradução das instruções de comando em acções directas sobre estes (ex. abrir comporta, aumentar temperatura, etc).

Assim, uma URT tem duas funções primordiais: 1) recolher informação proveniente da rede de sensores para transmissão ao centro de comando, e 2) enviar os pedidos de accionamento de equipamentos, provenientes do centro de comando, aos actuadores respectivos.

2.2.2 Centro de Comando

O centro de comando é o lugar que processa toda a informação recebida das estações remotas do sistema e a partir do qual é possível efectuar o seu controlo e supervisão.

Uma vez recebida, a informação é veiculada para o Software de Gestão Central (SGC)[1]. O SGC procede então ao tratamento da informação em tempo-real, disponibilizando-a aos operadores através das aplicações de supervisão, e garantido simultaneamente o seu registo em arquivos históricos. O SGC é também a unidade responsável por enviar às URTs os pedidos de execução de telecontrolos, sendo por isso uma peça capital na arquitectura de um SCADA. É aliás por essa razão, que o SGC é normalmente utilizado numa configuração duplicada, por forma a minimizar situações de indisponibilidade.

A figura 2.1, ilustra este tipo de configuração. O SGC residente no **Servidor SCADA B** pode, em caso de falha, substituir o SGC residente no **Servidor SCADA A**.

É no centro de comando onde os operadores têm acesso ao sistema através dos seus postos de operação que se encontram ligados ao SGC através de uma rede local, normalmente independente de outras redes da empresa [Wie01]. A esta rede interligam-se também outros equipamentos afectos ao funcionamento global do sistema; alarmes, comunicações, impressoras de registo contínuo de alarmes, ou no caso em estudo, o interface do sistema para a *Web*.

2.2.3 Comunicações

A rede de comunicações é também um elemento fundamental na arquitectura de um sistema SCADA, na medida em que assegura a troca de informação constante entre estações remotas e o centro de comando. A ocorrência de uma falha na rede de

[1] O termo Software de Gestão Central é da responsabilidade do autor. A literatura refere-se ao SGC como MTU – *Master Terminal Unit* [Boy93].

comunicações é de tal forma grave, que impossibilita toda e qualquer utilização do sistema.

A gestão das comunicações é um problema complexo que tem que ter em conta três factores importantes:

- Devido à dimensão de alguns processos controlados por sistemas SCADA, é normal existirem estações remotas equipadas com URTs de fabricantes diversos, implementando protocolos de comunicação distintos;
- O meio físico de transmissão da informação pode utilizar tecnologias tão variadas como rádio, fibra óptica, GSM, linhas alugadas, etc;
- É necessário concentrar todos os canais de comunicação num único de interligação com SGC.

Deste modo, é imperativa a existência de um elemento especializado capaz de resolver as questões próprias de comunicação com cada uma das estações remotas, estabelecendo um único canal de transmissão com o SGC. Este elemento está identificado na figura como **Front-End de Comunicações**.

2.3 Sistema de Tempo-Real de Alta Disponibilidade

Do ponto de vista de aquisição de dados, os sistemas SCADA devem ser capazes de reproduzir alterações ocorridas nos processos que controlam no menor intervalo de tempo possível, idealmente zero [Boy93]. Este factor permite não só a tradução fiel do estado dos processos físicos controlados como eleva também o grau de confiança de operadores na utilização do sistema.

Esta necessidade de aquisição quase imediata da informação é uma das preocupações fundamentais de quem desenvolve e de quem utiliza sistemas SCADA, sendo tanto mais importante quanto maior fôr a dimensão dos processos controlados, como é o caso dos sistemas eléctricos de energia. A título de exemplo, apresentam-se dados relativos a um dos sistemas instalado pela EFACEC SE/AS para o controlo e supervisão da rede de distribuição eléctrica da LTE (Electricidade de Lisboa e Vale do Tejo) [pESdES]. Na altura de escrita deste texto, o sistema instalado permitia a aquisição de cerca de 75 mil pontos de medida (sensores analógicos e digitais) e o controlo de cerca de 15 mil pontos de actuação (actuadores). Segundo o caderno de encargos deste projecto, o tempo máximo admissível para a aquisição de informação referente a sensores é de 2 segundos para sensores digitais e de 4 segundos para sensores analógicos. Medições efectuadas no local permitiram ainda verificar a recepção de variações de valores de sensores (na sua maior parte analógicos) a uma taxa de cerca de 200 variações por

segundo. Estes serão os valores tomados como referência durante a apresentação deste trabalho.

Devido à natureza de operação contínua dos processos que controla, os sistemas SCADA devem garantir taxas de serviço próximas dos 100%. De facto, e voltando à informação recolhida a partir do caderno de encargos supracitado, nele se definem três importantes requisitos:

Disponibilidade global do sistema – 99,999%

Tempo máximo de perda de uma funcionalidade crítica – 1 hora por mês

Tempo médio entre falhas [2] – 10 mil horas

A necessidade de cumprimento destes requisitos conduz, normalmente, à utilização de equipamentos menos sujeitos a falhas, à existência de mecanismos redundantes de operação e à implementação de estratégias de recuperação e comutação de funcionamento. Estas características são normalmente conseguidas pela utilização de servidores duplicados, *watch-dogs* para aviso de interrupções, capacidade de comutação entre servidores, mecanismos de *backup* e recuperação de dados, discos RAID, etc.

Estas condicionantes fazem dos sistemas SCADA sistemas de tempo-real de alta disponibilidade.

2.4 Software de Gestão Central

A análise do Software de Gestão Central reveste-se de vital importância para a realização da presente dissertação, na medida em que é o elemento de *software* mais importante num sistema SCADA, sendo através dele que operadores contactam com o processo físico controlado, por meio de um conjunto de aplicações muito bem conhecidas. Nesta secção, será feita uma descrição geral dos componentes típicos de um SGC, seguido de um levantamento detalhado das características gráficas das aplicações que interagem com o utilizador. Pretende-se, no final, obter o conjunto de requisitos que nos permitirão estudar o modo de transferir esse tipo de aplicações para um ambiente *Web*.

Apesar da ampla utilização de sistemas SCADA na indústria, não existe nenhum tipo de norma que descreva a constituição de um SGC em termos formais. Como tal, a descrição que ora se apresenta tem como base a informação recolhida no seio da EFACEC SE/AS em conjunto com a análise de cadernos de encargos apresentados à EFACEC SE/AS no âmbito da sua actividade empresarial na área de sistemas de automação, tanto em termos nacionais como internacionais, ao longo dos seus mais de 15 anos de actividade.

[2] *Mean Time Between Failures* (MTBF)

A figura 2.2 apresenta uma estrutura possível de um SGC SCADA.

A figura encontra-se dividida em três zonas funcionais, que serão alvo de análise detalhada: interface física, módulos principais e módulos de operador.

2.4.1 Interface Física

Do lado esquerdo da figura representam-se elementos físicos que agem directamente no sistema: sensores e actuadores. Do ponto de vista do SGC, toda a informação referente a sensores será designada internamente pelo termo *entidade*. Uma entidade pode ser do tipo digital ou analógica. Uma entidade digital define um estado de um conjunto de estados possíveis enquanto que uma entidade analógica representa um valor numérico. No sentido contrário, as acções permitidas sobre actuadores designar-se-ão por *controlos*. Os controlos podem ser do tipo comando ou *setpoint*. Um comando refere-se à actuação de um equipamento que tem um conjunto finito de estados (ex. porta aberta, porta fechada). Um *setpoint* refere-se à execução de uma acção que é dependente de um valor analógico. (ex. subir a temperatura 20 graus).

2.4.2 Módulos Principais

Após a caracterização do tipo de informação que flui dentro de um SGC (entidades e controlos), importa agora identificar os vários módulos que o constituem.

2.4.2.1 Módulo de Telemetria

O módulo de telemetria tem como função receber a informação das entidades provenientes da interface física e veiculá-la para os módulos de arquivos, de alarmes e de áreas de responsabilidade. O seu funcionamento é regido pelas definições registadas e disponibilizadas pelo módulo de configuração, nomeadamente quais as entidades que deverão ser propagadas pela telemetria ao resto do sistema. Uma funcionalidade normal num SCADA consiste em definir um valor fixo para uma dada entidade, de modo a que a sua telemetria não seja propagada ao resto do sistema[3]. Esta funcionalidade permite, por exemplo, ignorar valores errados enviados por URTs defeituosas.

2.4.2.2 Módulo de Alarmes

O módulo de alarmes recebe a informação das entidades e verifica se o seu valor deve dar origem ao disparo de um alarme. As condições que definem o disparo de um alarme fazem parte da configuração do sistema e são obtidas a partir do módulo de configuração. No caso de haver alarmes, estes são enviados para o módulo de áreas

[3]Esta acção designa-se por imposição de valor.

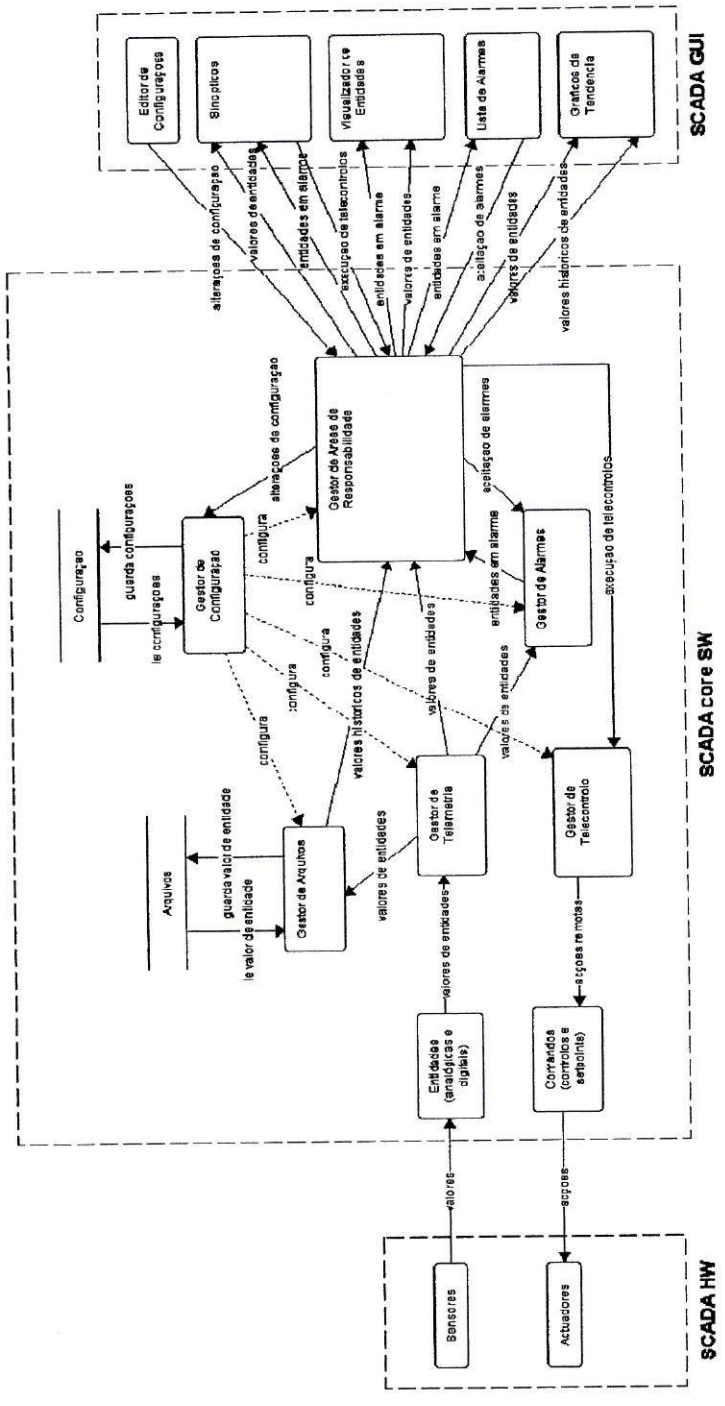


Figura 2.2: Diagrama dos blocos funcionais de um possível Software de Gestão Central SCADA

de responsabilidade para tratamento por parte dos operadores. Todos os alarmes de um sistema SCADA devem ser aceites pelos operadores, por forma a garantir que estes tomaram consciência da sua ocorrência, mesmo que a causa do alarme tenha já desaparecido [Boy93]. Uma vez aceites, os alarmes são removidos do módulo de alarmes.

2.4.2.3 Módulo de Arquivos

A função do módulo de arquivos é a de garantir que toda a informação relevante recebida pelo sistema é registada num repositório persistente para posterior tratamento [Boy93]. Este repositório poderá ser um sistema de ficheiros ou uma base da dados. Cabe ao módulo configurador informar que informação deve ser considerada relevante para arquivo. É também o módulo de arquivos que fornece informação acerca de entidades já arquivadas, enviando-a para o módulo de áreas de responsabilidade.

2.4.2.4 Módulo de Telecontrolos

O módulo de telecontrolos é a ligação do SGC com o sistema físico constituído pela rede de actuadores. A execução de um telecontrolo provoca o envio de uma mensagem que resulta no accionamento de um equipamento. Para alguns controlos é possível obter informação acerca do resultado da operação e enviá-la de volta ao módulo de áreas de responsabilidade (origem do pedido)[4].

2.4.2.5 Módulo de Configurações

Este módulo é responsável pelo registo de toda a informação referente à configuração geral do sistema num repositório persistente, normalmente numa base de dados. É com base nesta configuração que os restantes módulos definirão o seu modo de actuar. Assim, o módulo de configurações deverá disponibilizar mecanismos de acesso à informação e mecanismos de notificação, permitindo a sincronização dos restantes módulos com eventuais alterações à sua configuração.

O autor inclui neste módulo todas as funcionalidades que implicam o registo de informação indispensável à utilização do sistema, como é o caso do registo das entidades e acções existentes. É também aqui que se enquadram as actividades de construção dos diagramas sinópticos que serão discutidos mais adiante.

[4] A existência de uma confirmação depende do tipo de equipamento actuado (actuador), visto não ser possível obter essa informação para todos os equipamentos.

2.4.2.6 Módulo de Áreas de Responsabilidade

O módulo de áreas de responsabilidade rege todo o acesso de utilizadores ao sistema pela definição de um modelo de permissões. O objectivo deste modelo é garantir que o acesso a recursos é validado pela existência de uma permissão para o efeito [Boy93]. Um recurso pode ser algo tão diverso como a informação física (entidades e controlos), uma impressora ou mesmo uma aplicação do próprio sistema. O modelo deve também definir os tipos de acesso possíveis, bem como que utilizadores poderão utilizar que modos de acesso sobre que tipo de recursos.

A existência desta funcionalidade permite garantir que apenas utilizadores qualificados para a execução de uma determinada tarefa tenham permissão para o fazer. Daí ser vulgar a definição de grupos ou perfis de utilizadores, que são posteriormente atribuídos aos operadores do sistema.

Um exemplo da utilização deste mecanismo é a criação de grupos de utilizadores para a gestão de zonas do processo geograficamente distintas. Assim, os utilizadores de cada grupo poderão apenas actuar sobre as zonas do processo referentes à sua área geográfica (ex. área norte, área sul, etc).

Outro exemplo de utilização consiste na definição de permissões por categoria hierárquica. Neste contexto, poder-se-á criar um grupo específico para administradores e outro para operadores. Aos operadores poderão ser dadas permissões de utilização do sistema, mas sem a possibilidade de configuração, visto esta ser uma tarefa tendencialmente de administração e, portanto, da responsabilidade dos administradores.

2.4.3 Módulos de Utilizador

A figura 2.2, apresenta do lado direito o conjunto de aplicações dedicadas à interacção directa com os utilizadores do sistema e que, por isso, designamos por módulos de utilizador.

A sua análise conduzirá à definição das características gráficas comuns aos sistemas SCADA, constituindo-se assim como requisitos para a sua implementação na *Web*. A figura 2.3 ilustra o aspecto deste tipo de aplicações.

Em termos globais, este módulos constituem visões diferentes da mesma informação, variando apenas nas funcionalidades específicas que disponibilizam.

2.4.3.1 Diagramas Sinópticos

Diagramas sinópticos são aplicações que representam graficamente o processo físico controlado, por meio da utilização de símbolos, imagens e animações. Estas características permitem aos operadores encararem o diagrama da mesma forma que o fariam com o processo real, simplificando a tarefa de gestão do sistema.

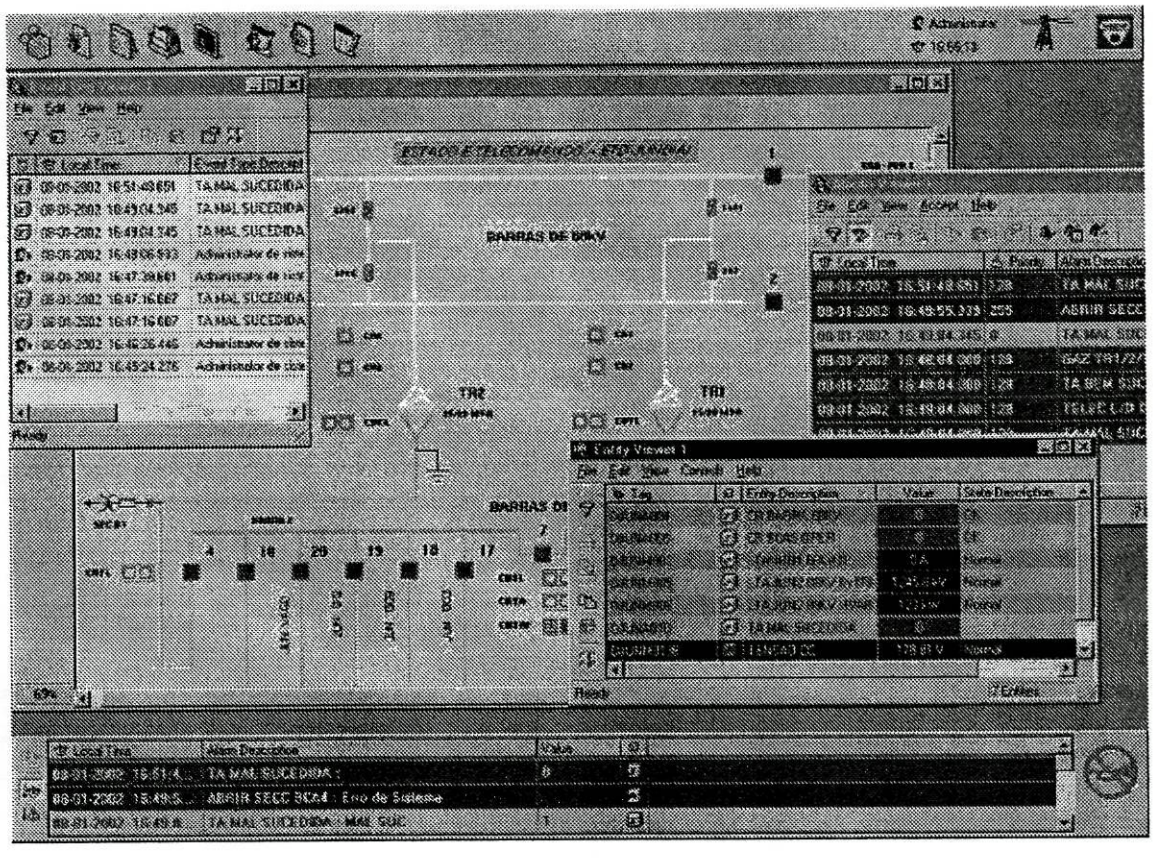


Figura 2.3: Exemplo de aplicações SCADA gráficas

Num sinóptico, a alteração do valor de uma entidade traduz-se numa actualização gráfica do seu diagrama, reproduzindo a ocorrência física que a originou. Assim, a alteração do valor da medida de um caldeira (entidade analógica) poderá ser representada pela subida do nível de mercúrio de um termómetro gráfico, mimetizando o comportamento de um termómetro físico presente no processo. A representação gráfica desta situação tem o benefício de fornecer ao operador dois tipos de informação: a ocorrência de uma variação de temperatura (devido ao termómetro) e o seu valor. Se ao diagrama juntarmos uma escala que indique os valores máximo e mínimo admissíveis, então o utilizador poderá rapidamente aperceber-se se o novo valor de temperatura está, ou não, dentro dos limites.

A utilização deste tipo de aplicações possibilita a recolha de informações que, estando indirectamente ligadas com os dados recebidos, não fazem contudo parte da alteração ocorrida no processo, mas que proporcionam uma percepção mais alargado do estado real do sistema.

Contudo, a visualização do valor de entidades não está sempre ligada a estruturas gráficas complexas, podendo, por exemplo, limitar-se à sua apresentação textual no diagrama. Mas mesmo nestes casos, é comum a existência de regras que alterem a côr do texto ou do fundo em função do valor representado. Uma entidade em alarme, por exemplo, pode ser representada pela apresentação do seu valor que pisca alternadamente entre duas cores.

Os diagramas sinópticos permitem ainda a visualização das características das entidades ou mesmo dos equipamentos presentes no sistema físico, dotando os operadores de informação extremamente útil para as várias tarefas de gestão (ex. características eléctricas de equipamentos).

Outra das suas características é a possibilidade de execução de telecontrolos também com recurso ao mesmo tipo de metáforas gráficas. Voltando ao exemplo da temperatura, o diagrama poderá permitir aumentar a temperatura de um forno por actuação na válvula de alimentação de combustível (através da execução de um *setpoint*). Graficamente, tal pode ser modelado através de um botão giratório, um botão de progressão, um painel para a entrada do valor, etc.

Para sistemas mais complexos (com muitos pontos de medida e actuação), como os sistemas eléctricos de energia, é comum existirem facilidades de navegação nos diagramas através da representação de camadas que vão aparecendo, ou desaparecendo, conforme a variação do nível de ampliação (*zoom*). Paralelamente, e devido à dimensão de alguns diagramas, é normal haver a possibilidade de arrastamento do diagrama de modo a posicioná-lo melhor no ecrã (*pan*).

Estas facilidades permitem a selecção de áreas de visualização específicas, tanto em termos posicionais, como em termos de pormenor, facilitando a gestão global do sistema.

Ainda nos sistemas eléctricos de energia, é comum basear a representação gráfica

dos equipamentos, nos diagramas sinópticos, num conjunto substancialmente menor de símbolos que vão sendo repetidos nos vários diagramas e que são, muitas vezes, constituídos por figuras simples, à semelhança, aliás, do que acontece com os diagramas unifilares de rede disponíveis noutros formatos (papel por exemplo).

Podemos então resumir as características de uma aplicação deste tipo:

- Visualização de valores de entidades com recurso a animações, alternância de símbolos ou imagens;
- Existência de regras que condicionam o modo de apresentação da informação (côr por exemplo);
- Visualização de informação complementar;
- Possibilidade de execução de telecontrolos (botões, painéis, etc);
- Facilidades de navegação (ampliações, camadas e arrastamento).

2.4.3.2 Listas de Entidades

As listas de entidades constituem outra forma de visualização dos dados recolhidos pelo sistema, através da sua apresentação num formato tabular.

Este tipo de metáfora permite a consulta rápida da informação, pela disponibilização de mecanismos de ordenação e pesquisa, em vez de obrigar ao conhecimento da posição física de uma qualquer entidade num diagrama. A execução de pesquisas socorre-se normalmente da definição de filtros que restringem a quantidade de informação apresentada apenas à desejada pelos operadores.

As listas de entidades são caracterizadas por permitirem a definição de regras de coloração em função do tipo e valor da informação recebida. Estas regras permitem a catalogação gráfica da informação, fazendo, por exemplo, que entidades cujo valor excedeu um determinado limite sejam apresentadas a vermelho, ou que entidades associadas ao disparo de alarmes apareçam com um fundo amarelo, ou com outra qualquer configuração.

Operadores familiarizados com este código de côr são, assim, capazes de identificar de forma rápida qual a informação mais importante e que por isso merece ser analisada com maior cuidado.

Em resumo, as listas de entidades apresentam as seguintes características:

- Visualização de valores de entidades em formato tabular;
- Facilidades de ordenação e pesquisa de valores;
- Definição de filtros;

- Existência de regras que condicionam o modo de apresentação gráfica da informação.

2.4.3.3 Lista de Alarmes

No ponto 2.1.1 foi introduzida a questão da supervisão que inclui naturalmente a capacidade de geração de alarmes. Os alarmes de um sistema SCADA são disparados com o objectivo de alertar os operadores para a ocorrência de situações especiais que deverão ser analisadas de forma mais atenta.

Qualquer uma das aplicações apresentadas até ao momento permite a visualização de informação de alarme. Contudo, nenhuma delas o faz de maneira tão especializada como as listas de alarmes.

Nestas aplicações, os alarmes são apresentados em formato tabular, à semelhança do que acontece com as listas de entidades, beneficiando por isso do mesmo tipo de facilidades de pesquisa e ordenação.

Todos os alarmes apresentados numa lista de alarmes devem ser aceites pelos operadores, por forma a garantir que estes tomaram conhecimento da sua ocorrência.

Também aqui é comum a definição de códigos de cores que indicam o estado dos alarmes: alarme recebido, alarme recebido mas não aceite, alarme aceite mas não resolvido, etc.

Resumindo as características das listas de alarmes:

- Visualização de alarmes em formato tabular;
- Mecanismos de aceitação de alarmes;
- Facilidades de ordenação e pesquisa de valores;
- Possibilidade de execução de filtros;
- Definição de regras de apresentação da informação.

2.4.3.4 Gráficos de Tendência

Sistemas SCADA são importantes fontes de informação, cuja análise constitui uma ferramenta singular para a compreensão do processo que permite a definição de estratégias de actuação a curto, médio e longo prazo. Neste contexto, os gráficos de tendência apresentam-se como meios ideais para a apresentação dos dados.

Num gráfico de tendência, os dados de uma entidade são apresentados em função do tempo, proporcionando uma visão global da sua evolução, sejam eles referentes a informação histórica ou de tempo real.

Em termos de facilidades, estas aplicações permitem a apresentação simultânea de curvas referentes a entidades distintas, definição do tipo de gráfico (linha, barras, etc), definição de aspectos relativos à visualização (côr das linhas, largura dos intervalos, legendas, etc) e possibilidades de pesquisa de informação.

Deste modo, os gráficos de tendência comportam:

- Visualização de curvas temporais de evolução do valor de entidades;
- Valores históricos e de tempo-real;
- Facilidades de pesquisa (para escolha da informação a visualizar);
- Definições gráficas.

2.4.3.5 Análise de Informação Histórica

As aplicações de análise de informação histórica constituem uma especialização das listas de entidades vocacionadas para dados de arquivo. A diferença consiste na possibilidade de realização de filtros que incluem intervalos temporais.

2.4.3.6 Configurador

O configurador é a aplicação menos ligada à operação directa do sistema e que, por isso, não inclui nenhuma das características discutidas até ao momento. Em termos estruturais, um configurador é normalmente uma aplicação 4GL[5] que altera directamente os parâmetros de configuração por manipulação do repositório de dados, normalmente uma base de dados. Este tipo de actividade é vulgarmente executada por administradores do sistema.

Para efeitos da presente dissertação, esta aplicação não constitui matéria de estudo.

2.5 SCADA em Sistemas Eléctricos de Energia

O estudo realizado no âmbito do presente trabalho baseou-se em sistemas SCADA especialmente vocacionados para a gestão de sistemas eléctricos de energia (SEE), visto serem estes os "clientes" naturais dos sistemas desenvolvidos pela EFACEC SE/AS.

A gestão de um sistema eléctrico de energia envolve a utilização de um conjunto diverso de sistemas que são complementares às funcionalidades de controlo e supervisão

[5] 4GL — *Fourth Generation Language*. Linguagem de programação que permite a implementação de interfaces gráficas normalmente vocacionadas para a manipulação de bases de dados.

disponibilizadas por um sistema SCADA. Neste contexto, os sistemas mais comuns incluem:

Aplicações de cálculo eléctrico — incluem a execução de algoritmos especializados para redes eléctricas de energia: trânsito de potências, alocação de cargas, previsão de consumos (*Power Application Systems* — PAS);

Sistemas de informação geográfica (SIG) — sistemas que permitem o tratamento de informação em função de características geográficas;

Sistemas de gestão de incidentes (SGI) — sistemas que fornecem indicações da ocorrência de interrupções de serviço com base da informação recebida directamente pelos clientes da rede eléctrica;

Gestão de ordens de manobra — facilidades que permitem a criação e acompanhamento automático de intervenções de piquetes especializados no terreno;

Comunicação e localização de viaturas (CLV) — gestão da frotas de piquetes por acompanhamento da sua posição e actividade.

A necessidade de interligação destes sistemas com o sistema de supervisão levou ao aparecimento de sistemas complexos, com a capacidade de gerir de forma integrada todas as actividades inerentes ao funcionamento dos sistemas eléctricos de energia. Aos sistemas que integram funcionalidades especialmente ligadas à gestão de redes eléctricas de distribuição de energia (DMS — *Distribution Management Systems*) dá-se o nome de SCADA/DMS [Dis01]. A figura 2.4 apresenta o conceito de integração associado a um SCADA/DMS.

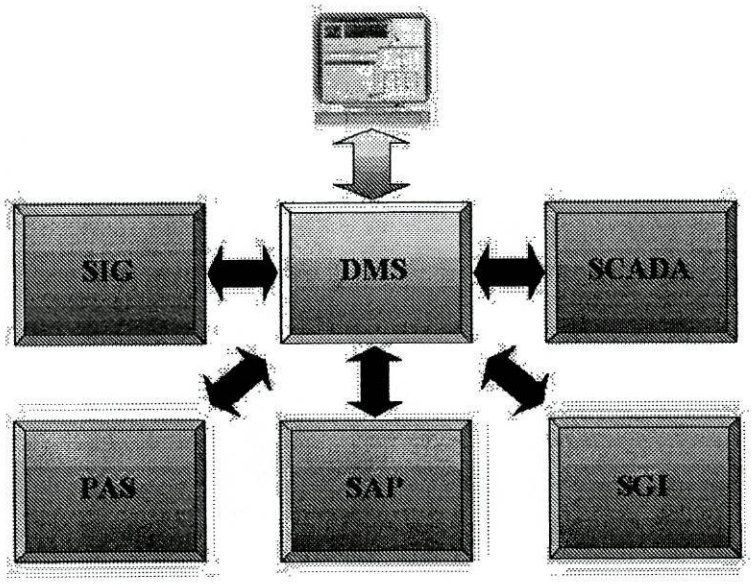


Figura 2.4: Estrutura de um SCADA/DMS

Não existe contudo consenso quanto às funcionalidades que deverão integrar um SCADA/DMS. Para o presente trabalho, considerar-se-á um SCADA/DMS como um sistema SCADA que inclui outras funcionalidades referentes à gestão de redes eléctricas de energia.

2.6 SCADA na EFACEC Sistemas de Electrónica, S.A.

A EFACEC Sistemas de Electrónica, S.A. (EFACEC SE) é uma das empresas do Grupo EFACEC, criada em 1 de Abril de 1991 com um capital social de 4.000.000,00, detido a 100 % pela EFACEC Capital SGPS. A estratégia empresarial da EFACEC SE engloba as seguintes actividades principais [EFA01].:

- Automação de sistemas de energia;
- Sistemas de alimentação;
- Soluções integradas para telecomunicações e transportes;
- Sistemas de transporte;
- Produção de electrónica.

A unidade de negócio de Automação de Sistemas de Energia da EFACEC Sistemas de Electrónica S.A (EFACEC SE/AS) desenvolve e comercializa sistemas de automação e controlo especialmente vocacionados para a gestão de redes de energia. Esta actividade engloba o projecto, desenvolvimento, instalação e colocação em serviço de sistemas para a gestão de produção, transporte e distribuição de energia, assim como para a supervisão e gestão de redes eléctricas ferroviárias [EFA02].

Neste contexto, a EFACEC SE/AS dispõe de uma gama de produtos que lhe permite fornecer soluções completas para controlo e automação de sistemas, desde a aquisição e controlo (local) de subestações, até ao próprio *software* de gestão central [EFA02].

2.6.1 Controlo de Subestações

A EFACEC SE/AS desenvolveu três gamas de produtos especialmente dedicados ao controlo de subestações. A URR (Unidade Remota Reduzida), constitui um equipamento dedicado à comunicação directa com redes de sensores e actuadores pouco numerosas. Já os CLP500 e SCCS500 são equipamentos mais inteligentes, com capacidade de gestão local da subestação, servindo de ligação entre as URRs e o centro de comando. O SCCS500 disponibiliza, inclusivamente, uma interface de operação

homem-máquina (IHM), com algumas funcionalidades SCADA, com vista à intervenção local dos operadores. Estes três equipamentos realizam as funções geralmente atribuídas às URTs, tal como apresentado no ponto 2.2.1.

2.6.2 Centro de Comando

Do ponto de vista do *software* de gestão central, a EFACEC SE/AS disponibiliza dois produtos especialmente desenvolvidos para o efeito: ScateX e GENESys.

2.6.2.1 ScateX

O ScateX é um SGC SCADA distribuído, desenvolvido pela EFACEC SE/AS, no início dos anos 90, para as plataformas DEC Alpha (agora Tru64) e OS/9, encontrando-se actualmente disponível também para a plataforma Windows NT. As linguagens de programação utilizadas para a sua execução foram o C e o C++, tendo o desenvolvimento das interfaces gráficas utilizado as bibliotecas X11/MOTIF e Windows4GL [EFA96]. O ScateX integra com sistemas de gestão de bases de dados relacionais (SGBDR) Ingres e Oracle para registo de configurações e armazenamento histórico de informação [EFA96].

Na sua arquitectura, o ScateX decompõe-se em: servidor (que pode ser duplicado), estações de trabalho e postos de operação. O servidor contém todos os módulos de funcionamento do sistema e serve estações de trabalho e/ou postos de operação, interligando-os ao *front-end* de comunicações para recepção de telemetria. Uma estação de trabalho consiste num servidor local que é actualizado pelo servidor central. Os postos de operação constituem apenas postos remotos que acedem às aplicações residentes tanto no servidor como nas estações de trabalho.

A natureza distribuída do ScateX obriga à existência de mecanismos de comunicação inter-processos baseados em memória partilhada e *sockets* TCP e UDP. O ScateX permite ainda a interligação com outros SGCs através da utilização do protocolo ELCOM-90[6].

De entre as funcionalidades SCADA disponibilizadas pelo ScateX, destacam-se as seguintes [EFA96]:

- Diagramas sinópticos;
- Registos de dados históricos;
- Gráficos de tendência;
- Alarmes;

[6]Protocolo que estabelece um standard para a comunicação entre centros de comando, originalmente desenvolvido na Escandinávia.

- Visualização de entidades.

O ScateX disponibiliza ainda as funcionalidades próprias da gestão de redes de energia eléctrica (funcionalidades DMS) [EFA96]:

- Análise de topologia;
- Análise de curto-circuitos;
- Optimização de rede;
- Planeamento;
- Trânsito de potências;
- Alocação de cargas;
- Estimação de estados.

2.6.2.2 GENESys

O GENESys é o resultado de um projecto conjunto entre a EFACEC SE/AS e a EDP (Electricidade de Portugal) para o desenvolvimento de um sistemas SCADA/DMS de nova geração, no âmbito da execução de um projecto da EDP mais alargado, o projecto SIRED (Sistemas Integrados para Redes Eléctricas de Distribuição).

Em termos estruturais, o GENESys incorpora o núcleo SCADA do ScateX, e adiciona-lhe um conjunto de funcionalidades próprias à gestão de redes eléctricas. Destas funcionalidades, destacam-se:

- Integração com o sistema de informação geográfico (SIG) da EDP baseado no SmallWorld
- Quadro electrónico para afixação de pinos e operação da rede;
- Criação e seguimento de ordens de manobra;
- Execução de estudos baseados na informação recolhida em tempo real;
- Aplicações de energia (as mesmas do ScateX);
- Facilidades de navegação e coloração dos diagramas baseados nas características dos equipamentos (coloração por níveis de tensão, por ponto injector, por ligação galvânica, etc).

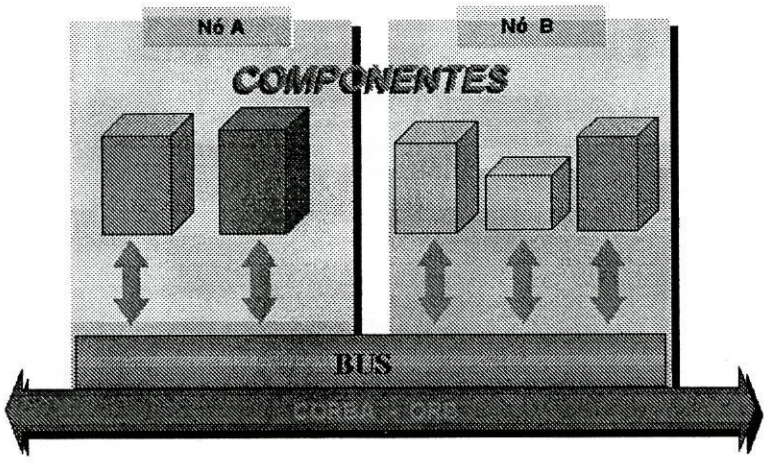


Figura 2.5: BUS — Arquitectura de publicação/subscrição para componentes colaborantes

O GENESys é baseado numa arquitectura de componentes colaborantes que implementa o paradigma de publicação/subscrição de tópicos para a troca de eventos. Esta arquitectura foi desenvolvida em C++ e assenta na plataforma CORBA (*Common Object Request Broker Architecture*).

A figura 2.5 ilustra os componentes que comunicam entre si através da troca de mensagens (eventos) veiculados pelo BUS. O BUS é um *middleware* desenvolvido pela EFACEC SE/AS que implementa as funcionalidades de publicação/subscrição de tópicos, por utilização das características multi-plataforma e multi-linguagem proporcionadas pelo CORBA. Assim, é possível a integração entre componentes desenvolvidos em plataformas distintas de forma transparente.

Uma das características principais do BUS, é o facto dos eventos que nele circulam serem mensagens de texto, facilitando a interoperabilidade com outros sistemas. Assim, a integração de sistemas é possível pela simples implementação de componentes que traduzam a lógica própria desses sistemas em eventos que circulam no BUS[7].

O desenvolvimento deste tipo de componentes permite a obtenção de um sistema perfeitamente integrado para a gestão de redes eléctricas, tal como o GENESys. Exemplos de componentes desenvolvidos com vista à integração são:

- Componente para a interacção com SGBDR (Oracle, Ingres, etc);
- Componente para interacção com o SIG SmallWorld;
- Componente para a interacção com o núcleo SCADA do ScateX.

O GENESys é, assim, um sistema distribuído, multi-plataforma, que incorpora um único meio de comunicação inter-componentes / inter-processos / inter-sistemas — o

[7] *Bridge components.*

BUS. As interfaces gráficas foram desenvolvidas com recurso à biblioteca *Ilog Views* da ILOG [ILOa].

2.7 Integração SCADA / SI

Durante anos, os sistemas SCADA funcionaram isolados do resto dos sistemas de informação (SI) das empresa que serviam, tanto devido ao facto de, em muitos casos, terem antecedido a sua instalação [Rip01], como à própria natureza determinística do fluxo dos seus dados [Wie01].

Esta situação tem no entanto vindo a alterar-se. Por um lado existe a necessidade das empresas diminuírem custos provenientes do facto de terem de gerir várias redes e vários sistemas separadamente. Esta gestão dispersa implica custos de equipamento e pessoas especializadas. Por outro, há a consciência crescente de que a integração de sistemas é um meio eficaz, que permite dotar as empresas de ferramentas mais adequadas para enfrentar os desafios do mercado [Wie01].

Segundo [Wie01], esta integração permite que o resultado final seja um sistema cujo valor supere largamente o valor individual de cada um dos sistemas que o constitui.

O esforço de integração de sistemas SCADA com o resto dos SI de uma empresa envolve:

Integração de redes — as redes que servem sistemas SCADA são normalmente pensadas para suportar taxas de transmissão de dados muito bem conhecidas e devem garantir um funcionamento 24x24 horas. Ora nos SI isto nem sempre é verdade. Se por um lado os requisitos de fluxo de dados não são tão elevados, por outro nem sempre há a necessidade de manter o bom funcionamento dos serviços em horários não laborais, como durante a noite e ao fim de semana. A integração destes dois mundos proporciona uma diminuição significativa em custos de manutenção e instalação, pelo que é compensatório fazer evoluir as redes dos SI por forma a poderem suportar as especificidades próprias de sistemas SCADA.

Integração de dados — para uma empresa que utiliza um SCADA, este será o sistema de informação mais próximo da sua área de negócio. Assim, poder aceder aos seus dados significa obter informações importantes para a própria gestão da empresa, seja através da geração de relatórios ou análise de gráficos baseados em informação obtida directamente do processo.

Integração de aplicações — este é o tipo de integração conseguida, por exemplo, nos sistemas SCADA/DMS (ponto 2.5) e que permite valorizar ao máximo a coexistência entre SCADAS e SI.

2.8 SCADA na Web

A tendência de integração dos SCADA com os SI de uma empresa abre perspectivas interessantes de expansão do sistema com recurso a tecnologias de rede actuais baseadas na internet, em particular, as tecnologias *Web*.

Apresentar soluções baseadas na *Web* significa, antes de mais, tornar os sistemas acessíveis de forma global. Esta característica apresenta factores positivos e negativos.

Do ponto de vista da utilização do sistema, o avanço para a *Web* proporciona uma maneira fácil de interacção com o sistema por parte das pessoas que dele dependem, sejam eles operadores, gestores ou mesmo clientes. Este tipo de aplicações têm ainda a virtude de poderem ser utilizadas a partir de qualquer ponto de acesso à rede, seja ela local ou remota, simplesmente pela utilização de um vulgar *browser Web*. Uma tal filosofia, permite ainda diminuir custos associados à instalação e manutenção de terminais de operação especializados.

Em contrapartida, a disponibilização de acesso global a sistemas SCADA, significa também aumentar a vulnerabilidade de sistemas que, por lidarem directamente com os processos base da actividade das empresas, se querem seguros e afastados de olhares curiosos. Segundo [Wie01], a ligação de um sistema à internet aumenta 8 vezes a probabilidade de sofrer um ataque. Contudo, desenvolver aplicações para a *Web* não implica a sua ligação à rede global. Os sistemas poderão manter-se isolados e mesmo assim disponibilizarem serviços baseados em aplicações *Web*.

Caberá às empresas a definição das suas políticas de disponibilização de serviços através da *Web*, de modo a retirarem o máximo aproveitamento das suas características, ao mesmo tempo que preservam a integridade dos sistemas que controlam.

A presente dissertação não define qualquer tipo de restrição quanto à utilização de aplicações SCADA através da *Web*, limitando-se a estudar a possibilidade da sua implementação.

2.9 Sumário

Sistemas SCADA permitem o controlo, supervisão e aquisição de dados de processos físicos geograficamente dispersos, pela existência de um centro de comando que coordena e comunica com URTs presentes em estações remotas.

O SGC é o elemento do sistema que gere toda a informação e proporciona a interacção com os operadores através de aplicações com características muito bem definidas:

- Apresentação da informação histórica e em tempo real;
- Possibilidade de execução de controlos;

- Apresentação da informação sob diversas formas (sinópticos, listas, gráficos de tendência);
- Características gráficas complexas que incluem imagens e animações;
- Definição de regras que condicionam o modo de apresentação da informação;
- Facilidades de filtragem e navegação (ampliações, arrastamentos, camadas).

A gestão de redes eléctricas de energia requer a utilização de sistemas que implementem funcionalidades específicas à sua natureza, e que são disponibilizadas por sistemas SCADA/DMS.

A EFACEC SE/AS dispõe de uma série de produtos que lhe permite fornecer sistemas SCADA completos. Desses produtos, destacam-se os SGC ScateX e GENESys especialmente vocacionados para a gestão de redes de energia, implementando funcionalidades SCADA/DMS. Este sistemas abrangem uma vasta área tecnológica e permitem a integração com outros sistemas pela utilização do BUS. O BUS é uma plataforma para o desenvolvimento de aplicações distribuídas baseadas em componentes CORBA, que implementa a lógica de publicação/subscrição de tópicos, desenvolvido pela EFACEC SE/AS.

A integração entre sistemas SCADA e os sistemas de informação das empresas proporciona vantagens ao nível da gestão global da empresa, tanto do ponto de vista de redução de custos, como em termos de disponibilização generalizada da informação. Esta tendência abre perspectivas de expansão dos sistemas SCADA para outros ambientes, nomeadamente para a *Web*.

Capítulo 3

Aplicações na World Wide Web

Este capítulo introduz a questão do desenvolvimento de aplicações na *World Wide Web*. Nele se descrevem os elementos fundamentais da sua arquitectura, as tecnologias mais comuns e os modelos existentes para o desenvolvimento de aplicações.

No final, apresenta-se o resultado da recolha de informação sobre alguns sistemas SCADA que apresentam soluções baseadas na *Web* e que são, por isso, interessantes para a realização da presente dissertação.

3.1 Um Pouco de História

Foi no início da década de noventa que um cientista do Centro Europeu de Pesquisa Nuclear (CERN), Tim Berners-Lee, desenvolveu o que viria a ser o embrião da *World Wide Web*. A sua motivação era a de facilitar o modo como os cientistas do CERN acediam à informação que se encontrava espalhada por uma série de sistemas diferentes entre si. Estas diferenças tornavam o processo de procura e obtenção de informação numa actividade complexa, exigindo o acesso a vários terminais ligados a computadores diferentes e requerendo o conhecimento de um conjunto vasto de programas [BL96].

A solução encontrada foi a de desenvolver um sistema que unificasse o modo de acesso independentemente do local físico, plataforma e tipo de informação desejada. Nesse sistema, a uniformização do acesso era assegurada por um único programa (navegador[1]), baseado numa infra-estrutura de comunicação simples, sem estado e rápida, proporcionada pelo protocolo de transferência de hipertexto, ou HTTP (*HyperText Transfer Protocol*). Berners-Lee definiu igualmente uma linguagem que seria reconhecida pelo navegador e que serviria como forma de estruturação e apresentação de informação:

[1] O termo *navegador* será utilizado para identificar o vulgar *browser Web*.

o HTML (*HyperText Markup Language*). Para além de ser um subconjunto de uma outra linguagem de anotação, o SGML, o HTML incorporava facilidades de navegação pela informação através de ligações, identificadas por URIs (Identificador de Recurso Universal), que permitiam aceder a outros documentos, fossem eles de texto, sons ou imagens. Este conceito de hipertexto[2], apesar de não ser completamente novo na altura, permitiu tornar a *Web* num sistema único de interligação de informação até então dispersa e isolada. Como peça final do sistema, Berners-Lee juntou-lhe o servidor HTTP, ou seja, o programa que nos servidores, disponibiliza a informação em HTML através do protocolo HTTP.

No sistema desenvolvido, *WorldWideWeb* foi o nome do navegador criado por Tim Berners-Lee. Mais tarde, decidiu mudar-lhe o nome, com o intuito de diferenciar entre a aplicação e o próprio sistema desenvolvido que ficou definitivamente conhecido como *World Wide Web.*

Com vista à divulgação do sistema recém-criado, Nicola Pellow, outro cientista do CERN, desenvolveu um cliente *Web* (navegador), com interface de texto. O facto desta aplicação ser bastante portável, permitiu a sua rápida disseminação por várias plataformas. Mais tarde, em 1993, apareceu o *Mosaic*, um navegador desenvolvido no *National Center for Supercomputing Applications* (NCSA) por Marc Adressen, e que por apresentar um interface gráfico apelativo, ser simples de usar e estar disponível gratuitamente, terá sido o primeiro grande impulsionador da *Web* [BL96].

3.2 Sistema Universal

O aparecimento da *Web* permitiu a generalização do acesso à informação pela definição de uma arquitectura universal [CH98]. Essa arquitectura baseia-se em cinco elementos fundamentais: o cliente, o servidor, a linguagem de apresentação, o protocolo de comunicação e o mecanismo de identificação de recursos.

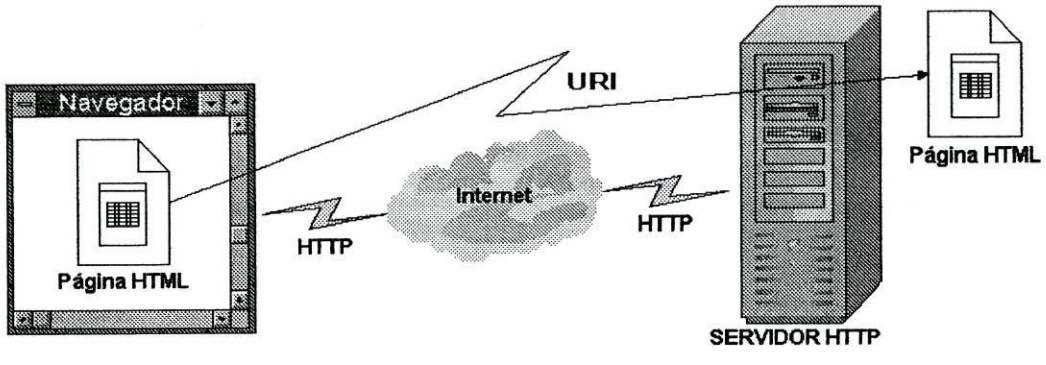


Figura 3.1: Elementos fundamentais da Web

[2]A capacidade de interligação com documentos contendo texto, sons ou imagens é vulgarmente designada por hipermédia.

A figura 3.1 ilustra os cinco elementos que integram a *World Wide Web.*

3.2.1 Navegador — O cliente universal

O navegador é a parte visível da *Web*, na medida em que é a aplicação que permite aos utilizadores acederem à informação por ela disponibilizada. Assim, e do ponto de vista do utilizador, a *Web* é muitas vezes reconhecida como sendo o próprio navegador [Tre99].

O navegador incorpora funcionalidades de comunicação e de visualização; As suas funcionalidades de comunicação permitem-lhe trocar informação com servidores *Web* em qualquer parte de uma rede, seja ela uma rede local ou a própria internet, pela utilização do protocolo HTTP. Assim, um navegador envia pedidos HTTP ao servidor indicando qual a informação desejada e recebe respostas HTTP contendo a referida informação na forma de documentos HTML. São as suas funcionalidades gráficas que lhe permitem depois interpretar o código HTML e apresentá-lo ao utilizador. Cada documento HTML é vulgarmente designado por página HTML.

O navegador permite que utilizadores circulem por entre páginas HTML através da selecção de porções de texto que constituem ligações para outros documentos (hipertexto), transformando-os num único documento virtual.

Para além destas funcionalidades, os navegadores permitem normalmente a utilização de outros serviços de rede tais como a transferência de ficheiros por FTP, o envio de mensagens de correio electrónico e outras.

Exemplos de navegadores comuns são o *Internet Explorer* da *Microsoft* e o *Netscape Navigator* da *Netscape*, entre outros.

3.2.2 Servidor Web — O servidor universal

Do outro lado do sistema encontra-se a aplicação conhecida como servidor *Web* ou servidor HTTP. A sua função é a de fornecer os documentos HTML requisitados pelos clientes via HTTP.

Na sua versão mais simples, um servidor disponibiliza apenas documentos com informação estática e residentes num sistema de ficheiros local. Como veremos mais adiante, um servidor HTTP permite actualmente a integração com um conjunto vasto de tecnologias que proporcionam a geração dinâmica de páginas HTML.

Exemplos de servidores HTTP são o *Apache* da *Apache Software Foundation* e o *Internet Information Server* da *Microsoft.*

3.2.3 HTML — A linguagem universal

HTML [W3C99a] é o acrónimo inglês de *Hypertext Markup Language*, ou numa tradução livre, *Linguagem de Anotação e de Hipertexto.* Esta é a linguagem usada para a codificação da informação que circula na *Web* [SS96].

O HTML é uma versão reduzida do SGML (*Standard Generalized Markup Language*) que é uma linguagem que permite a definição de outras linguagens de anotação. Numa linguagem de anotação, a informação incorpora elementos especiais que servem para a caracterizar. Com base nestes elementos, ou anotações, a informação poderá ser tratada convenientemente pelos intervenientes [RR00]. No caso do HTML, as anotações são identificadas por elementos delimitados pelos caracteres '<' e '>', tal como no SGML. Contudo, e ao contrário do SGML, o conjunto dos elementos de anotação do HTML é bem definido, fazendo com que não seja possível a criação de novos elementos de anotação. Apesar de limitado, os tipos de elementos definidos no HTML são, salvo algumas excepções, suficientemente generalistas e descritivos para que documentos HTML possam ser apresentados em terminais de texto, ambientes gráficos ou poderem mesmo ser reproduzidos sonoramente [GP02]. Terá sido, aliás, devido à sua simplicidade que o HTML estabeleceu-se como uma verdadeira linguagem universal [GP02].

Para além de linguagem de anotação, o HTML é também um sistema de hipertexto; num sistema de hipertexto, a informação pode conter elementos que identifiquem outras fontes de informação, sendo possível a navegação entre elas. Em HTML estas *hiperligações* são definidas por *Uniform Resource Identifiers* (URI) que identificam outras fontes de informação, sejam elas outros documentos HTML, sons, imagens ou vídeo [BL].

A sintaxe e semântica do HTML são definidas pelo *World Wide Web Consortium* (W3C), presidido por Tim Berners-Lee.

Nos primeiros anos da *Web*, assistiu-se a uma tendência por parte dos fabricantes de navegadores no sentido de criarem extensões aos elementos básicos do HTML, com vista à obtenção de quotas de mercado maiores. Estas extensões proprietárias eram sobretudo no sentido de definir a apresentação do documento, pondo assim em causa o carácter generalista e interoperável da *Web* [GP02][3].

Como forma de tentar manter a linguagem sob controlo, o W3C foi produzindo novas versões que incorporavam parte dos elementos definidos pelas empresas [HB98]. Se bem que esta atitude lhe permitiu diminuir o "surto" de extensões "não autorizadas", teve no entanto o efeito perverso de afastar a linguagem das suas raízes generalistas, por a tornar excessivamente dependente da forma em detrimento do conteúdo. Veremos mais adiante como a introdução do XML (outra linguagem baseada no SGML) pode ser utilizada para ultrapassar estas questões.

[3] O elemento `<blink>`, por exemplo não pode ser implementado em alguns dispositivos.

Outra das atitudes do W3C, foi a de introduzir o conceito de folhas de estilo, as *Cascading Style Sheets* [W3C98] (CSS), com vista à separação entre apresentação e conteúdo. Assim, um documento HTML deve conter apenas informação relativa ao âmbito do seu problema, deixando que os critérios de formatação sejam estabelecidos por folhas de estilo exteriores, evitando a necessidade de utilização de elementos proprietários [But01].

3.2.4 HTTP — O protocolo universal

O HTTP [IET99] é o protocolo de comunicação utilizado entre servidores e clientes *Web* para a transferência de dados e, por isso, independente da plataforma destes.

HTTP é o acrónimo inglês para *HyperText Transfer Protocol*, ou em português, *Protocolo para a Transferência de Hipertexto* e é definido pelo IEFT *Internet Engineering Task Force*. A versão corrente do protocolo é a HTTP/1.1 [IET99].

Tal como definido em [IET99], o HTTP é um protocolo aplicacional para sistemas de informação de hipermédia, distribuídos e colaborantes. É um protocolo genérico, que pode ser usado para outros fins, que não somente a transferência de hipertexto, tais como em servidores de domínios e sistemas baseados em arquitecturas distribuídas, através da introdução de extensões suportadas pelo protocolo. Uma característica do HTTP é a possibilidade de negociação do tipo de dados a transferir permitindo que os sistemas possam ser desenvolvidos independentemente dos dados que forem transmitidos.

É importante realçar que o HTTP é um protocolo sem manutenção de estado e de sentido único; o envio de um pedido HTTP é efectuado pelo cliente, estabelecendo uma ligação temporária com o servidor, que termina assim que a resposta é devolvida.

3.2.5 URI — O identificador universal

O URI [IET98] (*Uniform Resource Identifier*) é uma porção de texto que identifica recursos na *Web*: documentos, imagens, ficheiros, serviços, etc. Segundo [IET98], um URI pode ser classificado como URL (*Uniform Resource Locator*) ou URN (*Uniform Resource Name*). Um URL diz respeito ao subconjunto dos URIs que identificam o método de acesso primário ao recurso, em vez de o identificarem por um nome ou por outros quaisquer atributos. Assim, a definição do método de acesso de um recurso pode incluir o protocolo (http, ftp, telnet, etc), o endereço do servidor (www.fc.up.pt, www.google.com, etc), a porta, o caminho do servidor e, finalmente, o recurso. Genericamente, um URL tem o seguinte formato:

```
protocolo://servidor:porta/caminho/recurso
```

Exemplos deste tipo de URLs são:

`http://www.fc.up.pt:80/index.html`

`ftp://foo.bar/image.png`

Existem contudo outros URLs válidos com formatos diferentes. Um exemplo:

`mailto:rfernandes@se.efacec.pt`

Já os URNs são aqueles URIs que pretendem identificar um recurso de forma global, mesmo que se encontre inacessível ou que já não exista [IET98]. Uma metáfora interessante para descrever um URN seria o modo de identificar univocamente alguém através do seu número de bilhete de identidade. Assim, essa pessoa poderá sempre ser identificada, mesmo que, por exemplo, altere a sua residência.

3.3 Tecnologias para Aplicações na Web

Nos primeiros anos, a *Web* existiu como um grande repositório de informação, mais ou menos estática, que ia sendo visitada pelos utilizadores interessados. Nessa altura, o entusiasmo das pessoas devia-se à facilidade com que os seus olhos atravessavam documentos sobre todo o tipo de assuntos e oriundos de virtualmente qualquer parte do mundo. Em termos estruturais, um documento *Web* não era mais do que um ficheiro HTML guardado numa base de dados ou sistema de ficheiros de um qualquer servidor no mundo. A informação estava lá para ser lida, ouvida e analisada, independentemente do tipo de utilizador que o fazia. O servidor de então, era uma espécie de guarda-livros que ia "despejando" as páginas ao sabor das vontades dos que as requeriam. Não tardou muito até que se percebesse que o mesmo tipo de informação poderia ser gerada dinamicamente por programas especializados presentes no servidor.

Nesse momento, nasceram as aplicações *Web* [Pip99].

3.3.1 Common Gateway Interface

O CGI (*Common Gateway Interface*) foi um dos primeiros métodos utilizados para a criação de aplicações *Web*. O CGI define um protocolo de comunicação entre o servidor HTTP e aplicações escritas noutras linguagens, tal como o Perl, C, C++, Java, VisualBasic, etc [But01].

Com os CGI's, a informação pode ser introduzida em formulários HTML e depois enviada para o servidor. O servidor passa então estes pedidos aos *scripts* ou programas externos, que os processam e, como resultado, produzem um novo documento HTML que é enviado de volta ao cliente. Durante o processamento dos pedidos, os CGIs podem efectuar tarefas complexas como a comunicação com outros processos ou o acesso a bases de dados.

Apesar de simples e dos seus resultados serem extremamente interessantes, as aplicações

baseadas em CGIs apresentam problemas relativos ao processamento concorrente de pedidos. De facto, cada pedido enviado ao servidor dá origem à execução um novo CGI para o servir, que termina assim que a resposta é devolvida ao cliente [But01]. Ora, a criação de processos requer tempo e recursos do servidor (memória e processador), tornando o processamento concorrente de pedidos numa tarefa ineficiente e de capacidade limitada [HC98].

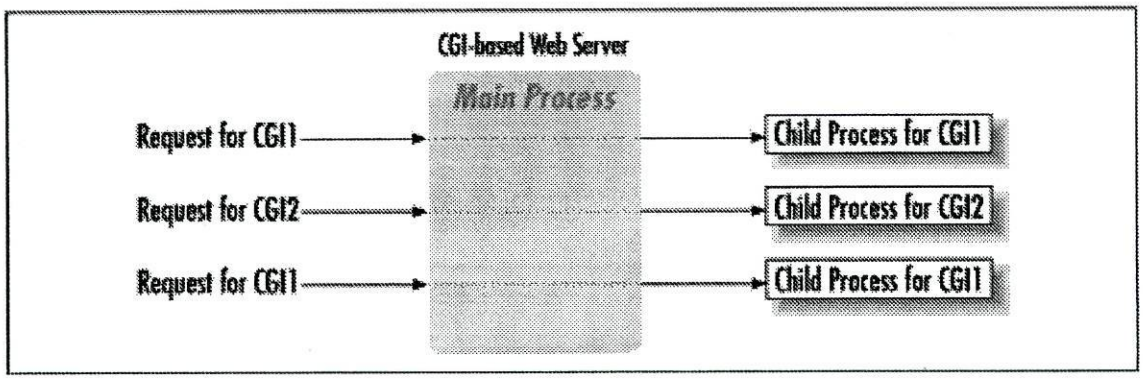


Figura 3.2: Ciclo de processamento de um CGI

Outro problema dos CGIs, é o facto dos seus processos serem externos ao servidor HTTP, o que faz com que um processo não possa interagir com o servidor e assim aproveitar as suas potencialidades.

3.3.2 FastCGI

Como forma de ultrapassar as limitações dos CGIs, foram desenvolvidas algumas técnicas que incidem principalmente na rapidez de execução dos programas que servem os pedidos. O *FastCGI* por exemplo, permite que todos os pedidos recebidos pelo servidor para um determinado CGI, sejam direccionados para um único processo persistente já em execução, reduzindo o ónus associado ao arranque de novos processos. Ainda assim, o modo do *FastCGI* servir pedidos concorrentes de forma eficiente é através da existência de conjuntos de processos iguais (*pool*) que vão sendo atribuídos aos pedidos à medida das suas disponibilidades.

Apesar dos melhoramentos introduzidos pelo *FastCGI*, este continua a basear-se em processos exteriores ao servidor HTTP e logo sem a possibilidade de interagir com ele.

3.3.3 Módulos de Scripting

Outras soluções, passaram por introduzir nos servidores *Web* a capacidade de interpretação de linguagens como o Perl, acelerando o processamento dos pedidos pela sua execução directamente no servidor. Os CGIs baseados nessas linguagens podem

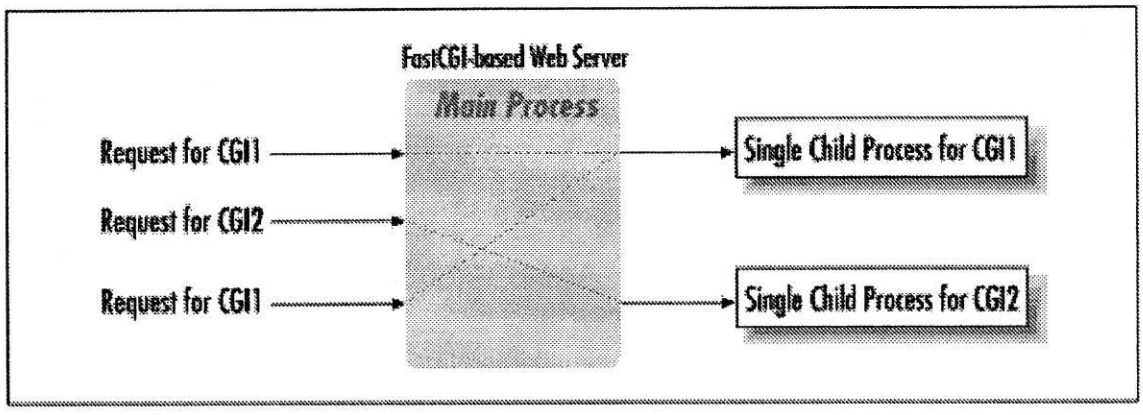


Figura 3.3: Ciclo de processamento de um FastCGI

então ser interpretados directamente pelo servidor sem a necessidade de se proceder à execução de processos exteriores, tornando o tempo de resposta aos pedidos bastante menor. O problema desta abordagem, reside no facto de não haver suporte unificado das mesmas linguagens em todos os servidores HTTP, tornando os CGIs assim desenvolvidos pouco portáveis, mesmo quando baseados em linguagens naturalmente independentes das plataformas. Exemplos deste tipo de módulos são o *mod_perl* (suportado pelo Apache) e o *PerlEx*.

3.3.4 API de Extensões ao Servidor

Algumas companhias desenvolveram interfaces de programação de aplicações (API) específicas para os seus servidores HTTP, de modo a que fosse possível o desenvolvimento de extensões às suas funcionalidades, que seriam executadas em vez de CGIs. Estas extensões são muitas vezes desenvolvidas em linguagens como o C e o C++, fazendo com que a sua execução seja extremamente rápida.

Um exemplo deste tipo de abordagem pode ser encontrado no *Microsoft Internet Information Server* que disponibiliza a *Internet Server Application Programming Interface* (ISAPI) para o desenvolvimento de extensões.

Embora esta solução apresente níveis de desempenho notáveis, existem problemas associados à sua utilização. Em primeiro lugar, cada vendedor disponibiliza a sua própria API para o desenvolvimento de extensões, fazendo com que as soluções sejam altamente dependentes da plataforma utilizada. Por outro lado, as extensões são difíceis de desenvolver e de manter, podendo mesmo pôr em risco a segurança e fiabilidade do sistema; erros durante a execução das extensões podem tornar o servidor inoperacional, levando consequentemente a uma indesejável quebra de serviço.

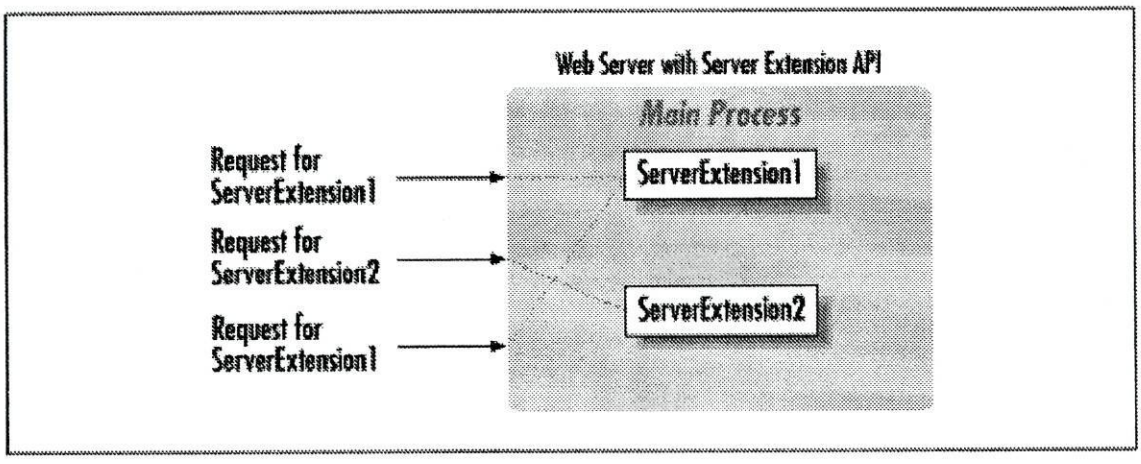


Figura 3.4: APIs de extensões ao servidor

3.3.5 Java Servlets

As Java *Servlets* constituem uma tecnologia desenvolvida pela *Sun* com o intuito de resolver os problemas de ineficiência próprios dos CGI [But01].

Uma *servlet* consiste numa unidade computacional situada no servidor, que funciona como um extensão genérica deste, expandindo as suas funcionalidades [HC98]. Uma *servlet* é uma classe que é carregada pelo servidor correndo num ambiente perfeitamente controlado, ou seja, dentro da máquina virtual Java, tornando-a segura e portável.

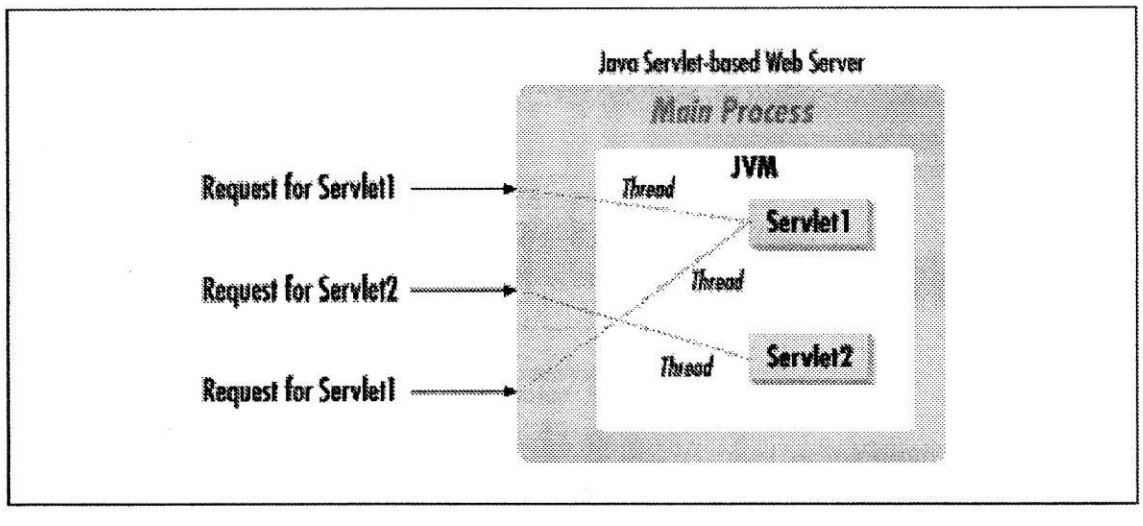


Figura 3.5: Java Servlets

Ao contrário dos CGIs, que usam múltiplos processos para a execução de diferentes programas e/ou diferentes pedidos, as *servlets* são executadas em *threads* distintos dentro do servidor *Web*, o que as torna altamente eficientes e escaláveis. O facto da

sua execução ocorrer dentro do servidor faz com que elas possam aceder directamente às suas funcionalidades, algo que não acontecia com os CGIs [HC98].

As *servlets* originais não haviam sido desenhadas para utilização exclusiva em servidores *Web*, elas apenas recebiam pedidos de clientes genéricos, processavam-nos e devolviam as respostas. Deste modo, uma *servlet* pode ser usada para extender as funcionalidades de qualquer tipo de servidor desde que este disponibilize uma máquina virtual para a sua execução. Numa versão posterior, a *Sun* introduziu a API para as *http servlets*, as quais permitem obter as características do cliente *Web* que efectuou o pedido [Pip99] (disponível no cabeçalho HTTP) e, assim, adequar a resposta.

Outra das vantagens das *servlets* é a sua portabilidade tanto entre sistemas operativos (devido às várias implementações disponíveis da máquina virtual Java), como entre servidores HTTP, já que é suportada pela maioria deles.

3.3.6 Scripting no Servidor

A utilização de *scripting* no servidor constitui uma alternativa à utilização de CGIs ou *servlets* para o desenvolvimento de aplicações *Web* [But01].

Nesta abordagem, introduz-se código dentro das páginas HTML que é executado durante o seu processamento. Assim, as páginas produzidas são, em regra geral, mais pequenas que as suas congéneres obtidas com recurso a CGIs e *servlets*, levando à rápida produção de sítios *Web*.

As primeiras versões de linguagens que utilizavam esta tecnologia faziam com que o desenvolvimento de aplicações fosse demasiado centrado nas páginas, tornando difícil a reutilização de código entre páginas. As novas versões destas linguagens têm tentado resolver este problema [But01].

Exemplos de linguagens que utilizam esta tecnologia são:

- *Java Server Pages* (JSP), que são baseadas em Java e foram introduzidas pela *Sun Microsystems*. Durante a sua execução, as páginas são transformadas em Java *Servlets*, beneficiando das suas características, nomeadamente no que respeita à eficiência de execução de pedidos. Na sua versão mais recente, as JSPs são capazes de utilizar objectos Java (Java *Beans*[4]) proporcionando um mecanismo de reutilização de código e separação entre conteúdo e apresentação.

- As *Active Server Pages* (ASP) foram desenvolvidas pela *Microsoft* e podem utilizar JScript, Perl ou VBScript como linguagens de *scripting*. Na plataforma *Windows*, as ASP podem basear a sua implementação na utilização de objectos COM (*Component Object Model*), à imagem do que as JSP fazem com os Java

[4]Os Java Beans são classes Java que seguem uma norma de desenho e de nomenclatura dos seus membros (métodos a atributos) definida pela API para Java Beans.

Beans. A utilização de objectos COM faz com que o desenvolvimento de páginas ASP seja efectuado em virtualmente qualquer linguagem de programação [HC98], sofrendo contudo da conhecida ineficiência inerente à utilização desses objectos [But01]. As ASP disponibilizam ainda métodos de acesso a bases de dados através da utilização de *Active Data Objects* (ADO) da *Microsoft* e por utilização de SQL[5].

- O PHP (*PHP: Hipertext Preprocessor*) é uma linguagem de *scripting* baseada em C que foi desenvolvida sob a égide da *Apache Software Foundation*. Tal como as ASP, também o PHP disponibiliza métodos de acesso a bases de dados (através do conjunto extenso de bibliotecas existentes) que, em versões mais recentes, permitem a gestão optimizada das ligações [But01].

3.3.7 eXtensible Markup Language (XML)

O XML [W3C00] (*eXtensible Markup Language*) é uma linguagem de anotação derivada do SGML (tal como o HTML), que permite a estruturação e descrição da informação [But01]. Ao contrário do que acontece com o HTML, com o XML é possível definir o conjunto das anotações que serão utilizadas pelos documentos, permitindo a adequação da sua estrutura à natureza da própria informação que contêm. A definição destas gramáticas é efectuada com o recurso a DTDs (*Document Type Definition*) ou esquemas XML (eles próprios, documentos XML que seguem uma gramática pré-definida).

Um documento XML não faz qualquer assunção quanto à forma como deve ser apresentado, ao contrário do HTML. O exemplo seguinte, adaptado de [GP02], ilustra uma hipotética representação da mesma informação em HTML e XML.

Em HTML:

```
<b>P200 Laptop</b>
<br>Microloja
<br>1500 euros
```

Em XML:

```
<produto>
  <modelo>P200 Laptop</modelo>
  <disponivel>Microloja</disponivel>
  <preco>1500</preco>
  <moeda>euro</moeda>
</produto>
```

[5]Com a plataforma empresarial .Net apareceu a mais recente evolução desta tecnologia, as ASP.Net.

Na versão HTML, existe uma clara alusão à forma como deve ser apresentada a informação, mas sem uma estruturação óbvia da mesma. O modelo do computador aparecerá impresso a negrito, e o resto da informação é apresentada em linhas distintas. Será errado assumir, por exemplo, que toda a informação escrita a negrito se refere a modelos de computador.

No caso do XML, a informação segue uma estrutura delimitada pelas anotações que se coadunam com a sua natureza e que foram definidas numa gramática: `produto`, `modelo`, `disponivel`, `preco` e `moeda`. Contudo, é importante realçar que a definição de uma gramática apenas permite garantir que a informação se apresenta conforme a uma estrutura bem definida (sintaxe), não fazendo qualquer alusão acerca do significado da mesma (semântica). Assim, é possível definir, por exemplo, que um produto pode ter vários modelos, ou que o seu preço deve ser um valor numérico, não sendo contudo possível dizer se a anotação `<disponivel>` se refere a um vendedor, a um fabricante, ou se informa da disponibilidade (existência ou não) do produto. Apesar desta limitação, a utilização de XML permite que dois intervenientes que conheçam a natureza do problema (semântica) possam concordar num modelo objectivo e bem definido para a troca de informação e sua validação.

Existem esforços por parte do W3C para a definição de modelos que permitam caracterizar a informação (semântica) através da utilização de um mecanismo conhecido como RDF [W3C99b] (*Resource Description Framework*).

A clara distinção entre conteúdo e apresentação introduzida pelo XML, é a base de um conjunto de novas abordagens para o desenvolvimento de aplicações na *Web*.

3.3.7.1 eXtensible Stylesheet Language Transformation (XSLT)

XSLT [W3C99d] (*eXtensible Stylesheet Language Transformation*) é uma linguagem baseada em XML que permite transformar documentos XML noutros documentos, com base na definição de regras [But01]. Esta linguagem é uma das três partes que formam a linguagem para definição de folhas de estilo em XML, ou seja, o XSL [W3C] (*eXtensible Stylesheet Language*). As outras duas partes são o XPath [W3C99c] (*XML Path Language*), que é usada pelo XSLT para aceder ou referir documentos XML, e o XSL-FO [W3C01a] (*XML Formatting Objects*), uma gramática XML que especifica a formatação das folhas de estilo.

Um processador XSLT funciona em função de regras definidas numa folha de estilos que é depois interpretada e aplicada ao documento XML. A figura 3.6 ilustra o funcionamento de um processador XSLT.

O processador XSLT recebe um documento XML e aplica as transformações descritas na folha de estilos XSL. O resultado final é um novo documento conforme às regras de aplicação definidas, podendo ser algo tão diverso como outro documento XML, um documento HTML, ou outro qualquer tipo de documento.

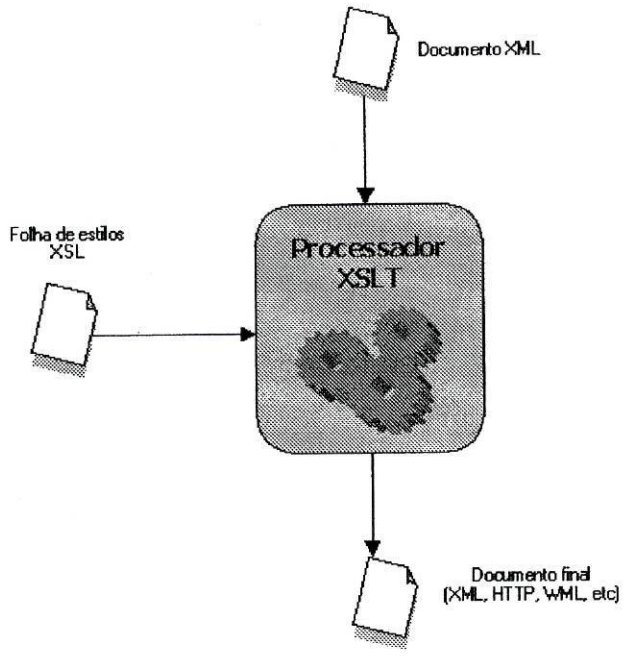


Figura 3.6: Processador XSLT

A utilização de transformações XSLT permite que a mesma informação descrita em XML possa ser facilmente visualizada em vários dispositivos, pela aplicação de folhas de estilo próprias para cada um deles, tornando clara a separação entre forma e conteúdo [GP02].

3.3.7.2 Scalable Vector Graphics (SVG)

SVG (*Scalable Vector Graphics*) é uma linguagem para a descrição bidimensional de gráficos em XML definida pelo W3C. O SVG permite a utilização de três tipos de objectos gráficos: figuras geométricas, texto e imagens. Os objectos gráficos podem ser agrupados, transformados ou compostos noutros objectos. A linguagem define um conjunto de funcionalidades que permitem o tratamento gráfico dos objectos, tais como aplicação de filtros e definição de áreas de redesenho[6] [W3C01b].

Os desenhos efectuados em SVG podem ser interactivos e dinâmicos; animações podem ser definidas e "arrancadas" declarativamente (por introdução de elementos SVG vocacionados para animações) ou através da utilização de *scripting* [W3C01b].

A visualização de documentos SVG implica a existência de programas especializados que saibam interpretá-lo. O projecto *Batik* da *Apache Software Foundation* disponibiliza aplicações e bibliotecas gráficas especificamente criadas para o efeito. Existem também *plug-ins* que permitem a visualização directa de documentos SVG a partir de um navegador.

[6]Este termo pretendo substituir a expressão inglesa ***clipping path*** que define a porção de uma imagem a ser redesenhada.

3.3.7.3 Simple API for XML (SAX)

O SAX (*Simple API for XML*) é uma API, definida por um grupo de discussão de XML conhecido como XML-DEV [GP02], para processadores de XML baseados em eventos. Esta API define um modelo sequencial de descodificação de documentos XML. Pacotes que implementem esta API, permitem que as aplicações que os utilizem possam receber notificações à medida que o processamento de documentos vai sendo efectuada, possibilitando a extracção da informação neles contida de forma eficiente [But01].

3.3.7.4 Document Object Model (DOM)

O DOM (*Document Object Model*) é outra API para processadores de documentos XML, criada pelo W3C. Num processador DOM, o documento XML é representado em memória numa estrutura em árvore que pode ser interrogada pelas aplicações [But01].

Um processador DOM permite a manipulação da sua árvore (edição de XML) e a geração do documento XML que representa.

Devido à necessidade de manter toda a estrutura em memória, um processador DOM é, geralmente, mais lento do que um processador SAX [GP02].

3.3.8 Serviços Web

Um serviço *Web* é uma interface de rede para uma determinada funcionalidade baseada em tecnologias *internet* comuns[7] [Yan02].

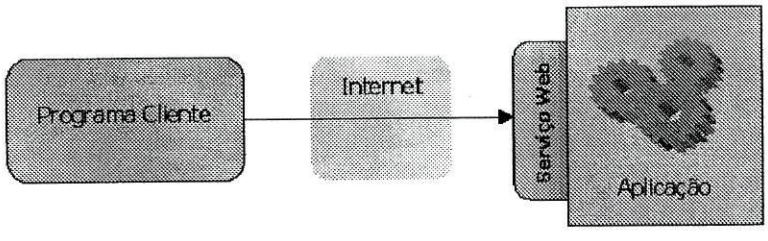


Figura 3.7: Acesso a um serviço Web através da internet

A figura 3.7 ilustra o conceito de serviço *Web*, onde um cliente utiliza a funcionalidade disponibilizada por uma aplicação através de uma interface de rede disponível via internet.

Uma vez que os serviços *Web* são baseados em protocolos *standard*, então é possível a integração de serviços desenvolvidos em plataformas e linguagens diferentes através de um processo comum.

[7] Aqui o termo *comum* refere-se às tecnologias da *internet* tidas como *standard*

Segundo [BS01], são cinco os pontos que caracterizam um serviço *Web*:

Componente reutilizável de *software* — as funcionalidades disponibilizadas pelo serviço podem ser utilizadas por vários clientes.

Componente fracamente ligado a outros componentes — a construção de uma aplicação baseada em serviços *Web* não se restringe à coexistência física dos componentes num mesmo servidor ou aplicação.

Encapsulamento — encapsula uma funcionalidade discreta e descreve a forma como deve ser utilizado.

É acessível programaticamente — por permitir a troca de dados com outros componentes de *software*.

É acessível pela *internet* através dos seus protocolos — um serviço *Web* é suficientemente genérico para utilizar qualquer protocolo (HTTP, SMTP, etc), sendo contudo o HTTP o mais utilizado.

3.3.8.1 Tecnologias dos Serviços Web

A pilha tecnológica dos serviços *Web* assenta noutras tecnologias que permitem a sua execução. São elas as tecnologias de rede, de transporte, de empacotamento, de descrição e de procura. Apresentamos uma breve descrição de cada uma:

Rede — a troca de mensagens entre computadores só é possível com base na existência de uma rede de interligação. Na maioria dos casos, essa rede é a própria internet, mas pode ser qualquer outra.

Transporte — o transporte das mensagens pode ser efectuado por utilização de qualquer protocolo de comunicação, sendo o HTTP o mais comum.

Empacotamento — para que a troca de mensagens entre sistemas distintos seja viável, é necessária a existência de um protocolo que defina o formato das mensagens transaccionadas e que seja independente da plataforma. Um desses protocolos é o SOAP (*Simple Object Access Protocol*), que sendo baseado em XML, permite o empacotamento de pedidos e respostas num formato universal (texto).

Descrição — para ser possível a utilização de um serviço *Web*, este deve disponibilizar informação acerca da forma como deve ser utilizado, ou seja, a descrição da sua interface pública. Esta informação pode ser apresentada através de uma linguagem também baseada em XML, conhecida como WSDL (*Web Service Description Language*).

Procura — por fim, a utilização de um serviço *Web* de forma generalizada implica que haja um meio de clientes tomarem conhecimento da sua existência. Esta funcionalidade pode ser implementada com o recurso a UDDI (*Universal Description, Discovery and Integration*).

O seguinte exemplo retirado de [Yan02], permite ilustrar o modo de actuação de um serviço *Web* especializado na correcção ortográfica de erros de documentos:

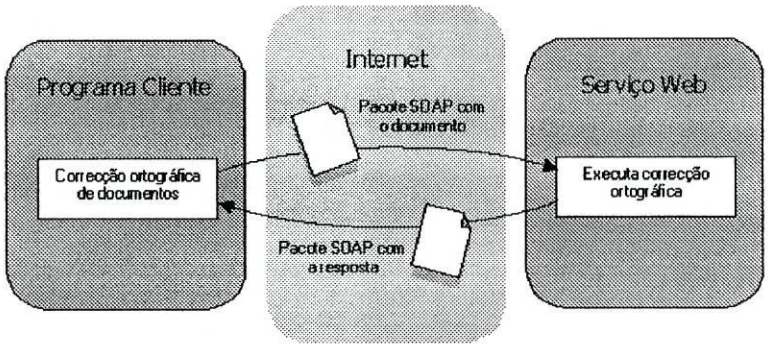


Figura 3.8: Serviço Web para a correcção ortográfica de documentos

O processamento da figura divide-se em 5 passos:

1. O programa cliente "embrulha" o documento a ser corrigido numa mensagem SOAP;
2. A mensagem SOAP é enviada para o serviço *Web* no corpo de um pedido HTTP do tipo POST;
3. O serviço *Web* "desembrulha" a mensagem SOAP e converte-a num formato próprio para processamento. A aplicação procede à verificação dos erros ortográficos contidos no documento e gera uma lista com eles;
4. O serviço *Web* "empacota" a resposta (a lista de erros ortográficos) numa mensagem SOAP que é devolvida ao cliente como resposta ao pedido HTTP;
5. O cliente "desempacota" a mensagem SOAP e obtém o resultado da verificação ortográfica.

3.3.9 Plug-ins

O *plug-in* é um mecanismo utilizado pelos navegadores que lhes permite extender as suas funcionalidades pela adição de pequenos módulos especializados. Com estes módulos, o navegador passa a ser capaz de interpretar formatos desenvolvidos por terceiros, potenciando as suas capacidades originais. Exemplos de *plug-ins* populares são o *Acrobat Reader Plug-in* para a visualização de ficheiros em formato PDF (*Portable*

Document Format); o *Flash Plug-in* para a execução de animações em *Flash*; o *Java Plug-in* que proporciona uma máquina virtual para a execução de *applets*, etc.

3.3.10 Código no Cliente

Em meados nos anos 90, as aplicações *Web* começaram a deixar o servidor em direcção ao cliente. A razão foi em parte devido ao peso da execução de CGIs no servidor, em parte devido às novas potencialidades oferecidas por linguagens emergentes como o Java [Pip99].

Apresentam-se de seguida as tecnologias mais importantes para a implementação de aplicações *Web* que apresentam código no cliente.

3.3.10.1 Applets Java

As *Applets* Java são pequenas aplicações descarregadas pelo servidor *Web* através da rede e que executam localmente dentro de uma máquina virtual disponibilizada pelo navegador. Devido à ampla divulgação dos navegadores, as *applets* constituem uma maneira extremamente útil e poderosa de distribuir programas desenvolvidos em Java aos utilizadores [Fla99]. No cliente, uma *applet* pode executar eficazmente, permitindo a implementação de funcionalidades de elevado grau de interactividade com o utilizador, sem que este se preocupe com questões de segurança, já que estas estão garantidas pelo próprio ambiente de execução, conhecido como *"sandbox"*. Uma das características mais interessantes na utilização das *applets* é a panóplia de recursos gráficos disponível a programadores para a implementação de interfaces ricos, muito interactivos e apelativos.

As *applets*, contudo, também têm os seus problemas, sendo o maior dos quais o tempo necessário para descarregar uma *applet* através da rede. Esta limitação levou por um lado à tendência de voltar as atenções novamente para o servidor [Pip99] e por outro ao desenvolvimento de *applets* menos potentes, mas mais pequenas e por isso mais fáceis de transferir.

3.3.10.2 ActiveX

A *Microsoft* desenvolveu uma arquitectura de componentes binários conhecida como COM (*Common Object Model*) que evoluiu mais tarde para DCOM (*Distributed* COM). O carácter binário dos componentes torna possível a sua execução em qualquer linguagem de programação desde que disponibilizem uma interface conforme à especificação da arquitectura. Assim, um componente COM funciona como uma "caixa preta", cuja funcionalidade é acessível através da sua interface pública.

A arquitectura COM permite o desenvolvimento de aplicações que recorram às fun-

cionalidades disponibilizadas por objectos COM. A arquitectura DCOM permite, por seu turno, desenvolver sistemas que utilizam componentes COM que podem residir em diferentes aplicações, máquinas ou noutras redes (incluindo a própria *internet*) [Kin97]. Apesar da *Microsoft* ter anunciado a disponibilização da arquitectura para outras plataformas, tal não foi efectuado [LB02], tornando a solução COM/DCOM altamente dependente da plataforma *Windows*.

Controlos ActiveX são componentes COM vocacionados para a interacção com o utilizador, ou seja, componentes gráficos. A sua relação com a *Web* reside na capacidade de embeber este tipo de componentes em páginas HTML para obtenção de pequenas aplicações (tais como as *applets* Java) que correm dentro do navegador. Esta possibilidade permite a criação de aplicações gráficas complexas com elevado grau de interactividade [Kin97].

Existem, contudo, dois problemas associados à utilização de controlos ActiveX:

1. A sua viabilidade está limitada à utilização do *Internet Explorer*, e por isso limitado a uma única plataforma cliente (*Windows*);
2. A sua utilização pode apresentar graves riscos de segurança, na medida em que a execução de um controlo ActiveX não ocorre dentro de um ambiente controlado (ao contrário das *applets* Java), podendo efectuar coisas tão diversas como executar programas, apagar ficheiros, ou comunicar com outros sistemas [Ber97].

3.3.10.3 Macromedia Flash

O *Flash* é uma linguagem desenvolvida pela *Macromedia* vocacionado para a construção de aplicações multimédia compactas e de alta qualidade.

O *Flash* apareceu essencialmente como uma ferramenta para desenhadores, que permitia o desenvolvimento de aplicações com recurso a um ambiente gráfico que encapsulava as especificidades da linguagem, não sendo por isso um ambiente totalmente programável.

Nas versões mais recentes, das quais a última é o *Flash MX*, a *Macromedia* foi introduzindo alterações no sentido de tornar o *Flash* numa tecnologia mais completa, permitindo a introdução de código nas animações (código *ActionScript*, uma linguagem baseada no *standard ECMAScript*), tanto para o processamento de eventos gráficos como para a implementação de outras funcionalidades. Nesta versão é agora possível a interacção com servidores aplicacionais J2EE ou .Net através de um mecanismo conhecido como *Flash Remoting*, baseado em serviços *Web* [All02]. Este tipo de interacção, obriga contudo, à existência de servidores proprietários da *Macromedia* para a tradução dos pedidos

A portabilidade é garantida pela existência de *plug-ins* para as plataformas mais comuns.

3.3.10.4 JavaScript

Outro exemplo de código no cliente é a utilização de *JavaScript* embebido em HTML. Esta linguagem foi desenvolvida pela *Netscape*, e permite dotar os documentos de funcionalidades mais interessantes do que o HTML simples (funções locais, despiste de erros no preenchimento de formulários, mensagens ao utilizador, etc), mas não tão avançadas como as disponibilizadas por outras linguagens. O suporte de *JavaScript* é generalizado à maioria dos navegadores existentes.

3.3.11 Plataformas Empresariais

A crescente evolução das tecnologias *Web* levou ao aparecimento de plataformas especialmente concebidas para a implementação de sistemas baseados em modelos de programação por camadas, estratificadas por áreas de funcionalidade, e que são normalmente associadas à apresentação (interacção com os utilizadores), à lógica de negócio e aos dados. Estes modelos serão analisados em detalhe na secção 3.4. A utilização destas plataformas, não é, contudo, exclusiva à utilização em ambientes puramente *Web*, podendo servir de base ao desenvolvimento de aplicações convencionais.

Na altura da escrita deste texto, destacam-se duas importantes plataformas empresariais: *O Java 2 Platform, Enterprise Edition* (J2EE) da *Sun Microsystems* e o .Net da *Microsoft*.

3.3.11.1 J2EE

O J2EE é um conjunto de especificações para tecnologias baseadas em Java, que permitem o desenvolvimento de aplicações que cumprem rigorosos requisitos em termos de disponibilidade, segurança, escalabilidade e manutenção [SM02a]. Destas tecnologias, fazem parte as *Java Server Pages* (JSP), as *Servlets*, os *Enterprise JavaBeans* (EJB), a *Java Connectivity Architecture* (JCA), o *Java Messaging Service* (JMS), as *Java Management Extensions* (JMX) e o *Java Naming Directory Interface* (JNDI). Outras tecnologias vulgarmente utilizadas, mesmo que pertencendo à plataforma Java convencional (referida como *Java 2 Platform, Standard Edition*, ou J2SE), são o mecanismo de ligação a bases de dados relacionais (*Java Database Connectivity* — JDBC) e a máquina virtual Java, que na sua versão *HotSpot* pode executar em modo servidor para beneficiar o desempenho de aplicações baseadas em J2EE [SM02a].

A figura 3.9 apresenta uma visão da pilha de tecnologias presente no J2EE. A disponibilização desta plataforma é assegurada por empresas que implementam produtos conforme às suas especificações, casos do *Websphere* da IBM e do *Weblogic* da *BEA Systems*, por exemplo.

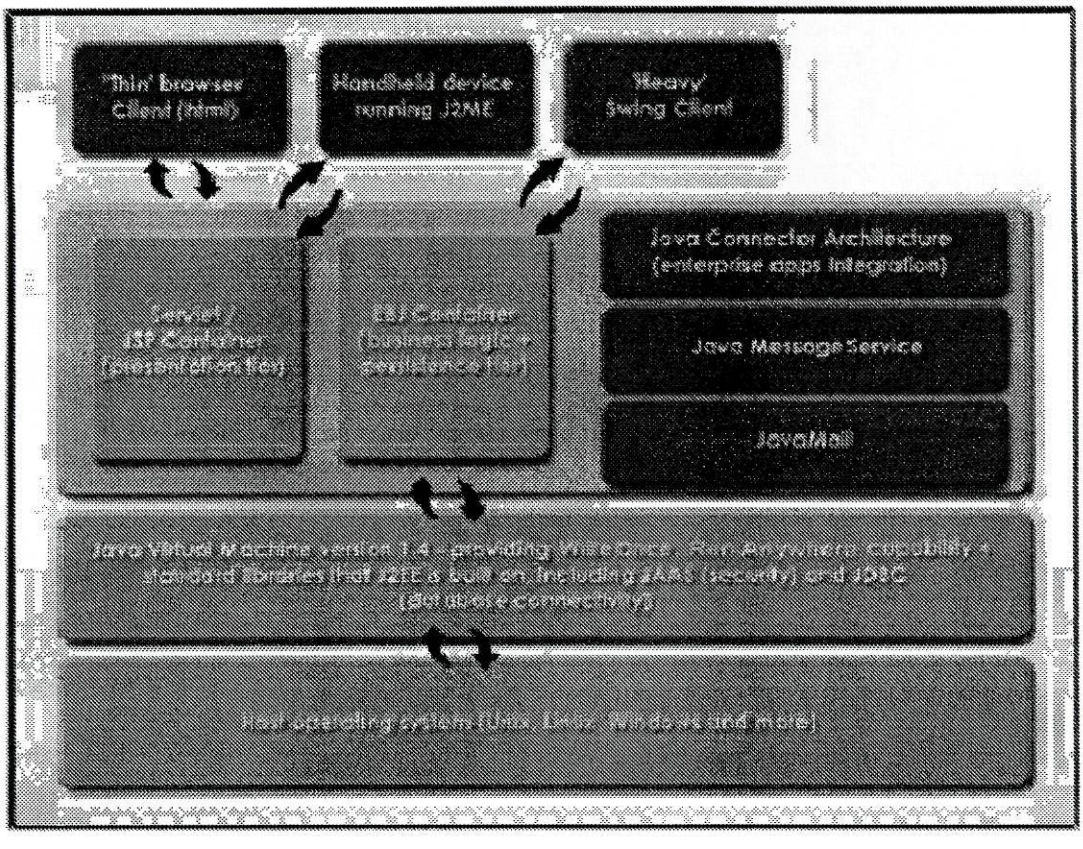


Figura 3.9: Pilha de tecnologias do J2EE

3.3.11.2 .Net

Ao contrário do J2EE, que se apresenta como uma especificação, o .Net é uma colecção de produtos, que integrados, totalizam a plataforma. Segundo a *Microsoft*, o .Net é um conjunto de tecnologias de *software* que permitem a interligação de informação, pessoas, sistemas e dispositivos [Mic03]

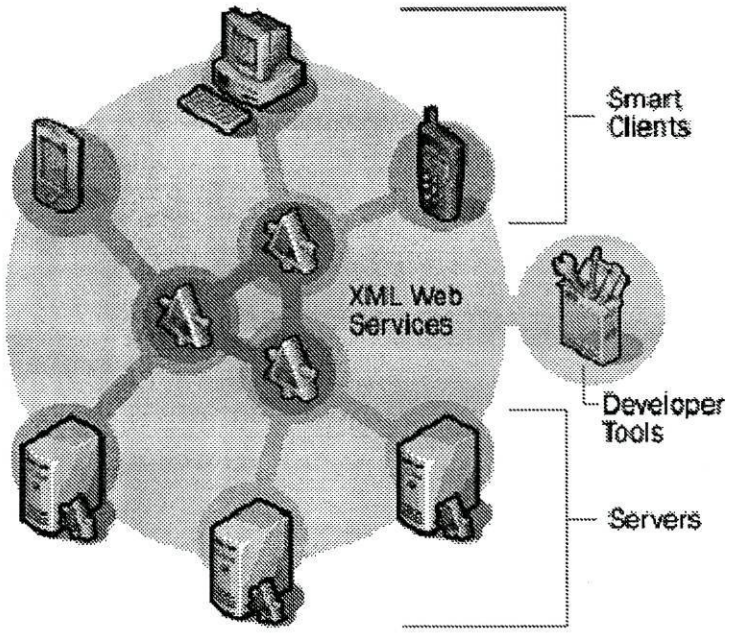


Figura 3.10: Plataforma .Net

A figura 3.10 ilustra o .Net como uma plataforma que permite interligar clientes (aqui denominados por "clientes inteligentes") com outros clientes, clientes com servidores ou mesmo servidores entre si, pela utilização de serviços *Web*. De facto, os serviços *Web* são os blocos fundamentais de estruturação de aplicações nesta plataforma.

Enquanto que o J2EE, por ser baseado em Java, garante uma utilização multi-plataforma das suas aplicações, o .Net contrapõe com a possibilidade de implementação de aplicações, baseadas em serviços *Web*, que podem ser escritas em várias linguagens, tais como o C, C++, C#, Fortran, COBOL, Visual Basic, etc.

3.4 Modelo de Programação Distribuído

Apesar de constituir um sistema universal, a *Web* apresenta especificidades que condicionam a forma de desenvolver aplicações, tanto pela sua natureza distribuída como pelo alcance global que permite. Após a análise das tecnologias mais comuns para o de-

senvolvimento de aplicações *Web*, é pois inevitável analisar os modelos de programação comummente utilizados.

A simplicidade dos servidores HTTP fez com que mesmo pessoas que nunca se sentiriam impelidas a desenvolver aplicações distribuídas, o fizessem, e o fizessem facilmente [MP02]. Tal como vimos no ponto 3.3, os programadores foram requerendo mais e mais funcionalidades dos seus servidores, e no processo, foram desenvolvendo modelos de programação cada vez mais complexos que lhes permitiram ir ultrapassando as dificuldades que enfrentavam. Segundo [MP02], a evolução das aplicações *Web* assenta em três modelos de programação distribuída:

- Modelo Cliente-Servidor;
- Modelo de 3-Camadas;
- Modelo de N-Camadas.

3.4.1 Modelo Cliente-Servidor

A *Web* nasceu como uma arquitectura cliente-servidor. O cliente, o navegador, efectua pedidos que são enviados e tratados pelo servidor (servidor HTTP), limitando-se depois a receber as respostas e a apresentá-las ao utilizador. Tudo neste tipo de aplicações, exceptuando a apresentação, é da responsabilidade do servidor[8]. Este modelo de programação é extremamente simples e descreve não só as aplicações *Web* originais, como também parte dos conteúdos disponibilizados hoje em dia.

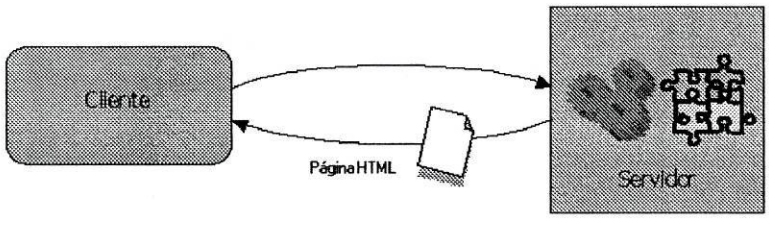


Figura 3.11: Modelo de programação cliente-servidor

O modelo cliente-servidor é um modelo que apresenta mais vantagens quando comparado com a utilização de aplicações únicas (*stand-alone*) [MP02], visto toda a lógica da aplicação se encontrar centralizada numa única máquina, e por isso mais fácil de gerir, mas mesmo assim disponível a um número vasto de utilizadores por via da capacidade de acesso universal da *Web*.

O problema do modelo reside no facto de não haver uma distinção clara entre a lógica de processamento dos pedidos, conhecida como lógica de negócio, e o próprio sistema

[8]Este conceito é verdade mesmo para aplicações que incluem código no cliente, já que estas tratam normalmente de aspectos relacionados com a apresentação.

de gestão da informação. Num cenário empresarial, onde os sistemas de informação servem vários tipos de aplicações, inclusivamente aplicações não-*Web*, a utilização de um sistema destes faz com que cada aplicação tenha que, redundantemente, re-implementar a lógica de negócio que define as regras de acesso aos mesmos dados, tornando a gestão global do sistema numa tarefa extremamente complexa e consequentemente mais sensível à ocorrência de erros.

O modelo cliente-servidor constitui um bom modelo para aplicações baseadas em páginas estáticas ou mesmo para aquelas que produzem informação dinâmica (no servidor), mas que não incorporam demasiado processamento para a sua execução [MP02].

3.4.2 Modelo de 3-Camadas

O modelo de 3-camadas surge como forma de ultrapassar as limitações do modelo cliente-servidor. Neste modelo, existem três zonas funcionais distintas, ou camadas: a camada de apresentação suportada pelo cliente, a camada de lógica de negócio que reside no servidor e a camada de dados que pode encontrar-se ou não no mesmo servidor.

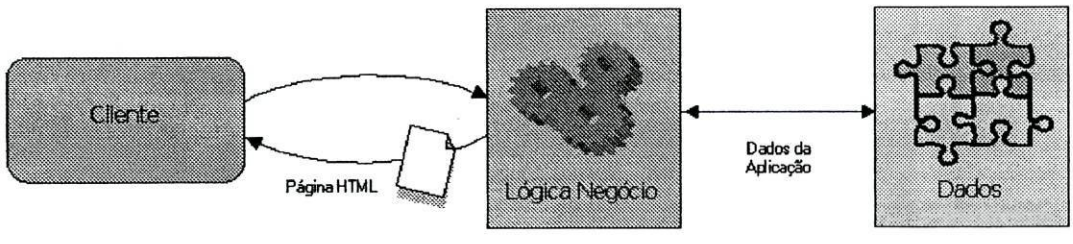


Figura 3.12: Modelo de programação de 3-camadas

A camada de apresentação tem exactamente as mesmas funções que no modelo anterior, isto é, enviar pedidos e apresentar as respostas ao utilizador. Já do lado do servidor teremos agora uma divisão clara entre lógica de negócio e dados. A função da camada intermédia (lógica de negócio) é a de proporcionar a construção dinâmica de informação que será enviada ao cliente sem preocupações de manutenção de integridade dos dados agora geridos independentemente. Esta separação permite, assim, que os mesmos dados possam ser acedidos facilmente por outras aplicações, que beneficiam das restrições de integridade que possam estar implementadas pela camada de dados, desde que esta disponibilize uma interface conveniente. O mesmo pode ser dito em relação à segunda camada, ou seja, é agora possível reaproveitar a lógica de negócio implementada desde que esta satisfaça os requisitos de outras aplicações que não precisam necessariamente de um interface *Web*.

Um exemplo deste tipo de reaproveitamento, pode ser a criação de uma aplicação de marcação de lugares em aviões que pode ser efectuada tanto na *Web* por um cliente, como no balcão da própria agência de viagens através de um formulário gráfico. Em ambos os casos, o modo de acesso à informação (camada de dados) e a lógica de

funcionamento da aplicação (lógica de negócio) é mesma, diferindo apenas na forma como é apresentada ao utilizador: num caso através de um navegador, noutro a partir de um formulário gráfico.

Sem bem que este modelo indicia uma clara separação entre apresentação e conteúdo, na realidade as coisas nem sempre são assim tão simples. O facto do HTML ser o resultado do processamento intermédio efectuado pela camada intermédia, faz com que a lógica tenha que ter conhecimento de conceitos relativos à camada de apresentação, já que o HTML é uma linguagem que também define a apresentação[9]. A solução passa por basear as respostas não na sua apresentação mas sim no seu conteúdo por utilização de uma linguagem como o XML. Desta forma, o mesmo tipo de informação poderia ser visualizada de forma mais apropriada ao cliente gráfico utilizado, por utilização de folhas de estilo diferentes [MP02].

3.4.3 Modelo de N-Camadas

À semelhança do que acontece com os modelos anteriores, também aqui a camada de apresentação permanece inalterada, no sentido em que constitui o ponto de contacto dos utilizadores com o sistema. A grande evolução é o facto do servidor poder agora ser um sistema extremamente complexo de interligações com outros servidores com vista à construção final do resultado.

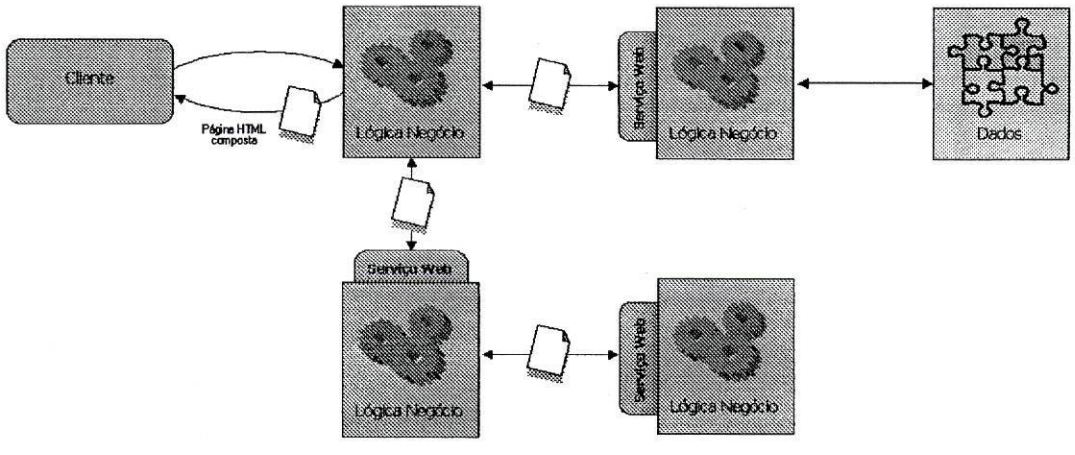


Figura 3.13: Modelo de programação de n-camadas

Neste modelo, o servidor é uma espécie de fachada que esconde o processamento do pedido. Por detrás dele, a lógica de processamento pode incluir a obtenção de informação noutros servidores com a sua própria lógica de negócio e dados. Em cada um destes servidores, o processamento poderá multiplicar-se novamente por vários outros pedidos que darão origem à resposta final, que será recolhida e integrada pelo servidor original.

[9]Tal como vimos, este facto deveu-se à proliferação de extensões "não autorizadas" à linguagem que acabaram por ser integradas por ela.

Um sistema assim construído só tem execução prática na *Web* se o processo de interconexão entre sistemas recorrer aos recursos por ela disponibilizados, ou seja, se o circuito de pedido-resposta fôr ele próprio um circuito *Web*. Como define [MP02], é preciso um mecanismo que suporte e encoraje a reutilização de aplicações *Web*, um mecanismo que seja conforme ao protocolo HTTP e à sua arquitectura universal, baseada em mensagens de texto.

A questão aqui, consiste em encontrar o modo como sistemas eventualmente baseados em tecnologias diferentes, possam ser capazes de trocar informação e cooperar entre si. O XML é sem dúvida o candidato natural para preencher esta lacuna, senão vejamos: É um *standard* aberto, baseado em texto e por isso independente da plataforma, e ainda permite a definição de gramáticas que possibilitam o estabelecimento de um formato para a construção e validação das mensagens. O sistema poderá finalmente estabelecer-se com o HTTP como meio de comunicação, o XML como linguagem e as ligações HTTP como meio de interconexão dos diversos serviços [MP02] — os serviços *Web*.

3.5 Aplicações SCADA na Web

A importância comercial da *Web* é uma notável força motriz que conduz empresas ao desenvolvimento de aplicações nela baseados. Como parte deste trabalho, foram analisados alguns produtos SCADA que apregoam funcionalidades *Web*, que agora se apresentam.

3.5.1 Rockwell Automation

A *Rockwell Automation* é uma das principais empresas que desenvolvem soluções de automação industrial com cerca de 23000 trabalhadores, com um volume de vendas anual de 4,3 mil milhões de dólares e sediada nos Estados Unidos[10].

Esta empresa disponibiliza soluções SCADA para a *Web* baseadas em dois produtos: *RSView32WebServer* e *RSView32 Active Display System*. Estes produtos não se encontravam disponíveis para análise.

3.5.1.1 RSView32WebServer

Este produto constitui um servidor Web que disponibiliza imagens estáticas (*snapshots*) das aplicações gráficas comuns aos sistemas SCADA (sinópticos, listas de entidades, listas de alarmes, etc) através de navegadores que suportem HTML 3.2, independentemente da sua plataforma. Esta solução é apresentada como ideal para clientes

[10]Dados retirados do sítio Web da empresa em: http://www.rockwellautomation.com.

"magros" [11], sem necessidade de instalação de *software* adicional.

3.5.1.2 RSView32 Active Display System

Esta é uma solução baseada na arquitectura DCOM da *Microsoft*, que disponibiliza interfaces de alto grau de interactividade e animados, graças à incorporação de controlos ActiveX nas páginas HTML. A viabilidade deste produto depende da utilização do navegador da *Microsoft* (*Internet Explorer*).

3.5.2 Axeda Systems Inc.

Empresa norte-americana que resultou da fusão entre a *eMation* e a *RAVISENT Technologies*, que desenvolve produtos de *software* para uma variedade de áreas empresariais: medicina, semi-condutores, escritório, indústria, etc[12]. O seu produto SCADA com suporte Web é o *Wizcon* que não se encontra disponível para análise (o sítio Web da empresa anunciava a disponibilização de uma demonstração *on-line* para breve).

3.5.2.1 Wizcon

O *Wizcon* é um SGC baseado na arquitectura DCOM da *Microsoft* que disponibiliza métodos de conversão das suas aplicações SCADA para HTML e para *applets* Java, prometendo capacidade de acesso ao sistema a partir de qualquer navegador que detenha suporte para Java. Este programa permite a criação de páginas HTML onde são introduzidas pequenas *applets* Java para a visualização de imagens, gráficos de tendência e listas de alarmes.

3.5.3 Serk Controls

A *Serk Controls* é uma empresa sediada no Reino Unido que desenvolve soluções de automação há mais de 40 anos. O seu produto SCADA para centros de comando é o SCX[13]. O sítio *Web* da empresa possibilita a visualização de uma demonstração *on-line*, que não se encontrava, contudo, disponível para análise na altura da execução do presente trabalho.

---

[11] *thin-clients*

[12] Dados retirados do sítio Web da empresa em: http://www.axeda.com.

[13] Dados retirados do sítio Web da empresa em: http://www.serck-controls.co.uk.

3.5.3.1 SCX

O SCX é um SGC SCADA baseado na plataforma *Windows* que disponibiliza aplicações de interface com o operador construídas com recurso a controlos ActiveX. O SCX disponibiliza acesso via *Web* através do módulo *WebView*. Este módulo possibilita o visionamento das aplicações pela utilização de navegadores que suportem Java ou ActiveX.

3.6 Sumário

A *World Wide Web* é um sistema de navegação por hipertexto desenvolvido por Tim Berners-Lee que visou unificar o processo de acesso à informação pela definição de uma arquitectura universal. A sua simplicidade, tornou-se na sua maior arma e permitiu a aceitação e envolvimento generalizados por parte da comunidade associada à rede global — a internet.

Estes dois factores (aceitação e envolvimento) conduziram ao aparecimento de várias tecnologias que permitiram a geração dinâmica de conteúdos, despoletando o processo de criação de aplicações baseadas na *Web*, revolucionando o conceito original da sua criação (visão estática da informação).

A natureza distribuída da *Web*, condicionou o modo de desenvolver aplicações que se traduziu na utilização de três modelos de programação distribuída: modelo cliente-servidor, modelo de 3-camadas e, mais recentemente, modelo de n-camadas.

A tomada de consciência da importância de *Web*, traduziu-de no desenvolvimento de aplicações SCADA nela baseados.

Capítulo 4

Análise do Problema e Definição da Tese

No capítulo 2 foram analisadas as aplicações mais comuns presentes em sistema SCADA, e descreveram-se as suas características, nomeadamente no que diz respeito ao interface com o utilizador. O capítulo 3, por seu turno, cobriu de forma alargada as tecnologias mais utilizadas para a implementação de aplicações na *Web*.

O objectivo deste capítulo é o de apresentar todos os requisitos recolhidos e, com base neles, discutir a viabilidade das abordagens, também aqui analisadas, e assim escolher qual a melhor opção para a implementação daquelas aplicações.

Uma vez escolhida, identificam-se as tarefas necessárias para a sua execução, que resultaram no desenvolvimento do trabalho que ora se apresenta.

4.1 Contextualização do Trabalho

A transposição de aplicações SCADA para um ambiente *Web* envolve a definição clara de quais os requisitos, quais as tecnologias que permitem a sua implementação e quais as características do ambiente onde aquelas serão utilizadas.

4.1.1 Revisão dos Requisitos

Os requisitos tidos em conta para a realização do presente trabalho, agrupam-se em três categorias:

Requisitos SCADA (RS) — decorrentes das características das aplicações SCADA apresentadas no capítulo 2;

Requisitos EFACEC (RE) — inferidos a partir da análise dos sistemas desenvolvidos pela EFACEC SE/AS e resultantes de consultas de opinião aos responsáveis dos sistemas desenvolvidos;

Requisitos Comerciais (RC) — visam o desenvolvimento de soluções com a obtenção de valor máximo, i.e. mínimo custo e máxima funcionalidade.

A apresentação dos requisitos permitirá a avaliação do sucesso ou não das opções tomadas durante a execução deste trabalho. Assim, os requisitos serão numerados para facilitar a sua referência. Cada requisito identificará a sua natureza (ou naturezas) pela utilização das siglas RS, RE e RC acima definidas.

1. Visualização de diagramas sinópticos, listas e gráficos de tendência (RS)
2. Actualização gráfica dos diagramas em função do valor de entidades (RS)
3. Suporte para animações e mecanismos de intermitência de imagens (piscar) (RS)
4. Definição de regras de coloração para diagramas e listas em função do valor de entidades (RS)
5. Suporte para ampliações e arrastamento de diagramas e disposição da informação em camadas (RS)
6. Definição de filtros para critérios de pesquisa em listas (RS)
7. Envio de comandos (RS)
8. Uniformização do interface de acesso a sistemas distintos (RE)
9. Possibilidade de interligação com sistemas heterogéneos implementados em plataformas distintas (RE,RC)
10. Permitir a introdução gradual de funcionalidades (escalabilidade) (RE,RC)
11. Possibilidade de comunicação a partir de pontos distintos (Rede local, linha dedicada, ligação *dial-up*, etc) (RE,RC)
12. Possibilidade de utilização do sistema através de dispositivos distintos[1], independentemente do sistema operativo que utilizem (RE,RC)

[1]Os requisitos gráficos das aplicações SCADA condicionam o tipo de dispositivo a utilizar, de modo a que a expressão ***dispositivos distintos*** inclui apenas dispositivos com capacidade gráfica e de processamento que possam suportar aqueles requisitos.

13. Minimização do esforço de configuração e instalação nos dispositivos de acesso (RC)
14. Evitar a utilização de tecnologias que impliquem o pagamento de licenças de exploração (*deploy*) (RC)
15. Evitar a utilização de tecnologias que apresentem questões de segurança para os utilizadores (RC)

4.1.2 Caracterização do Ambiente

Uma vez identificados os requisitos do trabalho, importa agora caracterizar os elementos que fazem parte do ambiente, que é naturalmente um ambiente *Web*. São três esses elementos: o cliente, o canal e o servidor.

4.1.2.1 Servidor

Num ambiente SCADA o servidor é, naturalmente, o próprio SGC que disponibiliza informação aos clientes e lhes permite actuar sobre o processo físico controlado. Pela análise dos SGC existentes na EFACEC SE/AS, poder-se-á inferir que o servidor é caracterizado por abranger uma vasta gama tecnológica que se traduz na sua dispersão por plataformas diversas e na existência de arquitecturas de *software* também distintas. Em termos de equipamentos, estes serão tão variados quanto o conjunto de plataformas existentes, que vão desde os computadores pessoais (PC), até aos servidores UNIX.

A disponibilização destes recursos através da *Web* implica a capacidade de integrar o próprio servidor HTTP com estes sistemas, por forma a permitir o fluxo de informação de e para os clientes.

4.1.2.2 Canal

O canal constitui o meio de ligação entre clientes e servidores, sendo por isso uma parte fundamental na definição do ambiente. O canal natural de comunicação de um SCADA é a própria rede local do sistema que interliga todos os equipamentos. Esta é normalmente uma rede *Ethernet* com débito de 10 ou 100 Mbit/s. Ainda no interior do sistema, poderão existir outros canais que permitam o acesso entre centros remotos e o centro de comando. Estes têm natureza diversa e podem variar desde ligações rádio, fibra-óptica ou mesmo redes alugadas. O seu aproveitamento para acesso *Web* entre estações remotas e centro de comando deverá sempre ser considerado. Outra possibilidade de acesso é através de ligações *dial-up* via *modem* directamente ao centro de condução. Estas são ligações de pouco débito (56 kbit/s) mas que são normalmente úteis para a realização de tarefas curtas e precisas. Finalmente, temos o canal de acesso global ao sistema, disponibilizado pela sua ligação à *internet*. Não é fácil precisar a

largura de banda disponibilizada por uma tal ligação. No entanto, e devido à recente explosão de técnicas de modulação de alto-débito como o ADSL (*Asymmetric Digital Subscriber Line*), poderemos assumir ligações a rondar os 512 kbit/s. Do ponto de vista do cliente, a sua largura de banda será limitada pelo seu próprio método de acesso à *Web*, variando desde os normais 56 kbit/s até ao máximo disponibilizado pelo centro de comando, i.e. 512 kbit/s.

4.1.2.3 Cliente

Em termos aplicacionais, o cliente do sistema é o navegador, uma vez que é ele quem permite o acesso dos utilizadores à *Web*. Quanto aos dispositivos, a sua utilização é condicionada pelos requisitos 1 a 7, que exigem equipamentos com capacidades gráficas e de processamento evoluídas, como o suporte de côr e animações. Deste modo, os equipamentos utilizados para o acesso *Web* a sistemas SCADA serão computadores pessoais (PCs), computadores portáteis e PDAs (*Personal Digital Assistant*) que disponham de ecrãs que permitam o visionamento destas aplicações.

4.2 Abordagens Analisadas

A implementação de soluções para a disponibilização de aplicações SCADA na *Web* pode ser realizada de diversas maneiras. O objectivo da presente dissertação consiste na análise do problema de apresentação destas aplicações do ponto de vista do cliente, normalmente designado na literatura por cliente-magro (*thin-client*) [LFCH02, LSFH00, SLN99, ZPMD97]. Seguindo a definição de [ZPMD97], considerar-se-á um cliente-magro como sendo uma aplicação leve, no sentido em que a sua lógica de processamento ocupa-se essencialmente do problema da apresentação de um interface gráfico ao utilizador (*Graphical User Interface — GUI*), com alguma lógica aplicacional adicional.

Para o desenvolvimento do presente trabalho, analisaram-se quatro abordagens de implementação de aplicações baseadas no navegador; duas baseadas na utilização de tecnologias directamente reconhecidas pelo navegador, e que por isso designamos por abordagens para clientes simples, e outras tantas baseadas na utilização de aplicações que estendem as capacidades naturais daquele, que designamos por abordagens para clientes complexos.

4.2.1 Clientes Simples

Nestas abordagens, a preocupação reside na utilização de tecnologias naturalmente suportadas pelos navegadores e que, por isso, não incorrem na necessidade de descarregamento de blocos de código pesados, tanto pelo seu tamanho, como pela capacidade

computacional que necessitam[2].

4.2.1.1 Páginas HTML

Nesta solução, utilizar-se-ia HTML para a definição do interface através da inclusão de imagens de diagramas geradas pela servidor e que iam sendo actualizadas ao longo do tempo, através do refrescamento periódico da página. A execução de controlos seria suportada pela inclusão de formulários (ou diálogos gerados por *JavaScript*) que resultavam na execução de um CGI ou de uma *servlet* no servidor para actuação no processo. Um mecanismo semelhante poderia ser utilizado com vista à inclusão de facilidades de navegação; pedidos de ampliações e arrastamentos que resultavam na geração de imagens ampliadas ou deslocadas, conforme requerido. As animações poderiam ser conseguidas pela utilização de GIFs ou PNGs animados. No servidor, haveria a necessidade de implementação de um sistema de geração de imagens com base na informação recolhida que ia servindo os pedidos recebidos.

O esforço de desenvolvimento desta solução centrar-se-ia no servidor, pela implementação de um motor para a geração dinâmica de páginas HTML. Este motor seria integrado com o SGC com vista à tradução dinâmica dos seus diagramas em imagens animadas (GIFs ou PNGs), e à geração também dinâmica de código *JavaScript* que possibilitasse a execução de controlos ou navegações.

Os principais problemas desta solução são:

- A geração dinâmica de imagens *rasterizadas* (GIFs e PNGs) com animações é pesada e pouco eficiente [Hun];
- Necessidade de carregamento total da página a cada pedido de refrescamento. Uma vez que o conteúdo de uma página poderia incluir uma ou mais imagens dinâmicas de grandes dimensões[3], então a quantidade de informação transaccionada entre clientes e servidor poderia ser muito grande e por isso condicionada pela largura de banda do canal de transmissão;
- Necessidade de uma taxa de refrescamento elevada. As características de tempo-real dos sistemas SCADA levam à necessidade de actualização frequente da informação visualizada. Este factor em conjunto com a eventual existência de outros clientes que efectuam acessos simultâneos ao sistema, introduz uma enorme sobrecarga no motor de geração de páginas que, por serem complexas e de grandes dimensões, pode conduzir ao atraso na disponibilização da resposta;

[2] Apesar dos navegadores não incluírem suporte nativo para processamento de SVG, considerou-se aceitável a utilização de um plug-in para o efeito.

[3] A supervisão de um processo complexo como os sistemas eléctricos de energia implica a existência de vários equipamentos a serem visualizados simultaneamente, levando à necessidade de uma imagem grande.

- Todas as acções do operador levam ao envio de um pedido ao servidor. O facto do cliente não conter qualquer capacidade computacional, faz com que não seja capaz de responder a solicitações do operador sem o envio de um pedido ao servidor, mesmo em questões puramente de interacção gráfica, como são os pedidos de ampliação e arrastamento;

4.2.1.2 Utilização de SVG

Esta solução é muito semelhante à anterior. A diferença reside na utilização de SVG para a representação de diagramas em vez das imagens animadas (rasters ou bitmaps). Devido à natureza vectorial do SVG, é possível obter imagens de muito boa qualidade que incluem animações e que são mais pequenas que imagens convencionais em formato GIF ou PNG. Para além disso, o SVG permite a implementação de pequenos blocos de código que possibilitam a interacção com a imagem representada [W3C01b]. Esta facilidade permite mesmo introduzir alterações ao diagrama localmente, sem necessidade do envio de um pedido ao servidor. O facto do SVG representar informação vectorial, possibilita a execução de ampliações e arrastamentos locais, sem que seja necessária a geração de uma nova imagem, diminuindo uma vez mais o fluxo de dados entre clientes e servidor [W3C01b].

Apesar de melhorar a abordagem anterior, uma solução baseada em SVG envolve ainda algumas questões:

- Não há suporte nativo nos navegadores para interpretação de SVG. A solução passa pela utilização de *plug-ins* especialmente concebidos para o efeito. A *Adobe* disponibiliza um *plug-in* para a visualização de documentos SVG, mas apenas para as plataformas *Windows* e *Macintosh*, não tendo sido encontrado nenhum outro que suporte outras plataformas (contraria o requisito 12);

- Continua a ser necessário o refrescamento periódico e total do documento para a actualização da informação gráfica, mesmo que não tenham ocorrido alterações ao sistema;

- O motor de geração dinâmica de páginas no servidor precisa de gerar documentos SVG correspondentes aos diagramas visualizados. Isto implica a utilização de ferramentas que facilitem esta geração, tais como as bibliotecas gráficas disponibilizadas pelo projecto *Batik*. Contudo, estas não dão ainda suporte para a utilização de animações e *scripting*. Existem outras ferramentas para a geração automática de SVG a partir de outros formatos, mas são na sua maioria de utilização comercial[4] (contraria o requisito 14);

[4]Informação obtida a partir do sítio *Web* do W3C referente às ferramentas para utilização de SVG em: http://www.w3.org/Graphics/SVG/SVGImplementations.htm8.

4.2.2 Clientes Complexos

Numa abordagem baseada em clientes complexos existe uma clara utilização da capacidade computacional do cliente, em oposição ao que acontece com as abordagens para clientes simples. Esta característica permite a implementação de funcionalidades complexas, mas apresenta o ónus de requerer o descarregamento de blocos substanciais de código para a sua execução, podendo, em alguns casos, limitar a utilização deste tipo de soluções.

4.2.2.1 Visualizador Remoto

Um visualizador remoto consiste numa aplicação que permite a visualização de interfaces gráficos que executam remotamente. Esta aplicação seria integrada no navegador (pela utilização de Java *applets* ou ActiveX) e comunicaria com um servidor residente no SGC através de um protocolo de transmissão de informação gráfica.

Esta é uma técnica conhecida como *Graphics Pipeline Interception* e o seu princípio de funcionamento consiste na existência de uma aplicação que intercepta os comandos gráficos ocorridos na máquina servidora, enviando a imagem resultante para visualização no cliente. Por seu turno, os eventos ocorridos no cliente são transmitidos ao servidor, que os executa localmente [LFCH02].

Este tipo de abordagem permite a visualização remota de todo o ambiente de trabalho (*desktop*), constituindo-se como uma solução interessante para qualquer tipo de aplicação [LFCH02], sendo disponibilizada por uma vasta gama de produtos comerciais: *Citrix MetaFrame* [Met], *Insignia NTrigue* [Ntr], *SCO Tarantella* [Tar], *Graphon RapidX* [Rap],*Symantec PC-Anywhere* [Any], e um dos mais recentes, de obtenção gratuita e multi-plataforma, o *AT&T VNC* [LSFH00].

O problema associado aos sistemas de visualização remotos é a largura de banda necessária para a transmissão de informação gráfica através da rede, mesmo que sejam utilizadas técnicas de compressão ou o envio de comandos gráficos de alto nível, em vez do envio total da imagem a visualizar [LFCH02]. Por outro lado, estes sistemas apresentam a importante vantagem de poderem ser integrados com sistemas existentes, através da intercepção das rotinas gráficas ao nível do sistema operativo.

Para o presente trabalho, foram considerados três factores que influenciaram decisivamente no sentido da exclusão desta abordagem:

1. A existência de produtos comerciais que implementam o mecanismo de visualização remota;
2. O interesse em uniformizar o interface de acesso a sistemas graficamente diferentes, como são o GENESys e o ScateX (requisito 8);
3. A largura de banda necessária para a transmissão de informação gráfica através

da rede;

Uma outra solução, semelhante à do visualizador remoto, é apresentada por [LFCH02] e consiste na utilização de *toolkits* de desenvolvimento gráfico que apresentam características distribuídas, para a implementação de aplicações. Este tipo de solução não foi considerada na altura da realização do presente trabalho, por este se encontrar já numa fase adiantada.

4.2.2.2 Componentes Especializados

A derradeira opção analisada, consiste na implementação de uma biblioteca de componentes gráficos Java, para a construção de clientes gráficos SCADA, que disponibilizam funcionalidades específicas deste tipo de aplicações. Estes componentes poderiam depois ser conjugados em *applets* que seriam descarregadas pelo navegador para execução local.

O desenvolvimento de componentes gráficos permite que grande parte das questões de interacção com o utilizador sejam resolvidas localmente sem necessidade de comunicação com as aplicações residentes no centro de comando, levando não só à diminuição do fluxo de informação entre clientes e servidores, como possibilitando o aumento do grau de interactividade entre operadores e aplicação.

Esta solução permite que a informação transaccionada entre clientes e servidores seja informação de alto nível, que corresponda à própria semântica do problema, em vez de informação de baixo nível, como a presente nos sistemas de visualização remota, onde circulam imagens e directivas de desenho. Assim, os valores correspondentes a alterações de entidades poderão ser enviados directamente a clientes especializados, com conhecimento suficiente para proceder à sua adequada representação; seja por alteração da coloração, por animação de imagens ou simplesmente por apresentação textual do seu valor.

A utilização do Java como tecnologia de base apresenta inúmeras vantagens:

- É multi-plataforma (requisito 12);
- É suportado pelos navegadores de forma nativa ou através de *plug-ins* (requisito 13);
- Permite o desenvolvimento de aplicações gráficas evoluídas através da utilização das suas bibliotecas gráficas (AWT e *Swing*) (requisitos 1, 2, 3, 4 e 5);
- É de obtenção e de utilização gratuitas (requisito 14);
- É orientada ao objecto;
- Permite a execução segura de aplicações no navegador devido ao modelo de segurança das *applets* (*sandbox*) (requisito 15).

Um dos problemas da utilização de componentes para a construção de interfaces SCADA na *Web* reside na necessidade destes componentes serem relativamente pequenos, para que o processo de descarregamento das aplicações possa ocorrer num intervalo de tempo aceitável.

Outro problema desta abordagem reside na necessidade de implementação específica de componentes para os vários aspectos dos interfaces SCADA: sinópticos, listas, gráficos de tendência e alarmes. Mas uma vez desenvolvidos, estes poderão ser utilizados para interligação com qualquer sistema, desde que seja estabelecido um mecanismo genérico de integração destes com os sistemas existentes, ou seja, uma arquitectura.

Assim, a implementação de um sistema baseado em componentes gráficos implica a definição e desenvolvimento de uma arquitectura de *software* que permita estabelecer a comunicação entre aqueles e as aplicações pertencentes a um SGC. Mais, uma tal arquitectura deve fazê-lo por utilização de um protocolo comum à *Web*, ou seja, o HTTP.

4.2.3 Escolha da Solução e Definição da Tese

A análise das opções apresentadas conduziu à escolha da solução baseada em componentes gráficos especializados, visto ser aquela que introduz um maior grau de inovação (necessidade de definição de uma arquitectura) e que se adequa de forma generalizada aos requisitos apresentados.

Será então possível enunciar a tese associada à realização da presente dissertação da seguinte forma:

A implementação de interfaces gráficos na Web para aplicações típicas de sistemas SCADA pode ser conseguida pela utilização de componentes Java especializados que executam localmente no navegador. A utilização destes componentes implica o desenvolvimento de uma arquitectura de software que suporte a sua integração com as aplicações residentes no centro de comando, nomeadamente pela implementação de um mecanismo uniforme de comunicação construído em cima do protocolo HTTP e de outros protocolos por elas utilizados.

A escolha de uma tal solução impossibilita, contudo, o desenvolvimento de todos os componentes referentes aos vários tipos de aplicações SCADA, no período disponível para a realização do presente trabalho. Em alternativa, decidiu-se pelo desenvolvimento de uma biblioteca de funcionalidades genéricas que permitisse o rápido desenvolvimento daqueles componentes, culminando com a implementação de um único componente especializado que permitirá validar a tese enunciada.

Desta forma, as tarefas necessárias à execução do trabalho podem ser resumidas a:

1. Especificar, implementar e testar uma arquitectura de *software* que permita o desenvolvimento de aplicações SCADA na *Web* baseada na utilização de com-

ponentes gráficos Java;

2. Desenhar, implementar e testar uma biblioteca de funcionalidades genéricas às aplicações SCADA que permita o rápido desenvolvimento de componentes gráficos especializados.

4.3 Arquitectura de Software

Uma vez decidido o tipo de abordagem a seguir para a disponibilização de interfaces gráficos SCADA na *Web*, importa agora especificar o modo como estas funcionalidades serão disponibilizadas, isto é, qual a arquitectura de *software* subjacente [MV01].

A definição de uma arquitectura de *software* está profundamente relacionada com o sistema onde deverá ser aplicada. A sua correcta definição permite facilitar o processo de gestão e desenvolvimento do sistema, sem pôr em causa as funcionalidades por ele disponibilizadas [MV01].

No caso presente, a arquitectura escolhida deverá conter as seguintes características:

- Ser distribuída (porque assim é a *Web*);
- Suportar ambientes heterogéneos. A arquitectura deve prever a existência de lógica aplicacional tanto em navegadores, como em servidores HTTP ou no próprio centro de comando onde residem SGCs de natureza heterogénea, tanto em termos de plataformas como em termos de tecnologias utilizadas;
- A arquitectura deve permitir a comunicação entre clientes e servidores *Web*, logo deve suportar comunicação via HTTP;
- A arquitectura deve permitir a comunicação entre clientes e aplicações residentes no SGC (via servidor *Web*) e, por isso, deve suportar outros protocolos de comunicação;
- A arquitectura deve esconder todas as especificidades próprias da comunicação (e.g. protocolos utilizados) proporcionando um interface de comunicação uniforme às aplicações;

A figura 4.1 apresenta um diagrama de blocos onde se apresenta a arquitectura como sendo a fundação para o desenvolvimento de aplicações em ambientes heterogéneos de forma uniforme, independentemente dos protocolos de comunicação de facto utilizados.

4.3.1 Arquitectura de Componentes

A maneira mais natural de especificar uma arquitectura que servisse os propósitos deste trabalho, foi analisar o trabalho desenvolvido na EFACEC SE/AS na busca de

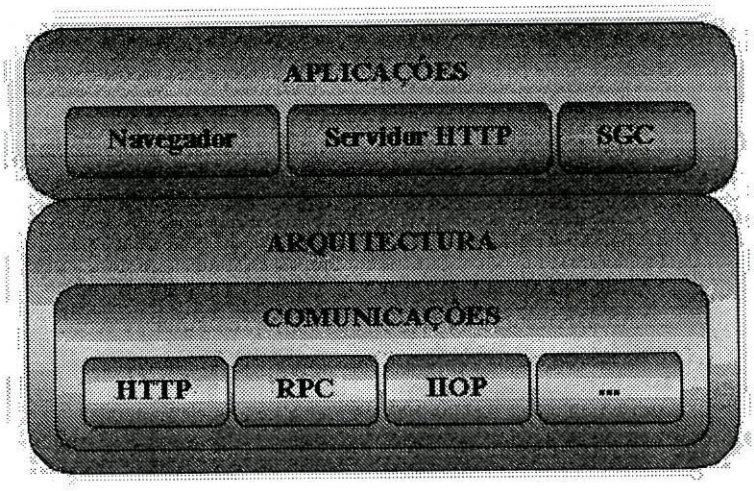


Figura 4.1: Características da arquitectura de software

uma arquitectura que pudesse ser utilizada como modelo para a arquitectura final. Essa arquitectura existe e é conhecida como BUS [MV01].

O BUS é uma arquitectura de componentes que colaboram entre si para a obtenção de uma funcionalidade específica. A colaboração entre componentes é baseada no *modelo push* [MV01].

Os pontos seguintes descrevem as características do BUS que foram tidas em consideração para a definição e implementação da arquitectura que serviu de base à realização deste trabalho, cujo projecto será apresentado mais adiante.

4.3.1.1 Modelo Push

O Modelo *Push* consiste na troca de eventos que se encontram associados em tópicos. Assim, um componente regista-se no BUS como subscritor e/ou publicador de tópicos; subscrever um tópico significa dizer que o componente está disposto a receber eventos desse tópico; de forma análoga, publicar num tópico significa que o componente irá enviar eventos desse tópico. A associação entre eventos e tópicos é mantida no *meta-modelo de eventos* [MV01].

4.3.1.2 Pedidos e Respostas de Dados

Sobre o modelo *push* existe o protocolo de *Data Request*, que permite que um componente efectue pedidos específicos de dados (eventos) que lhe serão enviados como resposta, caso haja no sistema alguém com capacidade para o fazer [MV01].

4.3.1.3 Componentes

Do ponto de vista do BUS, um sistema é constituído por componentes que trocam eventos entre si. Cabe ao BUS garantir que um determinado evento publicado é entregue a todos os componentes que subscreveram o tópico respectivo.

Do ponto de vista dos componentes, o seu mundo começa e acaba no BUS; dele recebe eventos e nele publica eventos. Um componente pode ser assim visto como uma máquina de estados que executa uma determinada funcionalidade em função dos eventos que recebe. Deste modo, programadores podem concordar na definição de contratos que são estabelecidos através da utilização da interface de eventos dos seus componentes [MV01].

4.3.1.4 Eventos

Os eventos do BUS seguem um formato de texto para a estruturação da informação conhecido como *stringified*. Sendo baseados em texto os eventos podem facilmente ser entendidos por humanos [MV01], e permitem a troca de informação com componentes desenvolvidos em plataformas distintas[5].

4.3.2 Nota de Implementação

O esforço de especificação e desenvolvimento da arquitectura que serviu de base à execução deste trabalho foi realizado em colaboração com o Miguel Ferreira Pereira Gomes, no âmbito do seu trabalho de mestrado intitulado *Canal SCADA na Web* [Gom02], também realizado na EFACEC SE/AS, e contemporâneo ao trabalho aqui apresentado.

O projecto descrito no capítulo 5 restringe-se à definição geral da camada da arquitectura responsável pelo modelo de componentes, sem a análise das especificidades da sua implementação sobre diversos protocolos. Para os testes efectuados foi utilizada uma implementação da arquitectura que utiliza RMI (*Remote Method Invocation*) como forma de comunicação entre componentes.

O autor considerou aceitável limitar a sua participação à definição e implementação da camada superior da arquitectura (componentes), por não achar importante nem conveniente[6] a sua intervenção no desenvolvimento das camadas inferiores, no âmbito do presente trabalho.

---

[5] Tal como em XML.

[6] A participação nestas actividades poderiam condicionar o tempo disponível para o desenvolvimento da biblioteca de funcionalidades SCADA.

4.4 Biblioteca de Funcionalidades SCADA

O sucesso da definição da arquitectura permitiu isolar o processo de desenvolvimento de aplicações na *Web* do processo de criação de uma biblioteca de funcionalidades SCADA. O objectivo desta biblioteca é dotar os programadores de um conjunto de ferramentas, gráficas e não gráficas, especificamente construídas para a resolução dos problemas comuns à criação destas aplicações.

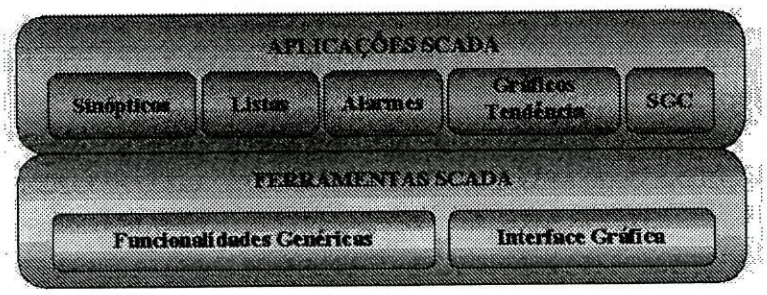


Figura 4.2: Biblioteca de funcionalidades SCADA

A figura 4.2 apresenta um diagrama de blocos onde se ilustra o propósito de desenvolver funcionalidades importantes para a generalidade das aplicações SCADA, desde as interfaces até mesmo às aplicações do próprio SGC.

4.4.1 Funcionalidades Genéricas

O desenvolvimento de funcionalidades genéricas tem como objectivo a implementação de um conjunto de ferramentas que possam ser usadas para a manipulação genérica da informação recolhida pelas aplicações. Não é seu intuito proporcionar qualquer tipo de facilidade gráfica.

As ferramentas incluídas neste pacote (que serão detalhadas mais adiante) permitem:

- gestão de atributos genéricos;
- criação e avaliação expressões definidas segundo uma gramática XML;
- criação, gestão e avaliação de regras (também definidas em XML) contra gestores de atributos (motor de regras).

4.4.2 Funcionalidades Gráficas

O pacote de funcionalidades gráficas visa a definição de uma estrutura de implementação de componentes gráficos baseados em componentes Java. Este pacote apre-

senta a implementação de um componente genérico com capacidade de visualização de figuras. O objectivo deste componente é de servir de modelo para a implementação de outros componentes que implementem funcionalidades específicas às interfaces SCADA, e não o de implementar todas elas. Um exemplo de extensão deste componente é também apresentado e consiste num componente especializado para a visualização de diagramas sinópticos.

Para além desta definição, o pacote inclui ainda:

- definição de uma caixa de ferramentas para captura de eventos gráficos e sua integração com outros componentes;
- implementação de ferramentas úteis (escalamento, arrastamento, selecção);
- suporte para animações;
- implementação de contentores de figuras gráficas organizadas em camadas com noção de níveis de visibilidade;
- estruturas para gestão espacial da informação;
- motor de simbologia baseado na definição de símbolos, estilos e diagramas descritos em XML e geridos por regras dinâmicas.

4.5 Sumário

A implementação de aplicações SCADA na *Web* pode ser conseguida de várias maneiras. Neste capítulo apresentaram-se quatro abordagens possíveis, duas baseadas na utilização de tecnologias naturais aos navegadores e outras duas baseadas na implementação de aplicações que permitem a extensão das suas funcionalidades.

A análise dessas soluções, tendo como base os requisitos previamente definidos, conduziu às escolha de uma solução baseada em componentes Java especializados que podem ser integrados em *applets* que são descarregadas pelo navegador para execução local.

A escolha desta opção permitiu enquadrar o trabalho como sendo a implementação de uma biblioteca de funcionalidades gráficas e não gráficas para o desenvolvimento de aplicações SCADA na *Web*, com o objectivo de facilitar o desenvolvimento dos componentes gráficos especializados.

A utilização de pequenas aplicações (*applets*) leva à necessidade de definir e implementar uma arquitectura de *software* que suporte o seu desenvolvimento e permita a sua integração com as aplicações do SGC. A definição de uma tal arquitectura deve ter em conta as características do problema em questão.

O BUS, uma arquitectura de componentes colaborantes, desenvolvido na EFACEC SE/AS apresenta características que se adequam ao problema actual, pelo que constituiu o modelo para a definição final da arquitectura utilizada neste trabalho.

Capítulo 5

Framework de Componentes

Este capítulo apresenta o projecto de *software* ligado ao desenvolvimento da arquitectura de componentes em Java, segundo os princípios descritos no capítulo 4.

Cada uma das funcionalidades descritas segue uma estrutura mais ou menos fixa que tem a seguinte forma:

- Contexto — descreve o âmbito de utilização, ou os motivos que levaram à necessidade de implementação de cada funcionalidade;
- Descrição do Problema — descreve de forma sucinta os objectivos pretendidos com cada funcionalidade, ou seja, o seu contrato;
- Descrição Funcional — descreve o modo como cada funcionalidade irá cumprir o seu contrato. Aqui utiliza-se a notação UML (*Unified Modelling Language*) para a apresentação do diagrama de classes respeitante a cada funcionalidade.

Durante a descrição do projecto serão feitas referências aos padrões de desenho de *software* utilizados, que serão identificados genericamente pelo termo *padrão*.

5.1 Arquitectura ou Framework?

Os termos *arquitectura de software* e *framework* são muitas vezes utilizados de forma indiferenciada na indústria do *software* [Nie01]. Tendo em vista a clarificação do seu significado, apresentar-se-ão as definições tomadas em consideração durante a realização deste trabalho.

5.1.1 Framework

Segundo [JF88], um *framework* pode ser entendido como um desenho reutilizável, expresso através de um conjunto de classes abstractas e pela definição da forma de colaboração entre as suas instâncias. Este desenho é reutilizável para a totalidade ou parte de um sistema de *software*.

Neste contexto, um *framework* afigura-se como uma estrutura bem definida, que estabelece os princípios de funcionamento de um sistema. Embora específico a um determinado domínio aplicacional, um *framework* pode ser reutilizado para o desenvolvimento de sistemas distintos.

5.1.2 Arquitectura de Software

O conceito de arquitectura é originário do domínio da criação de estruturas de edifícios. Actualmente, o termo é também utilizado para referir a criação de estruturas de qualquer tipo de sistemas, incluindo os sistemas de *software* [Nie01].

Segundo [SG99], uma arquitectura de *software* define um sistema em termos de componentes computacionais e as interacções entre eles. Neste contexto, um componente é algo que possa ser utilizado num sistema de *software*: bases de dados, aplicações, *middlewares*, servidores aplicacionais, etc.

5.1.3 Distinção

Embora semelhantes, os conceitos de arquitectura e *framework* apresentam significados diferentes. Adoptando a visão de [MF00], uma arquitectura de *software* refere-se aos modelos relativos às fases de análise e de desenho de alto-nível de um sistema. Um arquitectura de *software* é, assim, abstracta e independente das linguagens de programação utilizadas. Já um *framework* pode ser visto como uma implementação de uma arquitectura de *software*, que proporciona um conjunto de classes especificamente desenvolvidas com recurso a uma determinada linguagem de programação [MF00]. A utilização de um *framework* permite aos programadores especializar essas classes abstractas como parte do processo de desenvolvimento de um determinado sistema de *software*.

Com base nesta distinção, considerar-se-á o *framework* de componentes agora apresentado, como a implementação Java da arquitectura de componentes definida no capítulo 4.

De notar que o conceito de arquitectura agora definido, não contradiz aquele apresentado no capítulo 4, constituindo apenas um aprofundamento do mesmo.

5.2 Estrutura dos Pacotes de Software

O projecto estrutura-se em pacotes Java (`frk` e `cmp`) que encapsulam as funcionalidades referentes ao *framework* de componentes. O pacote `efa`, por seu turno, disponibiliza funcionalidades genéricas enquadradas no âmbito da biblioteca de funcionalidades SCADA sendo, por isso, descrita apenas no capítulo 6. A figura 5.1 apresenta um diagrama que ilustra a relação entre estes pacotes.

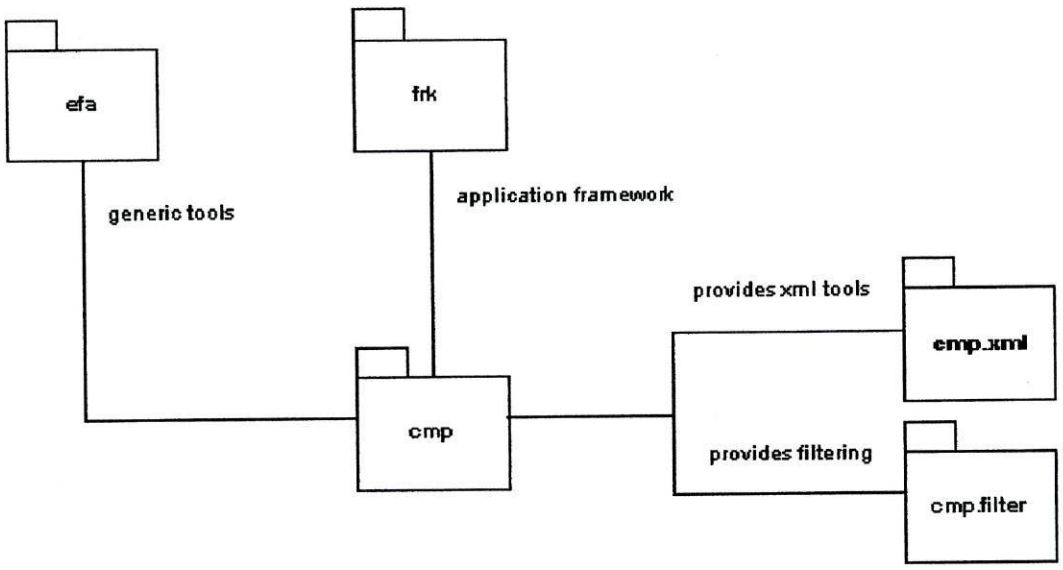


Figura 5.1: Estrutura de pacotes do framework de componentes

5.3 Framework para Aplicações

O *framework* para aplicações[1] é o pacote mais genérico de toda a estrutura apresentada e é identificado pela sigla `frk`.

5.3.1 Contexto

Uma das tarefas primordiais de uma arquitectura de *software* é a de providenciar mecanismos que acelerem o processo de desenvolvimento pela disponibilização das funcionalidades previsivelmente mais utilizadas. Dessas, as tarefas associadas com a construção de aplicações e sua interacção com o ambiente são fundamentais.

[1] O pacote `frk` modeliza um *framework* para aplicações e faz parte do modelo mais geral referente ao *framework* de componentes apresentado neste capítulo.

5.3.2 Descrição do Problema

O *framework* de aplicações, ou `frk`, deve proporcionar uma estrutura que facilite o desenvolvimento de aplicações genericamente. Uma aplicação pode ser entendida como uma imagem em memória que existe por si só (aplicações *stand-alone*) ou que existe integrada num contentor especializado (*applets* por exemplo). Uma aplicação deve ser identificada por uma *string* e deve proporcionar ciclos de arranque e paragem muito bem definidos. Uma aplicação deve fornecer mecanismos genéricos de leitura de parâmetros do ambiente e mecanismos de *logging*. Uma aplicação pode ser gráfica ou não-gráfica (geralmente designada por aplicação consola). Em ambos os casos, a aplicação deve disponibilizar um meio de serialização do seu processamento interno (ex. eventos gráficos) com eventuais eventos externos (ex. comunicação inter-processos) por forma a garantir um modelo de programação sequencial, sem a necessidade de utilização de mecanismos de exclusão-mútua.

5.3.3 Descrição Funcional

A figura 5.2 apresenta o diagrama com as classes principais do pacote `frk`. O interface `Application` define a estrutura que deve ser seguida por todas as classes aplicação, nomeadamente que todas as aplicações devem ter um nome e que devem fornecer um objecto de uma classe que implemente os métodos definidos pelo interface `Queue`. Este interface define a estrutura genérica de uma fila de eventos à qual são adicionadas *strings* que são depois entregues a instâncias de `QueueClient` (outro interface) para processamento. A classe `ApplicationQueue` é uma implementação do interface `Queue` que deriva de `java.awt.EventQueue` e que, por isso, beneficia das suas características, que são as de tratamento sequencial dos eventos pela ordem em que são introduzidos na fila [Sun, java.awt.EventQueue]. Os eventos no formato de *strings* são encapsulados por instâncias da classe `AwtStringWrapper` que, sendo uma extensão da classe `java.awt.AWTEvent`, pode ser inserida em `java.awt.EventQueue`.

A classe `AbstractApplication` é uma implementação abstracta do interface `Application` que implementa o padrão *Singleton* [EGV97, p 127] e que define a estrutura de arranque e paragem das aplicações. Uma aplicação é iniciada por uma chamada ao método `run()`, cuja execução culminará na execução do método *Templated* [EGV97, p 325] `startup()` que deve ser implementado pelas classes derivadas.

De forma análoga, uma invocação do método `exit()` dá origem ao processo de paragem da aplicação que culmina com uma invocação do método *Templated* `shutdown()` implementado pelas classes derivadas.

As classes `AbstractConsoleApplication` e `AbstractGraphicApplication` são extensões, também abstractas, da classe *AbstractApplication* que devem servir de base para o desenvolvimento de aplicações consola ou aplicações gráficas, respectivamente. A classe `AbstractGraphicApplication` substitui a fila de processamento de eventos gráficos do AWT (*Abstract Windowing Toolkit*) pela sua própria fila de eventos, de

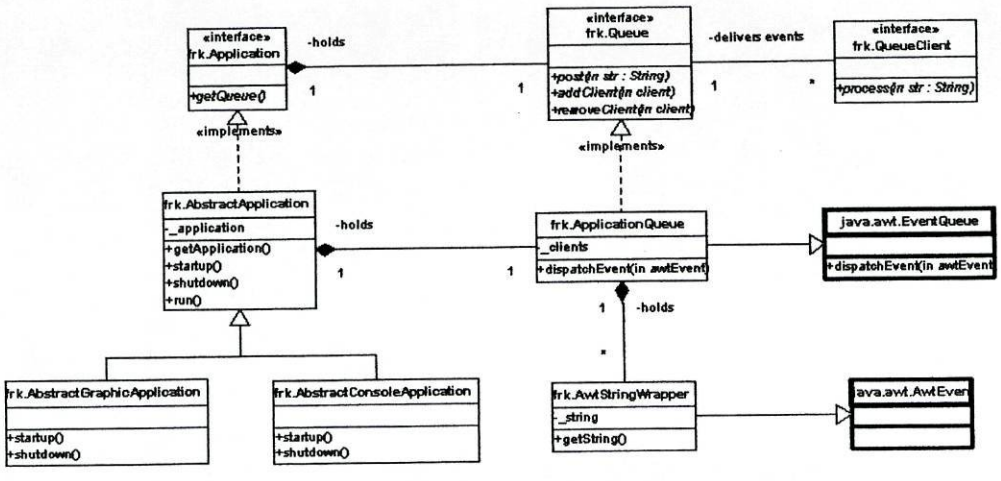


Figura 5.2: Diagrama de classes do pacote frk

modo a garantir a serialização destes eventos com outros quaisquer eventos baseados na classe `AwtStringWrapper`.

A classe Log disponibiliza as facilidades de *logging* (para consola) que pode ser estruturada em 5 níveis de *debug*. A utilização do nível *debug* está directamente relacionada com o grau de importância da informação a ser escrita; quanto mais importante, menor deverá ser o nível utilizado. É possível a identificação do nível de *debug* a utilizar por uma aplicação através de uma variável de ambiente.

A leitura de variáveis de ambiente é assegurada pela classe `Utils` que disponibiliza um método estático (`getParameter()`) para o efeito. A obtenção de variáveis de ambiente é efectuada por classes especializadas que implementam o interface `ParameterReader`[2].

5.4 Modelo de Componentes

O modelo de componentes encontra-se definido no pacote `cmp` e baseia-se em algumas funcionalidades disponibilizadas pelos pacotes `frk` e `efa`, este último será apresentado mais adiante.

5.4.1 Contexto

A discussão realizada no capítulo 4 permitiu concluir ser necessária a implementação de uma arquitectura que possibilite o desenvolvimento de sistemas baseados em componentes que colaboram entre si pela troca de mensagens estruturadas. Nesta secção

[2] O processo de leitura de variáveis de ambiente é diferente em *applets* e aplicações *stand-alone*. As *applets* obtêm esses valores a partir dos elementos presentes na própria página HTML.

procura-se a concretização de uma tal arquitectura segundo os princípios então discutidos.

5.4.2 Descrição do Problema

O objectivo deste pacote é a disponibilização de uma arquitectura de componentes que comunicam entre si pela troca de eventos segundo um paradigma de publicação/subscrição de tópicos. Um componente subscreve um tópico para receber eventos desse tópico e publica num tópico para enviar eventos desse tópico. Um componente pode requerer determinada informação pelo envio explícito de um pedido de dados. Um pedido de dados deve conter um evento com a informação do pedido, um identificador do serviço (texto) e um outro identificador que deve ser devolvido juntamente com os eventos de resposta para identificação do pedido que os originou. Os componentes têm um nome e apresentam um ciclo de vida bem definido, garantido por um mecanismo de arranque e paragem. Os componentes funcionam como máquinas de estado que operam pela recepção assíncrona de eventos e pedidos de dados com vista à implementação de uma funcionalidade específica. Um componente regista *Callbacks* [SM02b, p 238] para a recepção específica de eventos. A relação entre tópicos e eventos é estabelecida por um metamodelo de eventos.

5.4.3 Eventos, Pedidos de Dados e XML

Os eventos e pedidos de dados que circulam entre componentes têm formato de texto e são definidos em XML. A opção pelo XML baseou-se no facto de ser um formato *standard*, por permitir a definição de uma gramática própria à natureza do problema e por haver um conjunto de ferramentas disponíveis para o seu processamento. A semântica associada à compreensão dos eventos é entregue às classes que lidam directamente com eles, na sua maioria, os componentes.

A estrutura de um evento pode ser descrita através do DTD (*Document Type Definition*) que se apresenta na figura 5.3.

```
<!ELEMENT e (a*)>
<!ATTLIST e
    n CDATA #REQUIRED
    t CDATA #REQUIRED>
<!ELEMENT a (#PCDATA)>
<!ATTLIST a
    n CDATA #REQUIRED>
```

Figura 5.3: Estrutura de um evento (event.dtd)

No modelo da figura 5.3, um evento (elemento <e>) apresenta dois atributos obrigatórios, o nome (atributo n) e o tópico (atributo t) e é constituído por um número indeterminado de elementos do tipo chave-valor (elemento <a>). A chave destes elementos é o seu nome (atributo n), sendo o valor representado pelos dados delimitados pelo identificador do elemento (<a> e).

De forma análoga, apresenta-se na figura 5.4 o DTD que define a estrutura de um pedido de dados. Um pedido de dados (elemento <dr>) é identificado por um serviço (atributo urn), por um identificador de resposta (atributo rn) e contém um só evento.

```
<!ENTITY % event SYSTEM "event.dtd">
%event;
<!ELEMENT dr (e)>
<!ATTLIST dr
    urn CDATA #REQUIRED
    rn CDATA #REQUIRED>
```

Figura 5.4: Estrutura de um pedido de dados (data_request.dtd)

Eventos e pedidos de dados são encapsulados num elemento específico para transmissão (elemento <es>). A figura 5.5 ilustra o DTD que define esta estrutura.

```
<!ENTITY % data_request SYSTEM "data_request.dtd">
%data_request;
<!ELEMENT es (e*,dr*)>
```

Figura 5.5: Estrutura de eventos e pedidos de dados para transmissão (events.dtd)

Na figura 5.6 apresenta-se um documento XML conforme aos DTD descritos que contém um evento e um pedido de dados.

Os DTD acima identificados permitem estabelecer a estrutura dos eventos e pedidos de dados que circulam entre componentes, independentemente da informação que contêm. Para que dois componentes possam trocar informação entre si, ambos devem conhecer o tipo de informação que pretendem, isto é, devem conhecer qual o evento (ou pedido de dados), qual o tópico desse evento e qual a natureza dos seus atributos. Essa informação é disponibilizada pelo metamodelo de eventos, ele próprio um documento XML definido numa gramática específica.

A figura 5.7 ilustra o DTD do metamodelo de eventos, onde se define o elemento <metamodel> como sendo constituído por um número arbitrário de tópicos (elemento <topic>. Um tópico é por sua vez composto por eventos (elemento <event>) que são

```
<?xml version="1.0" encoding "UTF-8"?>
<es>
    <e n="Analog">
        <a n="Identifier">Termo15</a>
        <a n="Value">10.5</a>
    </e>
    <dr urn="GetAnalogs" rn="request23">
        <e n="Analog">
            <a n="Value">10</a>
        </e>
    </dr>
</es>
```

Figura 5.6: Exemplo de um documento XML com eventos e pedidos de dados (events.xml)

```
<!ELEMENT metamodel (topic*)>
<!ATTLIST e
    n CDATA #REQUIRED
    t CDATA #REQUIRED>
<!ELEMENT topic (event*)>
<!ATTLIST topic
    name CDATA #REQUIRED>
<!ELEMENT event (attribute*)>
<!ATTLIST event
    name CDATA #REQUIRED>
<!ELEMENT attribute EMPTY>
<!ATTLIST attribute
    name CDATA #REQUIRED>
    type CDATA #REQUIRED>
```

Figura 5.7: Estrutura do metamodelo de eventos (metamodel.dtd)

constituídos por atributos (elemento <attribute>). Todos os elementos são identificados por um nome à excepção do próprio metamodelo. O elemento <attribute> define ainda um atributo type que utilizado para identificar o seu tipo de dados. Os tipos de dados possíveis são os que se encontram definidos no ponto 6.2. Na figura 5.8 apresenta-se o exemplo de um metamodelo conforme ao DTD referido.

```
<?xml version="1.0" encoding "UTF-8"?>
<metamodel>
    <topic name="Entities">
        <event name="Analog">
            <attribute name="Identifier" type="EFAStringValue"/>
            <attribute name="Value" type="EFADoubleValue"/>
        </event>
        <event name="Digital">
            <attribute name="Identifier" type="EFAStringValue"/>
            <attribute name="IsOn" type="EFABoolValue"/>
        </event>
    </topic>
    <topic name="Alarm">
        <event name="EntityAlarm">
            <attribute name="Identifier" type="EFAStringValue"/>
            <attribute name="InAlarm" type="EFABoolValue"/>
        </event>
    </topic>
</metamodel>
```

Figura 5.8: Exemplo de um documento XML contendo um metamodelo de eventos (metamodel.xml)

5.4.4 Descrição Funcional

A estruturação apresentada na secção 5.4.3 formaliza o mecanismo utilizado para a codificação das mensagens trocadas entre componentes. Uma das tarefas do pacote cmp é promover o isolamento entre a lógica de codificação de eventos e pedidos de dados e os componentes que os utilizam, pela disponibilização de uma API de alto nível. A figura 5.9 ilustra as classes principais que garantem um tal isolamento.

A classe Event implementa o interface efa.Lamable (discutido mais adiante) que define um modelo para classes que gerem um conjunto de atributos do tipo chave-valor. Um evento sabe converter-se para a sua representação em XML e fornece uma API para o preenchimento e/ou leitura dos seus atributos. Cada instância de Event está associada a uma instância de EventGam que é responsável por guardar as definições

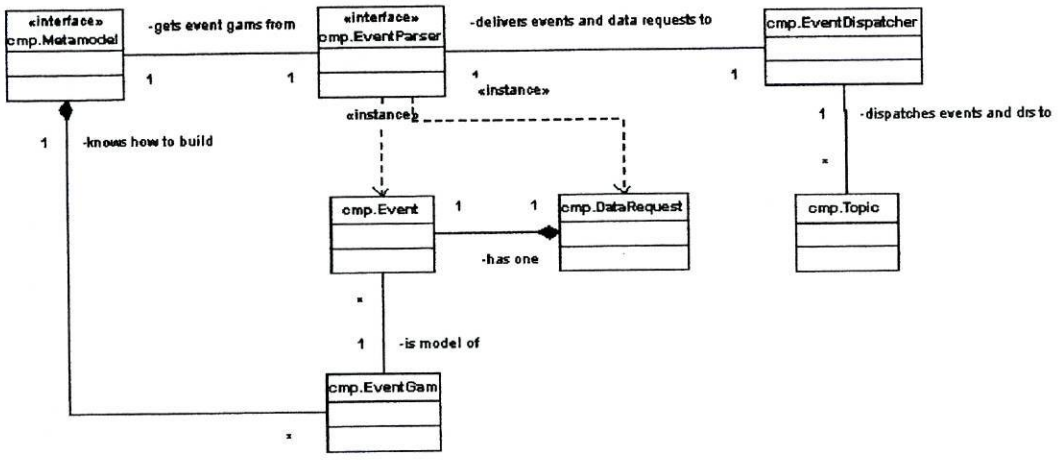


Figura 5.9: Diagrama de classes afectas ao processamento de eventos

dos tipos de atributos do evento respectivo. Esta classe é uma especialização da classe `efa.Gam` que também será analisada mais tarde. As instância de `EventGam` são obtidas a partir da classe que implementa o interface `Metamodel`. Este interface prevê ainda a existência de um método para a obtenção do tópico associado a um determinado evento.

A implementação concreta do metamodelo de eventos é fornecida pelas classes `DOMMetamodelParser` e `DOMMetamodelParserEngine` pertencentes ao pacote `cmp.xml`. Estas classes utilizam a API DOM (analisada no capítulo 3) para a interpretação do ficheiro XML referente ao metamodelo de eventos.

A classe `DataRequest` é a representação, no modelo, que mapeia o elemento XML `<dr>`. Em termos funcionais, um `DataRequest` é constituído por um evento (`Event`) e permite a escrita/leitura dos atributos de serviço e de identificação do pedido, designados respectivamente por *Universal Request Name*[3] e *Request Name*. Um `DataRequest` sabe converter-se para a sua representação em XML.

O interface `EventParser` constitui um dos elementos principais da arquitectura de eventos na medida em que define a API que deve ser implementada pelas classes capazes de converter eventos e pedidos de dados XML em eventos e pedidos de dados Java. Nesta API são definidos dois métodos. O método `parse()`, que aceita uma *string* contendo informação XML, conforme ao DTD apresentado na figura 5.5, para descodificação. O processo de descodificação deve culminar com a entrega dos respectivos eventos e pedidos de dados Java para a instância de `EventDispatcher` fornecida através do método `setEventDispatcher()`. Esta classe, por sua vez, sabe como veicular eventos e pedidos de dados ao resto do sistema.

A implementação concreta do interface `EventParser` é disponibilizada pela classe `SAXEventParser` do pacote `cmp.xml`. Desta vez optou-se pela utilização da API SAX como meio de interpretação do XML, na medida em que permite tempos de

[3]Não confundir com o URI do tipo URN (Uniform Resource Name).

processamento mais rápidos[4].

A figura 5.10 apresenta o resto do modelo de classes do pacote cmp que garante a disponibilização dos serviços de publicação/subscrição de tópicos e envio e recepção de eventos.

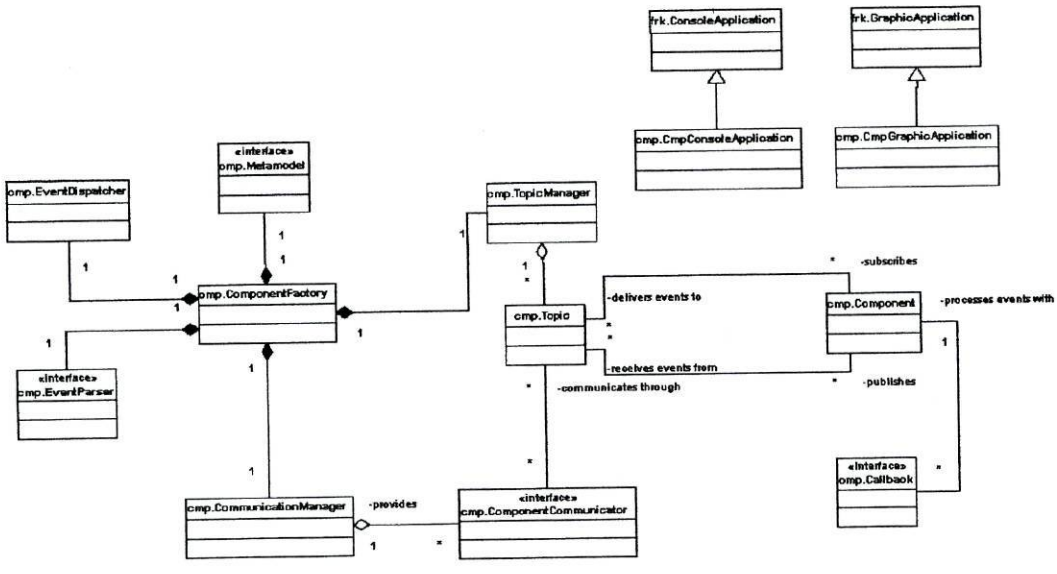


Figura 5.10: Diagrama de classes do pacote cmp

A classe `Component` modeliza o conceito de componente mas não implementa qualquer funcionalidade. Caberão às sub-classes procederem à sua especialização. O ciclo de vida de um componente é definido pela invocação de quatro métodos que devem ser correctamente implementados pelos seus descendentes (métodos *Templated*). A sua ordem de execução é garantida pela arquitectura e é a seguinte:

1. `connectInitialTopics ()` — serve para que os componentes possam registar-se nos tópicos tanto para publicação como para subscrição;
2. `connectInitialCallbacks ()` — local onde os componentes podem proceder ao registo de *callbacks* para o processamento de eventos;
3. `startup ()` — Início de processamento específico do componente;
4. `shutdown ()` — Indica o fim de processamento. Os componentes devem executar tarefas associadas à libertação de recursos, como ficheiros ou ligações via *socket*;

Apesar da estrutura apresentada, o registo de tópicos e *callbacks* pode ser executado em qualquer altura do ciclo de vida de um componente.

O componente fornece uma API que permite o registo de tópicos (classe `Topic`) tanto para publicação como para subscrição. O componente obtém as referências

[4]Tal como discutido no capítulo 3.

para os vários tópicos a partir da classe `TopicManager` que é a classe encarregada de gerir os tópicos, garantindo que dois componentes que registem o mesmo tópico obtenham uma referência para a mesma instância de `Topic`. É da responsabilidade da classe `Topic` guardar duas listas de componentes: uma lista de componentes que publicam eventos desse tópico e outra lista de componentes que subscrevem componentes desse tópico. O envio de eventos é conseguido pela execução do método `pushEvent()`. De forma análoga, o envio de pedidos de dados é efectuado através do método `pushDataRequest()`. Eventos e pedidos de dados são veiculados para os tópicos respectivos. Os tópicos mantêm uma associação com instâncias de `ComponentCommunication` que são as entidades que sabem como enviar eventos e pedidos de dados do ponto de vista físico. Este interface é implementado por classes especializadas que garantem a comunicação sobre diversos protocolos, tais como HTTP, IIOP (*Internet Inter-Orb Protocol*), RMI, etc[5]. A capacidade de implementação de comunicadores especializados para diversos protocolos será denominada por *modelo de comunicadores*. As instâncias de `ComponentCommunication` são obtidas a partir da classe `CommunicationManager`.

A recepção de eventos é conseguida pelo registo de instâncias de classes que implementam o interface `Callback`. Uma *callback* pode ser registada para a recepção de um evento específico ou para a recepção de todos os eventos de um determinado tópico. O mesmo se passa com o registo de *callbacks* para a recepção de pedidos de dados.

A construção de aplicações que utilizam componentes deve basear-se na especialização das classes abstractas `CmpConsoleApplication` e `CmpGraphicApplication` para aplicações consola e gráficas, respectivamente.

Como peça final do sistema existe a classe `ComponentFactory` (*singleton*) que permite a construção/obtenção dos objectos mais importantes para o suporte da estrutura de componentes: `EventParser`, `EventDispatcher`, `TopicManager`, `CommunicationManager` e `Metamodel`.

5.5 Sumário

Neste capítulo foi apresentado o projecto de *software* que orientou o desenvolvimento do *framework* de componentes Java.

Numa primeira fase, apresentaram-se as definições de *framework* e arquitectura de *software* tomadas em consideração durante a realização do trabalho. Com base nestes conceitos, justificou-se a implementação deste *framework* como forma de dar corpo aos princípios da arquitectura de componentes colaborantes apresentados no capítulo 4.

Do projecto apresentado, excluíram-se as especificidades de implementação da camada de comunicação, garantida pela possibilidade de especialização do modelo de comuni-

[5]Tal como foi referido no capítulo 4, a análise da implementação física não será objecto de estudo nesta dissertação.

cadores. Esta opção foi já justificada no capítulo 4 e visou, essencialmente, minimizar o dispêndio de tempo necessário ao desenvolvimento da biblioteca de funcionalidades SCADA, que irá ser apresentada do capítulo 6.

Capítulo 6

Biblioteca de Funcionalidades SCADA

Este capítulo apresenta o projecto de *software* ligado ao desenvolvimento da biblioteca de funcionalidades SCADA, em Java. Desta biblioteca fazem parte funcionalidades gráficas, úteis para o desenvolvimento de interfaces, e funcionalidades não-gráficas, pensadas para obviar o processo de desenvolvimento de aplicações SCADA genericamente.

A apresentação das funcionalidades culminará com a descrição do componente de simbologia, um componente gráfico Java que permite validar a tese avançada no capítulo 4.

Cada uma das funcionalidades descritas segue uma estrutura semelhante àquela que foi apresentada no capítulo anterior: contexto, descrição do problema e descrição funcional.

6.1 Descrição Geral dos Pacotes

A biblioteca de funcionalidades SCADA é constituída pelos pacotes `efa` e `toolkit`. O pacote `efa` disponibiliza ferramentas de carácter genérico que poderão ser utilizadas para o desenvolvimento de outras funcionalidades. Já o pacote `toolkit` apresenta funcionalidades de cariz marcadamente gráfico que visam dar resposta a alguns dos requisitos apresentados no capítulo 4.

Estes pacotes e seus sub-pacotes encontram-se ilustrados na figura 6.1.

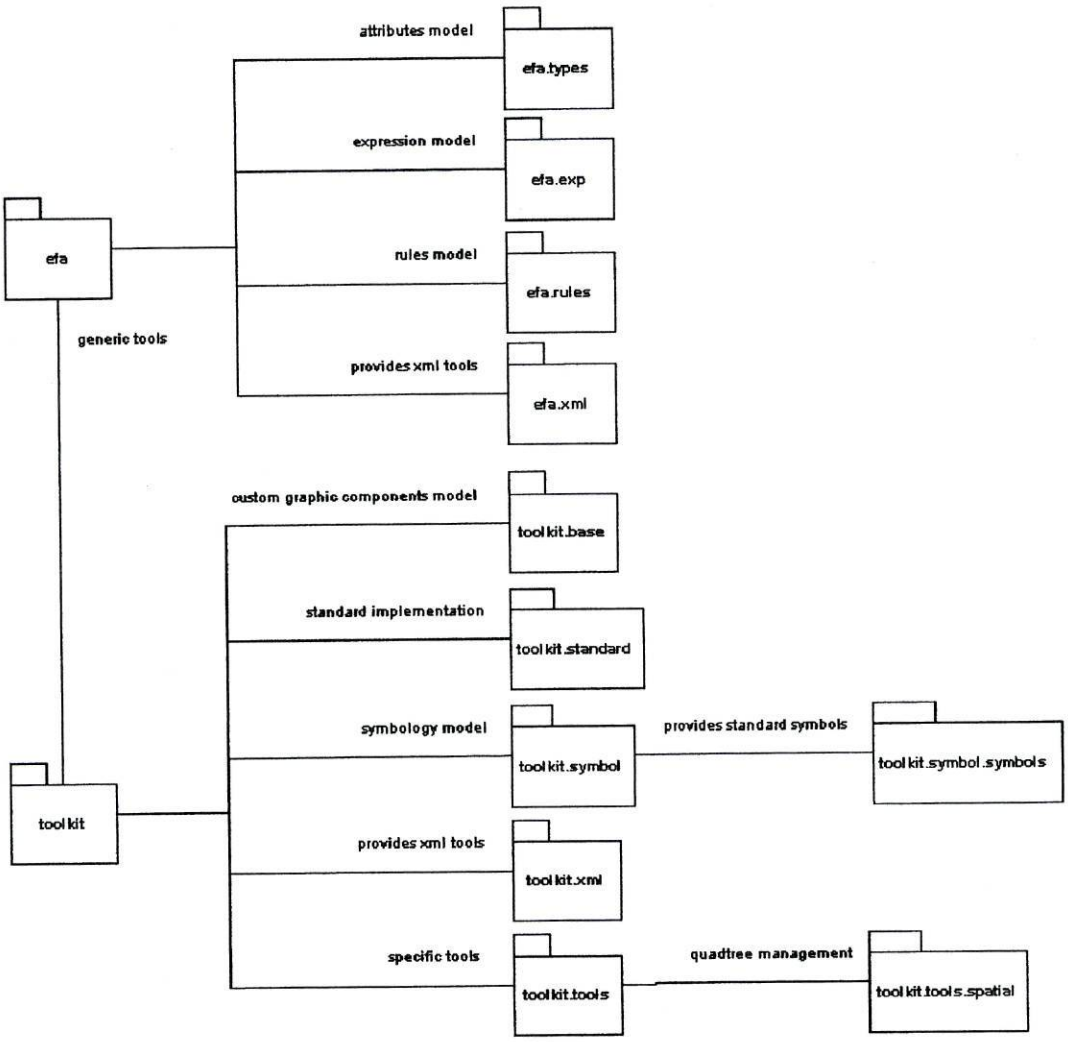


Figura 6.1: Estrutura geral de pacotes da biblioteca de funcionalidades SCADA

6.2 Modelo de Atributos

O modelo de atributos é disponibilizado pelos pacotes `efa` e `efa.types`.

6.2.1 Contexto

A informação referente a sistemas SCADA consiste normalmente na associação entre atributos de vários tipos (valor, data, identificador, alarme, etc) e entidades muito bem conhecidas. É assim importante desenvolver mecanismos que facilitem a gestão uniformizada de atributos independentemente dos seus tipos, pela definição de uma estrutura hierárquica baseada num único atributo genérico sem tipo definido[1].

6.2.2 Descrição do Problema

O modelo de atributos deve definir uma estrutura hierárquica de atributos baseada num único atributo genérico. Um atributo encapsula um tipo básico da linguagem (Java no caso presente) e contém um valor. Deve ser possível a realização de operações matemáticas e de comparação entre quaisquer atributos, embora seja previsível a existência de tipos de atributos incompatíveis. A realização de operações matemáticas ou de comparação entre atributos incompatíveis deve originar uma excepção. Todos os atributos compatíveis devem poder converter-se entre si. Todos os atributos devem ter uma representação em formato de texto. Todos os atributos devem saber converter-se para a sua representação textual e devem saber construir-se a partir dela. Um atributo deve ter um valor por omissão.

O modelo deve prever a existência de um gestor local de atributos e um gestor global de atributos. Um gestor local de atributos é identificado por um nome e mantém uma lista de elementos do tipo chave-valor, em que o valor é uma instância de um dos tipos de atributos definidos. Um gestor global também é identificado por um nome. Numa mesma aplicação pode apenas existir um gestor global por cada nome. O objectivo do gestor global é o de guardar uma lista de elementos do tipo chave-tipo, onde o tipo é o nome da classe de um dos tipos de atributos definidos. Um gestor global serve como metamodelo para vários gestores locais com o mesmo nome. Um gestor local pode obter uma instância do tipo do atributo referente a um dos seus elementos através do gestor global, a partir da chave desse elemento. As chaves são do tipo texto. Um gestor local de atributos deve permitir a escrita e leitura de todos os seus atributos.

[1]Este modelo é baseado num modelo semelhante existente na EFACEC SE/AS.

6.2.3 Descrição Funcional

Da descrição anterior, denota-se a existência de dois modelos: o modelo de atributos propriamente dito e o modelo de gestores de atributos.

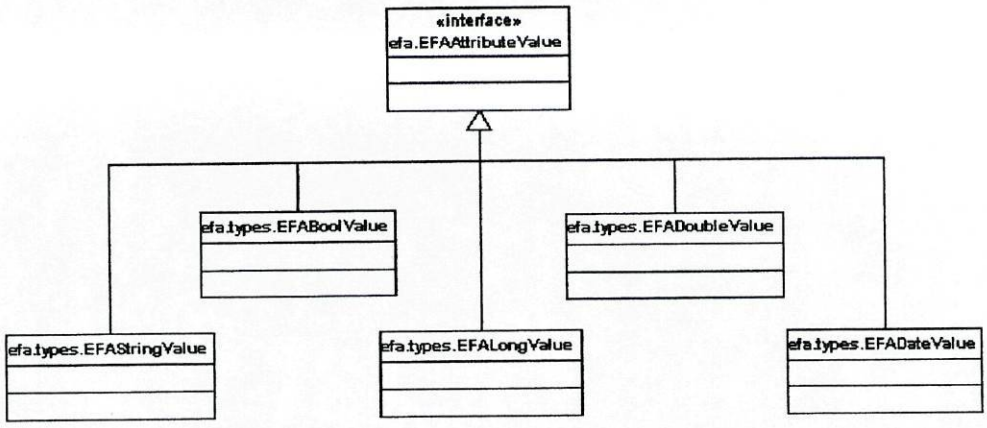


Figura 6.2: Diagrama de classes do pacote efa.types

A figura 6.2 apresenta o diagrama de classes do modelo de atributos definido no pacote `efa.types`. O interface `EFAAttributeValue` define o contrato que deve ser cumprido por todos as classes de atributos que o implementam: define um nome, prevê métodos de conversão para todos os tipos de atributos, define métodos de operação entre atributos (`add()`, `sub()`, `mul()` e `div()`), métodos de comparação entre atributos (`equals()` e `compareTo()`) e métodos de conversão de a para um formato de texto. As classes que implementam este interface são as que se encontram descritas na figura; como exemplo, as classes `EFAStringValue` e `EFABoolValue` encapsulam os tipos básicos `java.lang.String`[2] e `boolean`, respectivamente.

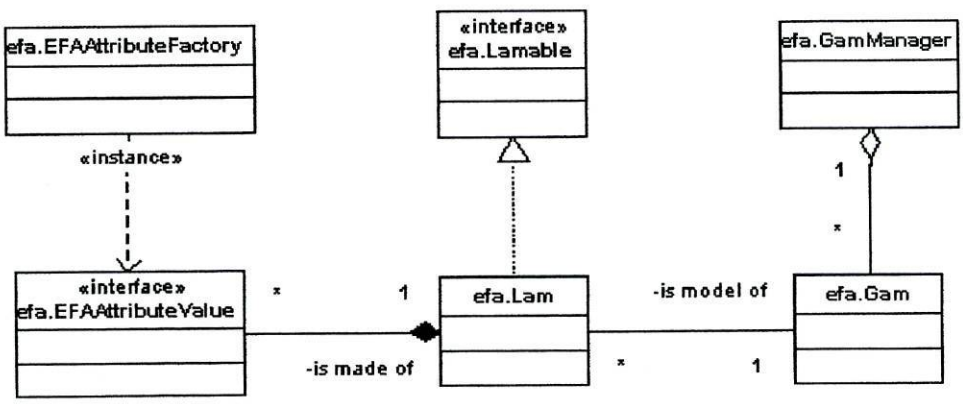


Figura 6.3: Diagrama de classes do pacote efa

Na figura 6.3 ilustram-se as classes principais do pacote `efa` onde se denotam as que definem o modelo de gestores de atributos. A classe `Lam` implementa o interface `Lamable`,

[2]Apesar da classe java.lang.String não ser, em rigor, um tipo básico, o modelo assume-a como tal.

comum a todas as classes que pretendam operar como gestor local de atributos, e mantém uma referência para o seu gestor global de atributos, ou seja, uma instância da classe `Gam` que lhe é passada no seu construtor. A classe `GamManager` implementa o padrão *singleton* e é responsável por garantir que não existe mais do que uma instância de `Gam` com um determinado nome. A classe `EFAAttributeFactory` disponibiliza um método estático para a construção de instâncias de `EFAAttributeValue`, a partir do nome da classe.

6.2.4 Voltando aos Eventos

Um exemplo de utilização do modelo de gestores de atributos foi já apresentado no ponto 5.4.4, onde a classe `Event` implementa o interface `Lamable`, sendo por isso um gestor local de atributos. São as suas características de gestor que permitem a escrita/leitura de atributos de eventos, eles próprios atributos baseados em `EFAAttributeValue`. Um evento obtém informação acerca do tipo dos seus atributos a partir de uma instância de `EventGam`. Como foi já referido, um `EventGam` constitui uma especialização da classe `Gam`, garantindo, assim, que todos os eventos com o mesmo nome partilharão o mesmo modelo de atributos.

6.3 Modelo de Expressões

O modelo de expressões encontra-se definido no pacote `efa.exp`.

6.3.1 Contexto

Algumas das funcionalidades disponibilizadas por aplicações SCADA consistem na capacidade de execução de filtros sobre características de entidades com vista à limitação do número de items a visualizar. Outras funcionalidades têm como base a capacidade de avaliação de condições que influenciam as características gráficas de apresentação da informação (ver capítulo 2).

A implementação destas funcionalidades pressupõe a existência de um mecanismo genérico que permita a criação, manutenção e avaliação de expressões baseadas em informação diversificada. Assim, o objectivo do modelo de expressões é o de fornecer uma infra-estrutura que permita a avaliação dinâmica de expressões a partir de valores presentes num gestor local de atributos.

6.3.2 Descrição do Problema

Uma expressão é constituída por um operador que tem um ou dois operandos. Um operador implementa uma operação básica (somar, dividir, comparação, etc) e sabe calcular o seu resultado em função dos valores fornecidos pelos seus operandos. Um operando pode ser uma constante, o valor de um atributo de um gestor local de atributos ou uma nova expressão. Uma expressão tem assim uma estrutura em árvore e pode conter um número indeterminado de níveis. A avaliação de uma expressão começa pela recepção de um gestor local de atributos que vai sendo passado aos níveis subsequentes para avaliação. O sucesso da avaliação de uma expressão depende da capacidade do seu operador em obter os seus operandos, fazendo com que o processo de avaliação ocorra de baixo (folhas) para cima (raiz).

Numa mesma expressão, todos os operandos baseados em gestores locais de atributos devem basear-se no mesmo gestor local (i.e. com o mesmo nome), ou então aceitar qualquer gestor local.

Uma expressão pode ser construída programaticamente (pela manipulação de objectos) ou a partir de uma representação em XML. A falha na avaliação de uma expressão deve dar origem a uma excepção bem conhecida.

Devem ser disponibilizados os operadores mais comuns para expressões aritméticas, expressões *booleanas*[3] e expressões relacionais. O resultado de uma expressão aritmética é uma instância de um dos tipos de atributos definidos no ponto 6.2. Por seu turno, a avaliação de expressões *booleanas* e relacionais resultam num valor *booleano* (verdadeiro ou falso).

6.3.3 Descrição Funcional

A figura 6.4 ilustra o diagrama com as classes principais do modelo de expressões que tem uma estrutura tripartida: há três tipos de expressões que se associam a operadores que por sua vez utilizam operandos.

A classe `RelationalExpression` modeliza o conceito de expressão relacional e está associada uma instância de `RelationalOperator`. Este, por sua vez, aceita dois operandos que são instâncias de classes que implementam o interface `AlgebraicOperand`. Este interface define operandos que devolvem valores que são instâncias de `efa.types.EFAAttributeValue`. Ambas as classes `RelationalExpression` e `RelationalOperator` devolvem valores booleanos.

A classe `ArithmeticExpression` define uma expressão aritmética. Expressões aritméticas são expressões baseadas em instâncias de `efa.types.EFAAttributeValue` que resultam numa outra instância dessa classe, ou melhor, numa de suas descendentes. Um `ArithmeticOperator` é a classe mãe de todas as operações aritméticas. Uma operação

[3]Este termo será utilizado para a caracterização de expressões que implementam álgebra de Boole.

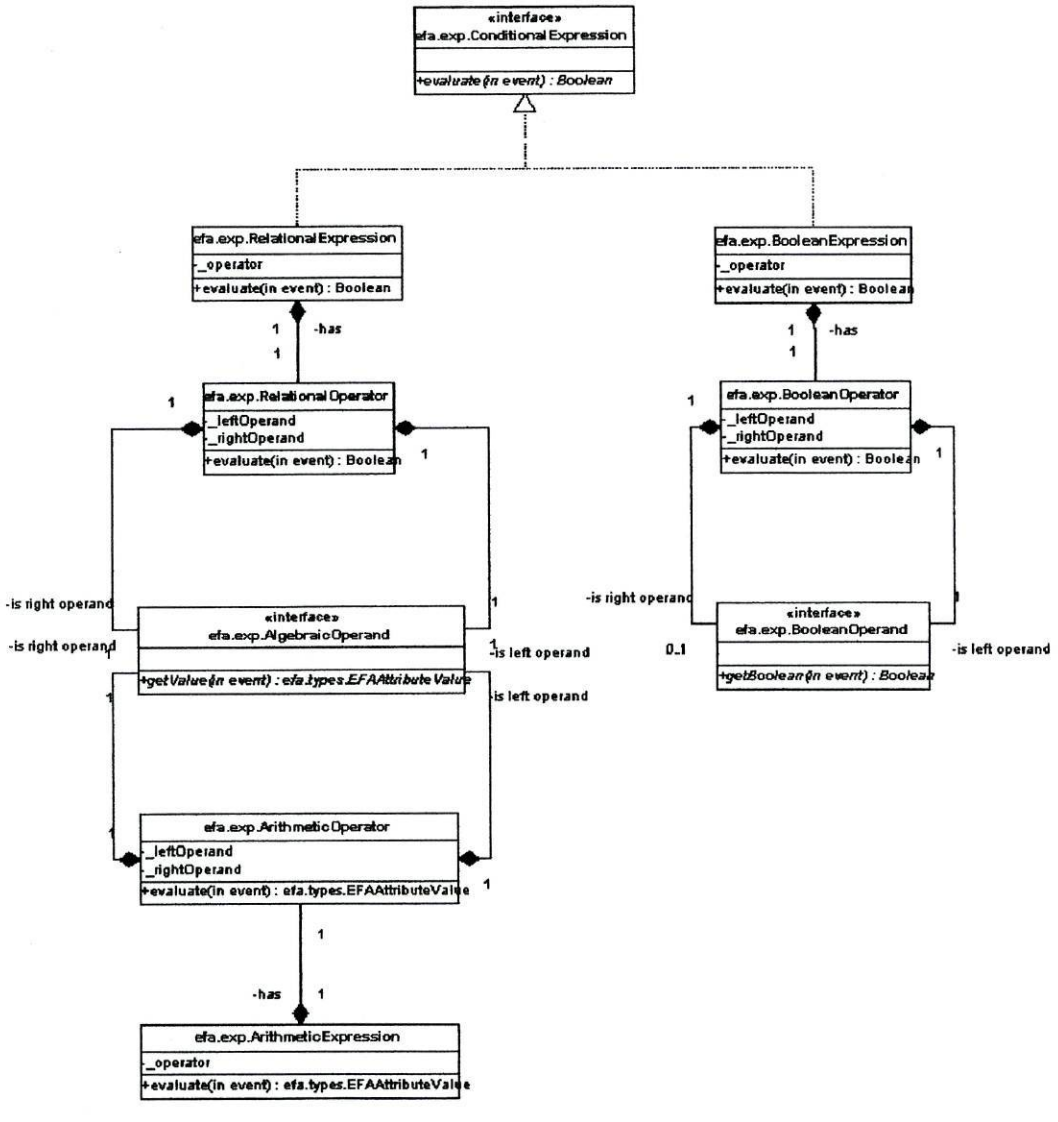


Figura 6.4: Diagrama de classes do pacote efa.exp — expressões

aritmética sabe efectuar operações também sobre instâncias de `efa.types.EFAAttributeValue`, obtidas através de instâncias de `AlgebraicOperand`.

O terceiro tipo de expressões é definido pela classe `BooleanExpression` que depende de uma instância de `BooleanOperator`. Um operador booleano efectua cálculos com base em instâncias de `BooleanOperand`, que é um interface que define o comportamento deste tipo de operandos. Mais concretamente, um operando booleano devolve um valor booleano. Expressões booleanas são expressões baseadas em valores booleanos.

A figura 6.4 apresenta ainda um outro interface implementado pelas classes `RelationalExpression` e `BooleanExpression`. Este interface é o `ConditionalExpression` e define o comportamento de expressões que resultam num valor booleano.

A modelização dos operandos, apresentada na figura 6.5, permite definir as classes que poderão ser utilizadas por cada um dos tipos de operadores para a construção final da estrutura de uma expressão.

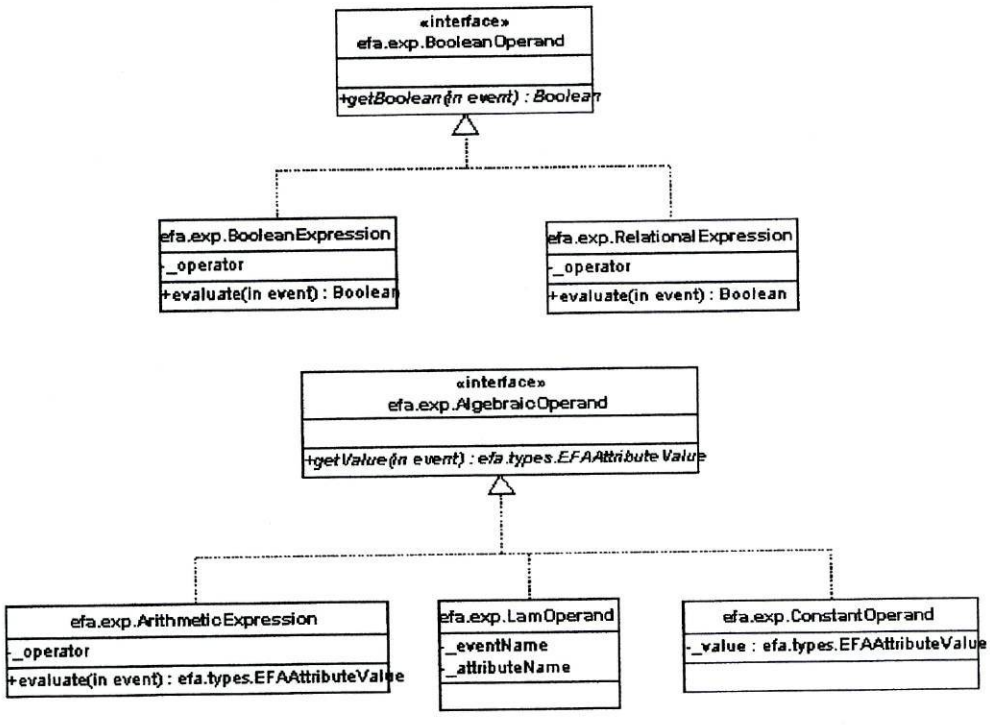


Figura 6.5: Diagrama de classes do pacote efa.exp — operandos

As classes `BooleanExpression` e `RelationalExpression` implementam o interface `BooleanOperand`, podendo por isso servir de operandos para expressões booleanas. Estes operandos têm a particularidade de retornarem sempre um valor booleano. Já os operadores algébricos, são as classes que implementam o interface `AlgebraicOperand` e que retornam instâncias da hierarquia de classes baseada em `EFAAttributeValue`. Destas fazem parte as classes `ArithmeticExpression`, `LamOperand` e `ConstantOperand`. A classe `LamOperand` constitui um operando que obtém o seu valor a partir de um dos atributos da instância de `efa.Lamable` que é entregue para avaliação da expressão. O

atributo e o gestor são identificados pelo seus nomes, que são guardados nos membros da classe `_attributeName` e `_lamName`, respectivamente. O processo de avaliação da expressão é interrompido pelo envio de uma excepção, no caso da expressão ser avaliada contra uma instância de `Lamable` de nome diferente do valor guardado em `_lamName`, ou no caso do atributo não se encontrar preenchido. Uma instância de `ConstantOperand` guarda uma instância de `EFAAttributeValue` que é entregue ao operador respectivo durante o processo de avaliação de expressões.

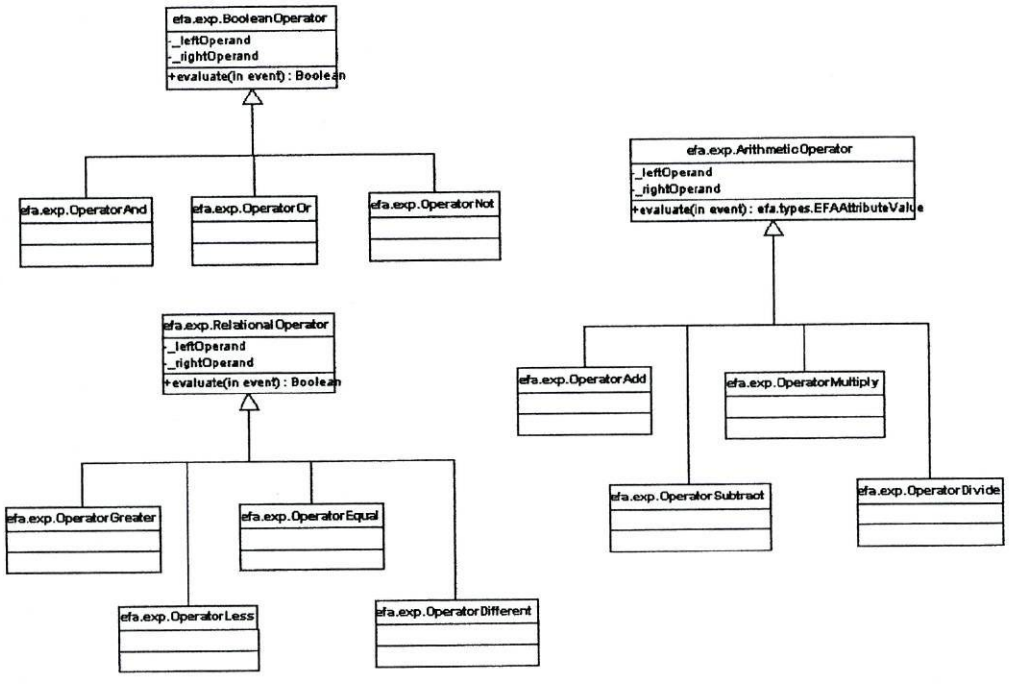


Figura 6.6: Diagrama de classes do pacote efa.exp — operadores

A figura 6.6 apresenta, finalmente, as especializações dos três tipos de operadores definidos: `BooleanOperator`, `RelationalOperator` e `ArithmeticOperator`.

6.3.4 Definição de Expressões em XML

A criação de uma gramática XML que permita a definição genérica de expressões reveste-se de particular interesse para o desenvolvimento de outras funcionalidades baseadas no modelo. Por dois motivos. Em primeiro lugar, porque é uma linguagem simples e cada vez mais conhecida, dotada de mecanismos de validação (sintáctica) pela utilização de DTDs ou de esquemas XML. Em segundo lugar, porque permite o desenvolvimento de aplicações cuja lógica de processamento pode ser completamente isolada da lógica de construção de expressões, pela utilização de definições externas que são posteriormente "carregadas" pelo modelo de expressões, para utilização imediata.

No apêndice B, encontra-se o esquema XML que define uma gramática para a escrita

de expressões em XML (`expressions.xsd`). A sua estrutura segue, naturalmente, a estrutura do modelo de classes apresentado.

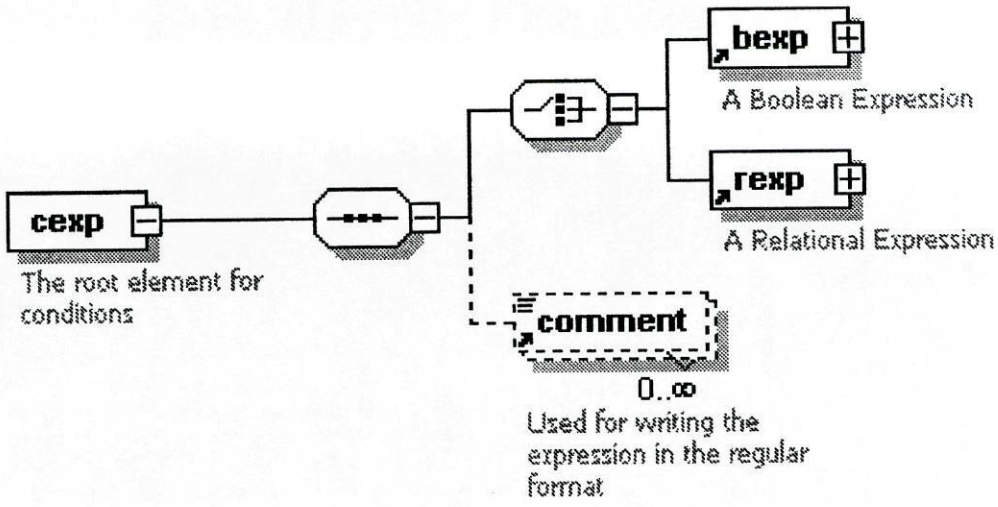


Figura 6.7: Esquema XML de expressões — expressão condicional

A descrição da gramática socorrer-se-á da utilização de representações gráficas do esquema XML em causa. Assim, a figura 6.7 apresenta o elemento raiz do modelo, o elemento `<cexp>`, que modeliza uma expressão condicional, isto é, uma expressão cujo resultado é um valor *booleano*[4]. Do modelo de classes da figura 6.4, verifica-se que uma expressão condicional pode ser modelizada por dois outros tipos de expressões: expressões *booleanas* e expressões relacionais. Esta situação é transposta para o esquema XML pela utilização de um dos elementos `<bexp>` ou `<rexp>`.

O elemento `<bexp>` é, por sua vez, composto por um dos elementos do tipo `BooleanOperator`, que são elementos que modelizam operações *booleanas*. Estes elementos estão ilustrados na figura 6.8 e são os elementos `<and>`, `<or>` e `<not>`. Os operandos de um operador *booleano* podem ser uma nova expressão *booleana* (elemento `<bexp>`) ou então uma expressão relacional (elemento `<rexp>`).

O elemento `<rexp>` (figura 6.9) é, por sua vez, composto por um dos elementos do tipo `RelationalOperator`. Este é o tipo dos elementos `<diff>`, `<equal>`, `<greater>`, `<less>`, `<isNull>` e `<isNotNull>` que aceitam como operandos (sub-elementos) um ou dois (dependendo das operações) dos elementos `<aexp>`, `<const>` ou `<lam>`. O elemento `<const>` é utilizado para descrever um valor constante, tal como o modelizado pela classe `ConstOperand`. Este valor é escrito segundo o formato textual dos atributos, tal como foi discutido durante a apresentação do modelo de atributos, no ponto 6.2. De forma semelhante, o elemento `<lam>` é uma imagem da classe `LamOperand` e permite

[4]A versão actual permite apenas a criação de expressões deste tipo, por terem sido as únicas utilizadas no desenvolvimento das restantes funcionalidades.

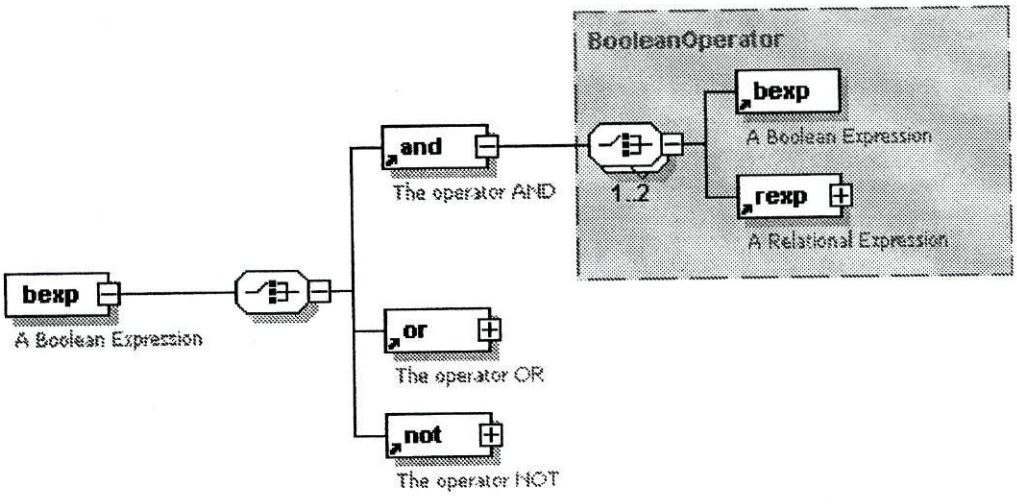


Figura 6.8: Esquema XML de expressões — expressão booleana

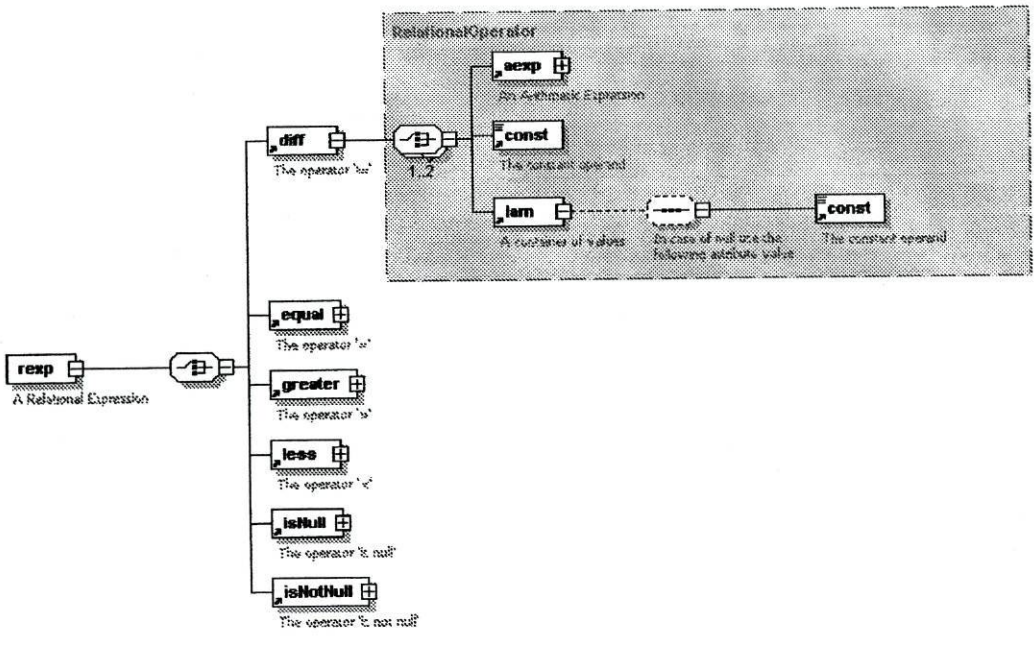


Figura 6.9: Esquema XML de expressões — expressão relacional

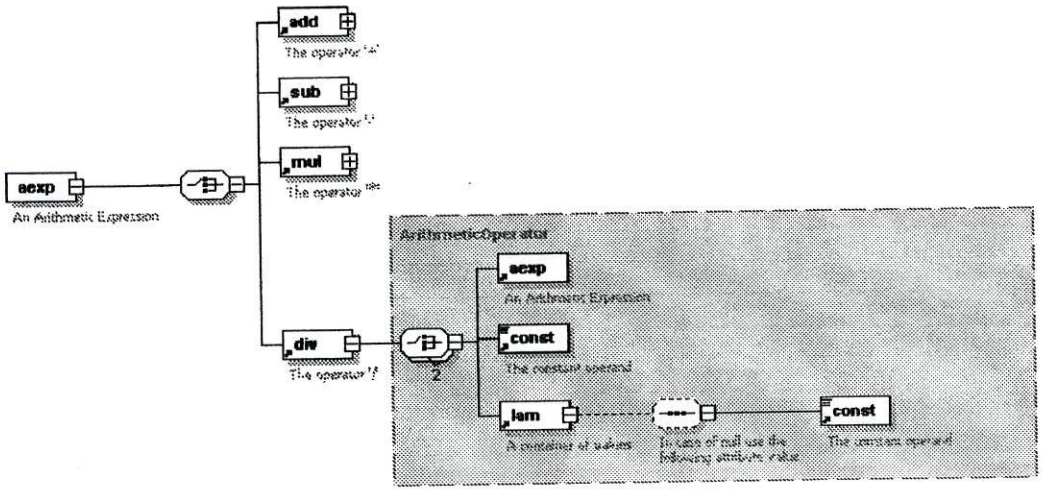


Figura 6.10: Esquema XML de expressões — expressão aritmética

descrever o nome do objecto `Lamable` a que corresponde, e o nome do atributo desse objecto cujo valor deve ser obtido. Este elemento permite ainda a utilização de um sub-elemento `<const>` contendo o valor a ser utilizado, no caso do atributo do objecto `Lamable` não se encontrar preenchido.

O elemento `<aexp>` modeliza uma expressão aritmética (figura 6.10) e é composto por um dos elementos `<add>`, `<sub>`, `<mul>` ou `<div>`, que são elementos do tipo `ArithmeticOperator`. Um elemento deste tipo é composto por dois dos elementos apresentados (podem ser os dois iguais): outra expressão aritmética (elemento `<axp>`), uma constante (elemento `<const>`) ou um operando baseado num `Lamable` (elemento `<lam>`).

O pacote `efa.xml` disponibiliza duas classes especializadas para a tradução de expressões em XML: a classe `DOMExpressionParser` e a classe `DOMExpressionParserEngine`. Ambas utilizam a API DOM para processamento de XML.

A título de exemplo, a seguinte expressão pode ser utilizada para verificar se o valor do atributo `Value` do evento `Analog`[5] é superior a 10, ou se o valor do atributo `Alarm` do mesmo evento é verdadeiro:

$$((Analog.Value + 10) > 20)\, OR\, (Analog.Alarm == true)$$

A mesma expressão pode ser construída em XML segundo o apresentado na figura 6.11.

[5]Note-se que um evento implementa o interface `Lamable`.

```
<cexp>
  <bexp>
    <or>
      <rexp>
        <greater>
          <aexp>
            <add>
              <lam name="Analog" attribute="Value"/>
              <const type="EFALongValue">10</const>
            </add>
          </aexp>
          <const type="EFALongValue">20</const>
        </greater>
      </rexp>
      <rexp>
        <equal>
          <lam name="Analog" attribute="Alarm"/>
          <const type="EFABoolValue">true</const>
        </equal>
      </rexp>
    </or>
  </bexp>
</cexp>
```

Figura 6.11: Exemplo de uma expressão em XML

6.4 Modelo de Regras

O modelo de regras encontra-se definido no pacote `efa.rules`.

6.4.1 Contexto

Qualquer tipo de lógica ligada à manipulação de informação beneficia da existência de modelos que permitam associar a avaliação de condições à execução de determinadas acções específicas, ou seja, uma regra. Motores de regras são ferramentas que permitem a gestão e execução de regras com vista à sua inclusão em sistemas que alteram o seu comportamento em função da ocorrência de determinadas condições.

Um exemplo típico deste tipo de sistemas são, as agora comuns, lojas virtuais. Nestes sistemas o utilizador está vulgarmente associado a um carrinho de compras que vai sendo preenchido com os produtos a adquirir. A utilização de motores de regras pode, por exemplo, propor descontos na compra de novos produtos em função da quantidade, ou do tipo de produtos já adquiridos pelo utilizador. É assim possível alterar o tipo de promoções oferecidas pela simples definição de novas regras, sem que com isso haja a necessidade de proceder a qualquer tipo de re-implementação da lógica do sistema[6].

A utilização de motores de regras tem sido amplamente aproveitada na indústria do *software* para a implementação de sistemas com este tipo de características [ILOb, Tec]. Um exemplo da sua importância é a recente conclusão do JCP94 (*Java Community Process 94*) que apresenta uma proposta para a definição de uma API Java para pacotes que implementam motores de regras, materializada numa especificação, a JSP94 (*Java Specification Request 94*)[7] [9402].

Ora, também os sistemas SCADA beneficiam deste tipo de funcionalidades, nomeadamente para o processamento das regras que condicionam a forma de visualizar a informação.

6.4.2 Descrição do Problema

O objectivo deste pacote é a modelização de um motor de regras simplificado. Uma regra define uma condição e contém uma acção, podendo ser descrita da seguinte forma:

```
SE <CONDIÇÃO>
ENTÃO <ACÇÃO>
```

[6] É comum aos motores de regras disponibilizarem ambientes de edição de regras que são posteriormente carregadas pelas aplicações.

[7] O processo de definição desta proposta foi simultâneo ao desenvolvimento do presente trabalho, não tendo sido, por isso, possível analisá-lo antes da implementação apresentada nesta secção.

Uma condição é modelizada por uma expressão condicional, isto é, uma expressão cujo resultado é um valor *booleano*. As expressões são baseadas em gestores de atributos locais. A condição de uma regra é avaliada sempre a pedido. O sucesso da avaliação da condição de uma regra deve resultar na exposição da sua acção. Uma acção é uma porção de texto cujo significado deve ser compreendido pelos clientes do modelo. As regras estão associadas em "sacos de regras". Um saco de regras é univocamente identificado por um nome e tem associada uma acção por omissão. Os clientes do modelo podem pedir a execução das regras de um saco de regras. O resultado da execução das regras de um saco de regras devem ser todas as acções referentes às regras que foram avaliadas com sucesso ou, no caso de nenhuma regra ser correctamente avaliada, a regra por omissão do saco de regras.

Um motor de regras mantém um conjunto de sacos de regras que podem ser obtidos através do seu nome. Deve ser possível a definição de regras em XML e o seu carregamento num motor de regras.

6.4.3 Descrição Funcional

A figura 6.12 apresenta a estrutura do modelo de regras.

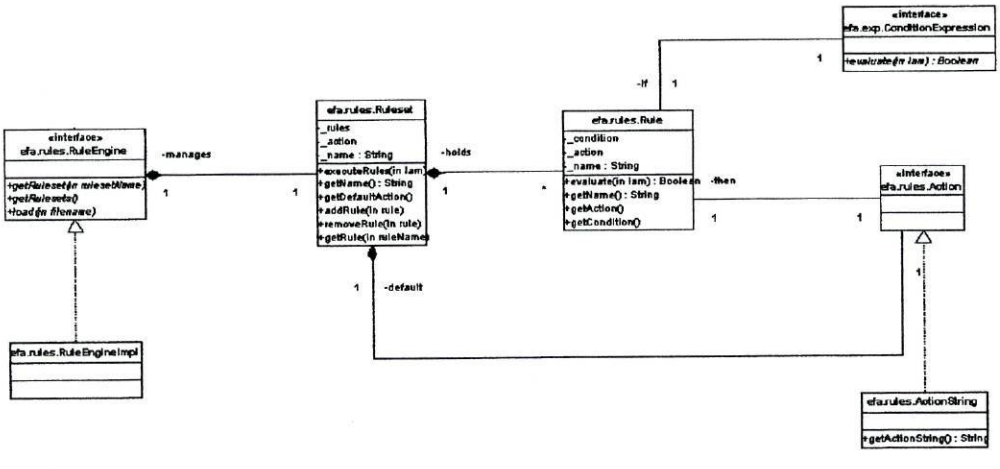


Figura 6.12: Diagrama de classes do pacote efa.rules — regras

A classe `Rule` implementa as funcionalidades de uma regra; tem um nome e mantém associações com uma instância de `efa.exp.ConditionalExpression` e uma instância de `Action`. Este interface (`Action`) não define qualquer método e serve apenas como interface de identificação[8] com vista a ser implementado por qualquer classe que possa

[8]Esta noção pretende substituir e expressão inglesa *tag interface* que identifica interfaces que são utilizados para dar um significado específico às classes que os implementam, em vez de definirem *a priori* o seu comportamento.

ser utilizada como acção de uma regra. Caberá aos cliente do modelo de regras a correcta definição das classes usadas como acção no âmbito específico dos seus problemas. O pacote disponibiliza a classe `ActionString` como sendo uma acção que encapsula um valor textual. É possível a avaliação da condição de uma regra contra uma instância de `efa.Lamable`, pela utilização do método `evaluate()`. Uma regra expõe o seu nome, condição e acção através dos métodos apresentados no modelo da classe.

A classe `Ruleset` modeliza o conceito de "saco de regras" atrás definido. Cada instância desta classe tem um nome, é composta por um conjunto de regras e tem uma acção por omissão. É possível adicionar ou remover regras de um `Ruleset` pela utilização dos métodos `addRule()` e `removeRule()`, respectivamente. É igualmente possível obter uma regra presente num `Ruleset`, pela utilização do método `getRule()`. O processamento das regras de um objecto desta classe é possível pela invocação do método `executeRules()`, ao qual deve ser passada uma instância de `Lamable` para avaliação contra as condições das várias regras existentes. O resultado é uma lista contendo as acções de todas as regras avaliadas com sucesso ou, a regra por omissão, no caso de não haver regras correctamente avaliadas.

O interface `RuleEngine` define o contrato para motores de regras. Um motor de regras é composto por instâncias de `Ruleset`. Um motor de regras permite obter uma instância de `Ruleset` através do seu nome por invocação do método `getRuleset()`. Um motor de regras permite também obter uma lista com todas as instâncias de `Ruleset` que controla (método `getRulesets()`). Um motor de regras deve ainda saber carregar regras definidas externamente num ficheiro, identificado através de um URL. A classe `RuleEngineImpl` é uma implementação concreta de um motor de regras que sabe como construir regras a partir de um documento XML que segue uma gramática bem definida, e que será analisada de seguida.

6.4.4 Definição de Regras em XML

O apêndice C apresenta o esquema XML que define a gramática para a construção de regras e sacos de regras em XML (`rules.xsd`). Este esquema utiliza as definições contidas no esquema `expressions.xsd`, disponíveis no apêndice B para consulta.

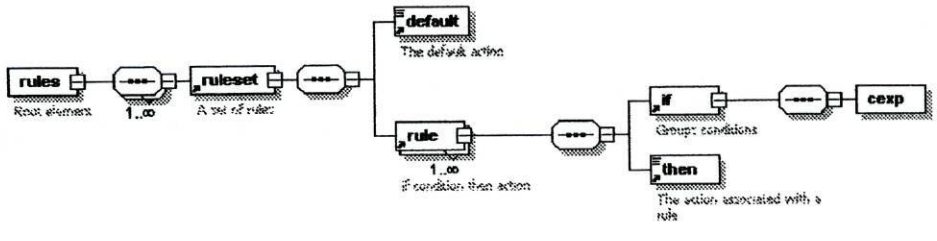


Figura 6.13: Representação gráfica do esquema XML para o modelo de regras

O diagrama 6.13 apresenta a representação gráfica do esquema XML para o modelo de regras. Neste modelo, o elemento `<rules>` é a raiz e modeliza um motor de regras. Este elemento é composto por um conjunto de elementos `<ruleset>` que modelizam sacos de regras. Um `<ruleset>` tem um nome e é constituído sempre por um elemento `<default>` (que constitui a acção por omissão) e por vários elementos `<rule>`. Este elemento representa uma regra, identificada por um nome, e tem dois sub-elementos: o elemento `<if>` e o elemento `<then>`. Enquanto que o elemento `<then>` modeliza a acção associada à regra, o elemento `<if>` é constituído por uma expressão (elemento `<cexp>`) que segue a estrutura definida no esquema `expressions.xsd`.

As classes que permitem a interpretação do modelo encontram-se definidas no pacote `efa.xml`. Destas, duas são exclusivas do modelo de regras (`DOMRulesParser` e `DOMRulesParserEngine`) e outra é reutilizada do modelo de expressões (`DOMExpressionParserEngine`). Todas as classes utilizam a API DOM para processamento de XML.

O exemplo 6.14 descreve um motor de regras com um saco de regras (denominado *MyRuleset*) onde é definida uma única regra chamada `AlarmRule`. Esta regra reutiliza a expressão definida na secção anterior na figura 6.11. A avaliação correcta da regra (se o valor do atributo `Value` do evento `Analog` fôr superior a 10 ou se o atributo `Alarm` do mesmo evento fôr verdadeiro) resulta na exposição da acção *SetAlarmOn*. A regra por omissão do saco de regras corresponde ao texto *SetAlarmOff*.

6.5 Modelo de Componentes Gráficos

O pacote `toolkit` contém o conjunto das ferramentas gráficas desenvolvidas neste trabalho. Dessas, o modelo de componentes gráficos é implementado pelo pacote `toolkit.base`. Este modelo é baseado no *HotDraw*, um conhecido *framework* utilizado por [E.J92] para demonstrar a documentação de *frameworks* pela utilização de padrões de desenho de *software*.

6.5.1 Contexto

A tese apresentada no capítulo 4 defende a *"...utilização de componentes Java especializados..."* para a construção de aplicações gráficas na *Web*. A utilização desta abordagem tem, contudo, o defeito de requerer o desenvolvimento específico de componentes para cada tipo de funcionalidade gráfica SCADA, sejam sinópticos, listas de alarmes, gráficos de tendência ou outras funcionalidades. Não sendo praticável o desenvolvimento de todas estas funcionalidades no tempo disponível para a realização deste trabalho, procurou-se a definição de uma estrutura genérica que suporte o desenvolvimento de componentes especializados. Esta estrutura baseia-se na definição de um componente de alto-nível que incorpora facilidades presumivelmente úteis para a implementação de componentes SCADA, nomeadamente as facilidades previstas nos modelos de interacção gráfica e de gestão de animações que serão apresentados.

```
<rules>
  <ruleset name="MyRuleset">
    <default>SetAlarmOff</default>
    <rule name="AlarmRule">
      <if>
        <cexp>
          <bexp>
            <or>
              <rexp>
                <greater>
                  <aexp>
                    <add>
                      <lam name="Analog" attribute="Value"/>
                      <const type="EFALongValue">10</const>
                    </add>
                  </aexp>
                  <const type="EFALongValue">20</const>
                </greater>
              </rexp>
              <rexp>
                <equal>
                  <lam name="Analog" attribute="Alarm"/>
                  <const type="EFABoolValue">true</const>
                </equal>
              </rexp>
            </or>
          </bexp>
        </cexp>
      </if>
      <then>SetAlarmOn</then>
    </rule>
  </ruleset>
</rules>
```

Figura 6.14: Definição de uma regra em XML

Uma vez que a maioria das aplicações SCADA trata do problema da apresentação gráfica de informação, este pacote define ainda um componente de visualização genérico.

6.5.2 Descrição do Problema

O objectivo deste pacote é o de definir uma estrutura de base para o desenvolvimento de componentes especializados. A hierarquia assenta num componente não visualizável, denominado por opaco, que tem uma associação com uma caixa de ferramentas. Uma caixa de ferramentas é um componente que contém ferramentas representadas na forma de botões. A selecção de um botão activa a ferramenta correspondente. Só é possível haver uma ferramenta activa a cada momento. Uma ferramenta pode também ser activada ou desactivada programaticamente, de modo a permitir a utilização de uma caixa de ferramentas não visualizável. Uma ferramenta define o modo de interacção com o componente opaco; todos os eventos gráficos (rato e teclado) ocorridos no componente opaco devem ser veiculados para a ferramenta activa. Se não houver nenhuma ferramenta activa, ou se a ferramenta activa não estiver interessada em processar o evento gráfico, então o componente opaco deve processar o evento normalmente. O evento de botão direito do rato é um evento especial que não deve ser entregue à ferramenta activa. Este evento deve ser utilizado para a criação de uma *popup* com opções segundo as necessidades das sub-classes do componente opaco.

O componente opaco deve dispôr de um relógio[9] para o registo de tarefas periódicas. Uma tarefa é identificada por um nome e tem passos; cada passo está associado a um intervalo de tempo. O componente opaco deve dispôr de um mecanismo de notificação (método) do fim da execução de um determinado passo de uma tarefa.

O pacote deve ainda disponibilizar um componente visualizador de figuras. Um visualizador tem um contexto gráfico onde podem ser definidos os parâmetros de ampliação, arrastamento ou rotação da sua área de desenho. O contexto gráfico do visualizador pode ser alterado externamente. Uma figura é uma entidade que sabe desenhar-se no visualizador e pode ser de tamanho fixo ou de tamanho variável. Uma figura de tamanho fixo deve manter as suas dimensões independentemente do nível de ampliação do contexto gráfico do visualizador. Uma figura variável varia de tamanho em função do nível de ampliação do contexto gráfico do visualizador. As figuras estão associadas num contentor de figuras para gestão espacial. O visualizador delega no contentor a responsabilidade de desenhar figuras. Um contentor sabe que figuras devem ser desenhadas a cada momento. Um contentor sabe quais as características gráficas que devem ser aplicadas durante o desenho de figuras (côr de fundo, côr do traço, tipo de traço, tipo de fonte, etc). As figuras podem registar-se no seu contentor para refrescamento periódico. Uma figura informa o contentor sempre que precisar de ser redesenhada. O processo de desenho parte sempre do visualizador, embora o contentor possa pedir, explicitamente, a execução de um desenho ao visualizador.

[9]Designado vulgarmente por *heart-beat*.

6.5.3 Descrição Funcional

A figura 6.15 ilustra o modelo de classes associadas à definição do componente opaco (classe `BaseComponent`). Este componente mantém uma associação com uma instância de `Toolbox` que pode ser partilhada por vários componentes opacos. Uma `Toolbox` pode ser inibida (inibindo a selecção de ferramentas) e fornece métodos para adicionar, remover e seleccionar ferramentas. O método `checkAndProcess()` permite a entrega de eventos gráficos à ferramenta activa da caixa de ferramentas. Este método retorna um valor *booleano* indicando se o referido evento gráfico foi ou não processado por alguma das suas ferramentas. Uma ferramenta é modelizada pelo interface `Tool` que estende o interface `Interactor`.

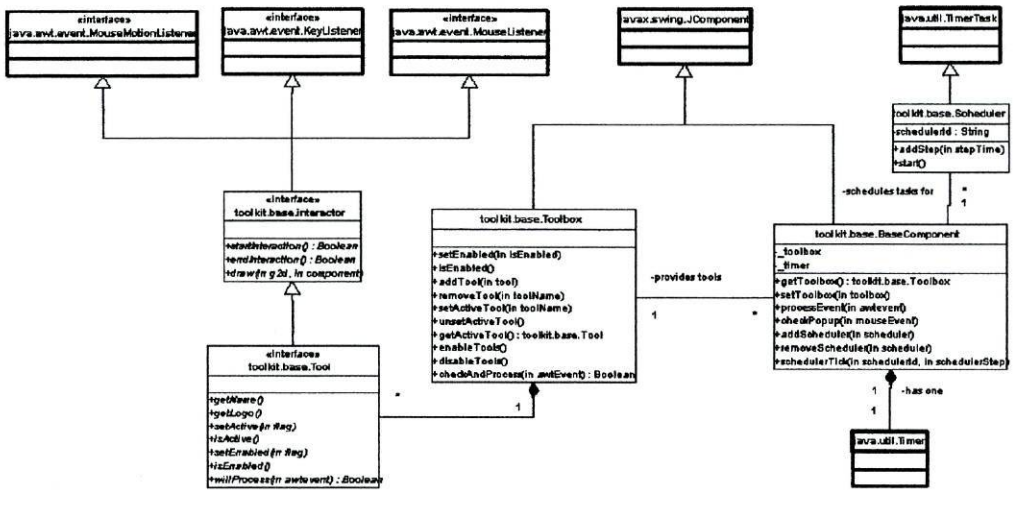


Figura 6.15: Diagrama de classes do pacote efa.base — Componente Opaco

Um interactor estende os interfaces de recepção gráfica `java.awt.event.MouseListener`, `java.awt.event.MouseMotionListener` e `java.awt.event.KeyListener` para processamento dos eventos gráficos correspondentes. Um interactor disponibiliza o método `startInteraction()`, que aceita um evento gráfico como parâmetro e que indica da sua disposição de processar o evento gráfico em causa. Um interactor decide se deve, ou não, interagir em função das características do evento gráfico (ex. associação de teclas). Assim, a utilização de vários interactores deve basear-se em condições de activação diferentes. Um interactor indica se acabou a sua interacção através do método `stopInteraction()`. Um interactor só pode actuar se nenhum outro interactor o estiver a fazer. Uma ferramenta é um caso específico de um interactor cuja condição de actuação é o facto de ser activado externamente (e não através de uma associação de teclas, por exemplo). Uma ferramenta actua enquanto não fôr desactivada. As ferramentas utilizam o padrão de desenho conhecido como *State* [EGV97, p 305]. Neste padrão, o contexto (`BaseComponent`) delega a execução de tarefas associadas à interacção com o utilizador ao estado corrente, ou seja, à ferramenta seleccionada.

A classe `BaseComponent` guarda uma instância de `java.util.Timer` para registo de objectos do tipo `Scheduler`. Um `Scheduler` modeliza o conceito de tarefa com passos temporizados. Esta classe é uma especialização de `java.util.TimerTask`. O componente permite a adição ou remoção de tarefas temporizadas através dos métodos `addScheduler()` e `removeScheduler()`, respectivamente. A execução de um passo resulta na invocação, no componente, do método `schedulerTick()` (método *Templated*), que deve ser implementado pelas classes derivadas. A natureza `multi-threaded` do temporizador implica que este método seja *sincronizado*[10].

A implementação do componente visualizador levou à necessidade de utilização de transformadas espaciais para a execução de ampliações, arrastamentos e rotações. O Java disponibiliza na sua API a classe `java.awt.geom.AffineTransform` que permite efectuar transformações lineares de coordenadas 2D para outras coordenadas 2D, que mantêm a "rectidão" e paralelismo das linhas [Sun, java.awt.geom.AffineTransform].

A classe `java.awt.Graphics` encapsula toda a informação necessária para a realização de operações gráficas em componentes Java [Sun, java.awt.Graphics]. A utilização da classe `java.awt.Graphics2D`, por seu turno, permite um controlo mais sofisticado sobre a geometria, transformações de coordenadas, gestão de côr e apresentação de texto em componentes gráficos [Sun, java.awt.Graphics2D]. Todas as coordenadas recebidas por este objecto são especificadas num sistema de coordenadas abstracto[11] denominado por *espaço de utilizador*[12]. Cada instância de `Graphics2D` contém uma instância de `java.awt.AffineTransform`[13] que lhe permite converter coordenadas definidas no espaço do utilizador em coordenadas definidas no *espaço do dispositivo*[14] [Sun, java.awt.Graphics2D].

Uma forma de manipular as características gráficas de uma imagem consiste na alteração da transformada utilizada pelo contexto gráfico de visualização. É assim possível manter um modelo de dados constante, baseado em coordenadas do utilizador, independentemente do resultado final da sua representação que pode ser ampliada, deslocada ou mesmo rodada. A conversão inversa de coordenadas (para, por exemplo, converter o ponto referente à posição do ponteiro do rato para o espaço do utilizador) requer a utilização da transformada inversa à utilizada pelo contexto gráfico em causa.

Por forma a facilitar o processo de mapeamento entre coordenadas do utilizador e coordenadas do dispositivo, definiu-se um modelo de conversão que é apresentado na figura 6.16. Os interfaces `PointTransformer` e `ShapeTraformer` definem um contrato para pontos e geometrias que sabem a sua representação tanto em coordenadas de utilizador como em coordenadas do dispositivo. Isto possibilita a utilização das suas instâncias independentemente do sistema de coordenadas, já que permite que classes

[10] O termo refere-se ao mecanismo de exclusão mútua disponível na linguagem Java através da utilização da partícula *synchronized*.

[11] Independente do dispositivo físico utilizado para representação, i.e. tipo de monitor, impressora, etc.

[12] *User Space*.

[13] Aqui designada genericamente pelo termo *transformada*.

[14] *Device Space*.

especializadas possam, sempre que necessário, utilizar o sistema que lhes fôr mais conveniente. Estes interfaces serão doravante denominados por pontos e geometrias de conversão.

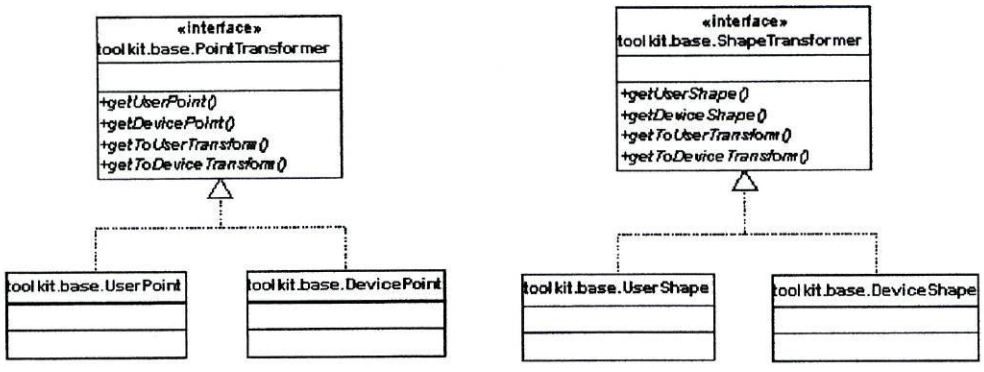


Figura 6.16: Diagrama de classes do pacote efa.base — mapeamento de coordenadas

O modelo da figura 6.17 ilustra as classes que interagem com o componente visualizador (classe `CanvasComponent`). Este componente constitui uma extensão da classe `BaseComponent`, sendo composto por uma instância de `FigureContainer`. Um `FigureContainer` é um interface para contentores de figuras que define uma API que incide em três áreas: gestão espacial de figuras, gestão gráfica e escalonamento. A gestão espacial inclui métodos que permitem obter conjuntos de figuras em função de um ponto ou de uma área. Estes métodos aceitam pontos e geometrias de conversão como parâmetros. A gestão gráfica inclui métodos de desenho e métodos para o registo e obtenção de áreas de redesenho[15] (por terem sofrido alterações). Finalmente, os métodos de escalonamento permitem a associação de figuras a tarefas periódicas, de modo a permitir a implementação de objectos dinâmicos (animações, por exemplo). A classe `Figure` define um interface que deve ser implementado por todas as classes que se pretendam representar num visualizador. Este interface define que uma figura sabe desenhar-se, sabe a sua posição, sabe a área que ocupa, sabe comparar-se com outras figuras para ordenação, sabe se é visível ou não e sabe se está ou não seleccionada. As coordenadas de uma figura são sempre representadas no espaço do utilizador. Uma figura tem a possibilidade de criar um menu do tipo *popup* e pode conter outras figuras pela implementação do padrão *Composite* [EGV97, p 163]. Um contentor está também associado a uma instância de `Palette` que fornece toda a informação gráfica a utilizar no processo de desenho. Esta informação é passada ao visualizador para actualização do seu contexto gráfico.

O papel do componente visualizador é o de expôr a transformada de desenho e garantir a correcta representação dos objectos pela manipulação de uma imagem *off-screen* que é actualizada em função da área de redesenho (fornecida pelo contentor) e pela sua dimensão actual. Os pedidos de redesenho são efectuados por chamadas aos métodos

[15] `Clipping path`. Este termo corresponde à definição da porção de uma imagem que será de facto redesenhada.

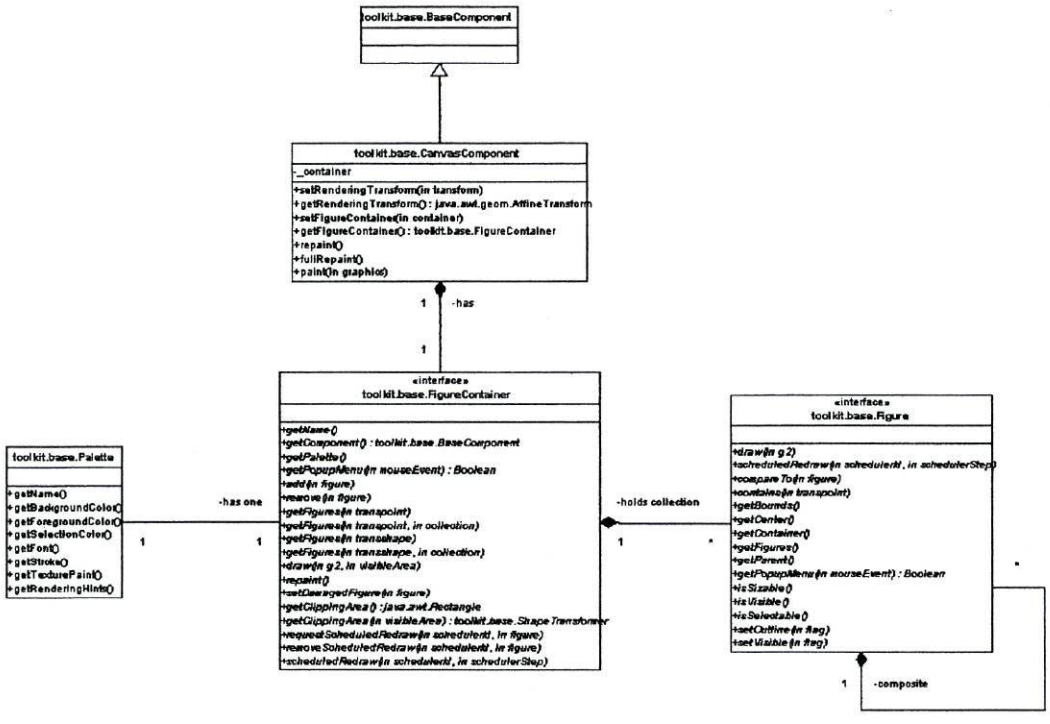


Figura 6.17: Diagrama de classes do pacote efa.base — componente visualizador

`repaint()` e `fullRepaint()`. A diferença reside no facto do primeiro ter em conta informação de áreas de redesenho e o segundo não. A utilização de uma imagem *off-screen* permite evitar a ocorrência do efeito de *flashing* e constitui uma técnica conhecida como *double-buffering*.

6.6 Gestão Espacial

O pacote `toolkit.tools.spatial` disponibiliza ferramentas para organização de objectos num plano.

6.6.1 Contexto

Recordando a caracterização de sistemas SCADA para sistemas eléctricos de energia (capítulo 2), vimos que são sistemas responsáveis pela supervisão de milhares de pontos de leitura de dados, o que resulta, muitas vezes, na existência de diagramas extensos e contendo grandes volumes de informação. Como forma de facilitar a interacção com estes diagramas, vimos ser comum disponibilizarem-se mecanismos de navegação baseados na variação do nível de ampliação, arrastamento da área de visualização ou na distribuição da informação por camadas de visualização.

O problema resume-se à necessidade de utilização de mecanismos que permitam gerir o tipo de informação representada de forma adequada. Ora um diagrama é uma representação bidimensional de informação e, como tal, depende da existência de algoritmos de gestão espacial.

Um exemplo. Um diagrama representa 30 mil entidades, das quais, apenas 50 se encontram na área visível da aplicação. O operador selecciona uma das 50 entidades visíveis, com o ponteiro do rato, para obter informação detalhada. Neste exemplo, a inexistência de uma estrutura de gestão de informação espacial obrigaria a que o programa (visualizador) precisasse de percorrer todas as 30 mil entidades com vista a descobrir qual, ou quais, correspondem às coordenadas definidas pelo ponteiro do rato (teste de localização[16]). Com a utilização de um sistema de gestão espacial, seria possível obter primeiro o conjunto de entidades prováveis[17], em função da sua posição no diagrama, e somente estas seriam percorridas para descobrir qual a entidade realmente afectada.

Poder-se-á argumentar que com a capacidade de computação actual, o processo de pesquisa em 30 mil entidades não é significativo. Mas e se forem 60 mil? E se forem 100 mil? E se forem ainda mais? Para além disso existem outras situações que beneficiam da existência de algoritmos de gestão espacial para além dos testes de localização.

Voltando ao exemplo anterior, imaginemos que o operador executa uma ampliação no diagrama de modo a visualizar apenas 20 das 50 entidades visíveis. Mais uma vez, a inexistência de um sistema de gestão espacial faria com que o programa tivesse que percorrer todas as 30 mil entidades, por forma a descobrir apenas as 20 que seriam desenhadas ou, em alternativa, desenhava todas, mesmo aquelas posicionadas fora da área visível. Em qualquer dos casos o tempo de processamento seria consideravelmente superior ao tempo gasto pelo desenho apenas das entidades visíveis. Este problema será tanto mais grave quanto mais próximos forem, no tempo, os pedidos de ampliação, como no caso da execução contínua de ampliações. O mesmo tipo de problema aplica-se ao processamento de pedidos de arrastamento da área visível do diagrama.

Com vista à resolução destes problemas, utilizou-se uma biblioteca de gestão espacial de informação baseada em *quadtrees*, disponibilizada pela OpenMap [Ope].

A utilização de *quadtrees* permite a execução de pesquisas espaciais de forma rápida, mas requer tempos de construção um pouco longos, como é aliás normal em estruturas de dados baseadas em critérios de organização. Este é um compromisso aceitável para os problemas relacionados com sistemas SCADA, uma vez que são sistemas que tratam de informação pouco variável em termos espaciais.

[16] *hit testing.*

[17] O conjunto das entidades retornadas pelo sistema depende do tipo de algoritmo utilizado

6.6.2 Descrição do Problema

Uma *quadtree* é uma estrutura em árvore que permite a organização de objectos dispostos numa representação bidimensional. A área é recursivamente subdividida em quadrantes que formam uma grelha de representação. A adição de um objecto à `quadtree` coloca-o na célula da grelha correspondente à sua posição no diagrama.

Cada nó de uma `quadtree` gere todos os objectos de um quadrante. Assim que o número de elementos de um nó ultrapassa um valor máximo pré-determinado, o nó subdivide-se em mais quatro nós, um para cada sub-quadrante, aos quais são entregues os objectos respectivos[18].

A biblioteca `openmap-4.2` apresenta uma implementação de uma *quadtree* capaz de gerir pontos (*PR-quadtree* [Sam95]). Pretende-se evoluir a implementação por forma a desenvolver uma *quadtree* para a gestão de objectos rectangulares.

Esta implementação deve permitir a adição e remoção de figuras em função da sua *caixa envolvente*[19], e deve fornecer dois tipos de métodos de acesso aos objectos que controla:

1. métodos que retornem todos os objectos do menor quadrante que contém um determinado ponto;
2. métodos que retornem todos os objectos de todos os quadrantes que interceptam uma determinada geometria;

6.6.3 Descrição Funcional

A figura 6.18 apresenta o diagrama das classes de gestão espacial. Uma `QuadTree` é a estrutura principal, sendo composta por uma instância de `QuadTreeNode` (nó), a raiz. Cada instância desta classe é responsável pela gestão de uma área rectangular definida por uma instância de `QuadTreeRect`. Os objectos geridos por um nó são "embrulhados" em instâncias de `QuadTreeLeaf` (folha) que define a sua caixa envolvente. A adição de objectos a uma *quadtree* é possível pela utilização do método `put()`. Este método aceita como parâmetros as dimensões da caixa envolvente associada ao objecto. Um nó define um número máximo de objectos que aceita. Assim que esse número é ultrapassado, o nó divide a sua área em 4 quadrantes iguais e atribui-os a 4 novos sub-nós, também eles instâncias de `QuadTreeNode`. Todos os seus objectos são repartidos pelos seus sub-nós em função da sua caixa envolvente: um objecto é entregue a todos os sub-nós cujo quadrante intercepte a sua caixa envolvente. Os nós definem um número máximo de sub-divisões para evitar situações de sub-divisão

[18] A descrição que se apresenta refere-se a uma *quadtree* cujos nós se subdividem em quadrantes iguais.

[19] Expressão traduzida do inglês *bounding box* que designa o menor rectângulo capaz de envolver completamente uma representação gráfica.

infinita (como seria o caso da caixa envolvente de um objecto ser igual ao quadrante de um nó). A adição de novos objectos a um nó dividido resulta na sua entrega aos sub-nos correspondentes, seguindo os mesmos princípios já enunciados.

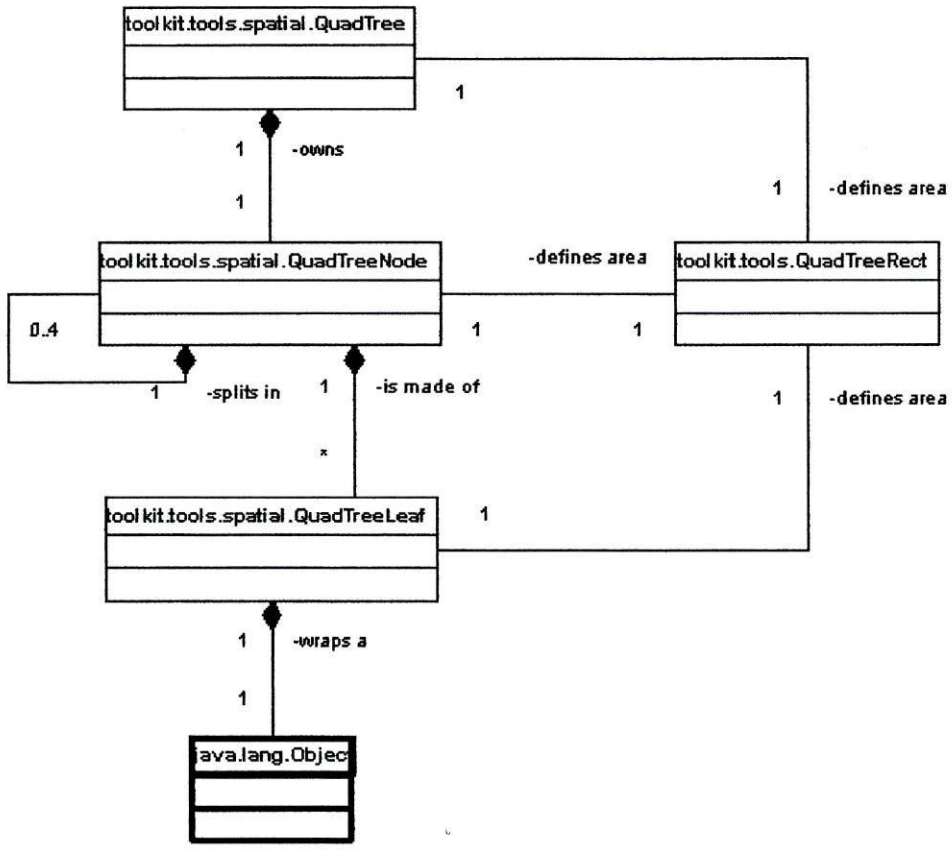


Figura 6.18: Diagrama de classes do pacote toolkit.tools.spatial — quadtrees

A pesquisa de objectos na *quadtree* restringe-se a devolver os objectos de todos os quadrantes que verifiquem o critério de pesquisa. Uma vez que é possível dois nós diferentes manterem uma referência para o mesmo objecto, o resultado da pesquisa é apresentado numa instância de `java.util.Set`, que garante a inexistência de repetições.

6.7 Implementações Standard

As implementações *standard* do modelo de componentes gráficos são disponibilizadas pelo pacote `toolkit.standard`.

6.7.1 Contexto

O modelo apresentado no pacote `toolkit.base` define uma estrutura abstracta para o desenvolvimento de componentes gráficos baseada em interfaces. O papel do pacote `toolkit.standard` é do fornecer implementações de referência desses interfaces que tenham interesse para o desenvolvimento do resto das funcionalidades disponibilizadas pela biblioteca.

Como tal, apresentam-se implementações concretas de ferramentas, figuras e contentores que disponibilizam características próprias de aplicações SCADA, como são a possibilidade de manipulação de imagem (ampliações e arrastamentos), suporte para animações e definição de camadas de visualização.

6.7.2 Descrição do Problema

O pacote deve implementar quatro tipos de ferramentas que actuam sobre instâncias do componente visualizador: ferramenta de ampliação, ferramenta de arrastamento, ferramenta de selecção de figuras e ferramenta de gestão de interactores. A ferramenta de ampliação permite alterar o nível de ampliação das figuras representadas no visualizador. A ferramenta de arrastamento permite mover a área de visualização de acordo com o posicionamento do ponteiro do rato. A ferramenta de selecção permite realçar figuras que são seleccionadas através do rato. A ferramenta de gestão de interactores permite aglutinar um conjunto de interactores que são activados em função de combinações de teclas específicas. Todas as outras ferramentas devem dispôr de uma combinação de teclas própria, de modo a permitir a sua integração nesta ferramenta.

Pretende-se a implementação de três tipos de figuras para avaliação das funcionalidades do componente visualizador: figuras geométricas, figuras baseadas em componentes Swing e figuras geométricas rotativas. Uma figura geométrica rotativa é uma especialização de uma figura geométrica, que roda em torno do seu eixo.

O pacote deve ainda fornecer uma implementação de referência para um contentor de figuras. Esta implementação será denominada por camada. Uma camada deve utilizar os mecanismos de gestão espacial disponibilizados no pacote `toolkit.tools.spatial` e deve ter níveis de visibilidade, que são função do valor corrente de ampliação do componente visualizador associado. Um nível de visibilidade define um valor máximo e um valor mínimo de ampliação para os quais a camada deve desenhar as figuras que controla. Deve ser possível ligar ou desligar o nível de visibilidade global de uma camada. Quando desligado, a camada não deve desenhar qualquer das suas figuras. No caso de estar ligado, a camada desenha, ou não, as figuras em função do nível de visibilidade. Uma camada deve oferecer suporte para figuras animadas e deve implementar um sistema de gestão de figuras intermitentes[20].

Várias camadas poderão associar-se numa multi-camada, ela própria um contentor de

[20]Este efeito consiste em tornar uma figura visível ou invisível alternadamente.

figuras. Uma multi-camada garante a ordem de desenho das suas camadas e fornece métodos para acesso externo às camadas que controla.

6.7.3 Descrição Funcional

A figura 6.19 apresenta o diagrama de classes onde são apresentadas as ferramentas implementadas neste pacote. A figura é suficientemente elucidativa das competências de cada classe. O processamento comum a este tipo de ferramentas gráficas consiste na actualização da transformada de desenho do componente visualizador, com vista à alteração das suas características em função do tipo de funcionalidade que cada ferramenta implementa.

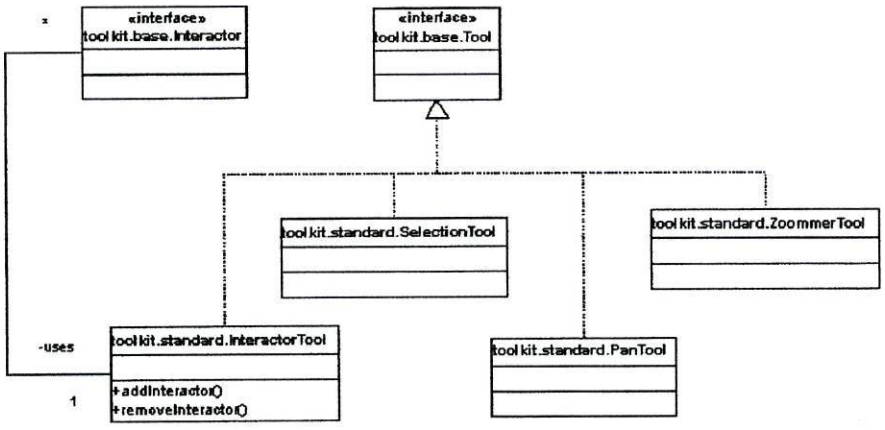


Figura 6.19: Diagrama de classes do pacote toolkit.standard — ferramentas

No diagrama 6.20 apresentam-se as figuras implementadas neste pacote. A classe abstracta `AbstractFigure` define um conjunto de características comuns a todas as figuras, deixando às suas sub-classes apenas as tarefas associadas com a sua representação gráfica. Esta classe contém também lógica que permite que as suas sub-classes não tenham que se preocupar com o facto de poderem ser variáveis ou não. Esta lógica assenta na manipulação interna de uma instância de `java.awt.geom.AffineTransform`, que no caso da figura não ser variável, mantém características inversas às da transformada correntemente utilizada pelo visualizador.

A classe `ShapeFigure` apresenta uma figura que sabe representar instâncias de `java.awt.Shape` no visualizador. Desta classe deriva a classe `ShapeRotatorFigure` que proporciona um efeito de animação, por utilização do mecanismo de tarefas temporizadas acessível através do contentor de figuras. A classe `JFigure` implementa uma figura que sabe representar componentes *Swing*, ou seja, componentes gráficos baseados na classe `javax.swing.JComponent`[21].

[21] A implementação desta classe não foi totalmente conseguida na medida em que a utilização deste tipo de figuras interfere com o sistema de gestão de eventos gráficos definido anteriormente. Tal

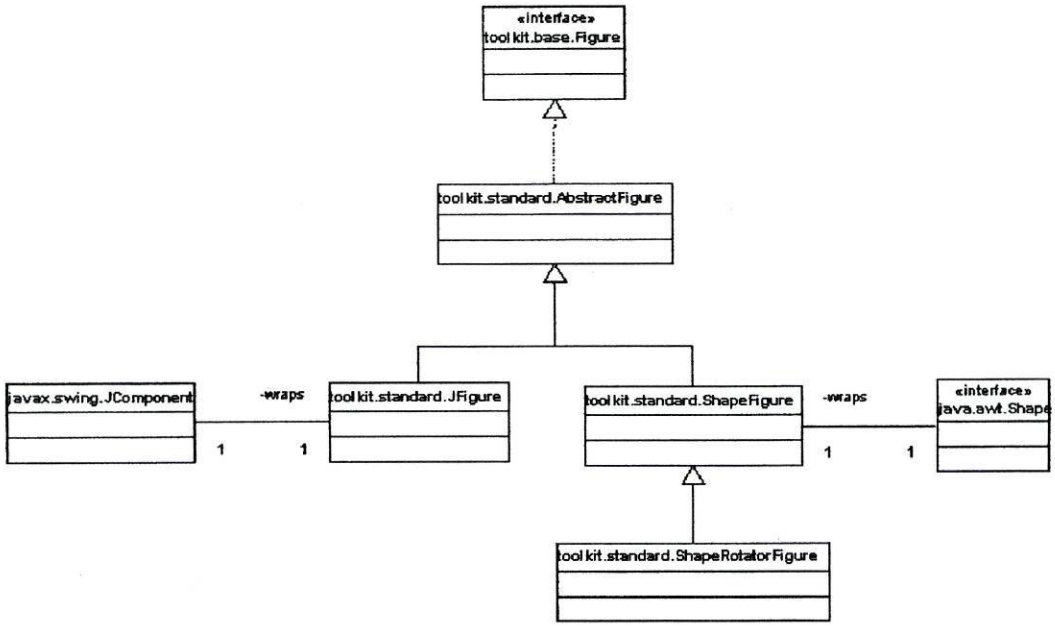


Figura 6.20: Diagrama de classes do pacote toolkit.standard — figuras

Uma nota especial para a utilização de componentes *Swing* em vez de componentes AWT (`java.awt.Component`). O uso de componentes *Swing* permite obter um aspecto gráfico uniforme para aplicações Java, independentemente da plataforma de *run-time* onde correm. Enquanto que os componentes AWT baseiam-se na associação a componentes nativos de cada sistema gráfico (*widgets*), o *Swing* não o faz. Em vez disso, socorre-se da utilização directa de primitivas gráficas para a construção dos seus componentes, garantindo a sua uniforme representação em qualquer sistema gráfico.

Finalmente, na figura 6.21 apresenta-se o diagrama de classes do modelo de camadas. Uma camada é implementada pela classe `Layer`, que mantém uma lista de figuras de tamanho fixo e uma instância de `toolkit.spatial.Quadtree` para gestão espacial de figuras variáveis. A razão de utilizar modelos de gestão diferentes é que as figuras de tamanho fixo, de facto, variam de tamanho na proporção inversa ao valor da ampliação do contexto gráfico do visualizador, dando a sensação de manter uma dimensão constante. Este factor torna impraticável a utilização de uma *quadtree* já que implicaria a remoção e adição destas figuras para cada novo valor de ampliação, levando a um inevitável atraso no processamento.

Uma camada mantém uma lista de intervalos de visibilidade definidos por instâncias de `DeltaScale`. Durante o processo de desenho, a camada certifica-se de que o nível de ampliação corrente se encontra dentro de um dos seus intervalos de visibilidade, para assim desenhar as figuras. Caso contrário, nenhuma figura é desenhada. Uma camada permite alterar o seu estado de visibilidade global (método `setVisible()`)

acontece porque o modelo de eventos do AWT procede à entrega de eventos directamente aos objectos gráficos *Swing* e depois aos seus contentores (no caso o `CanvasComponent`). É possível ultrapassar esta situação, mas tal não foi ainda implementado.

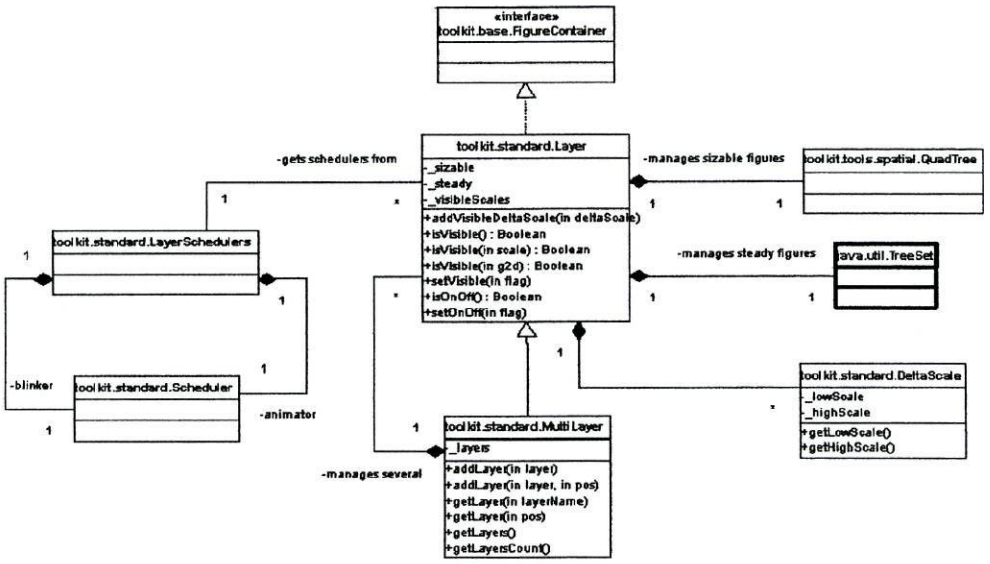


Figura 6.21: Diagrama de classes do pacote toolkit.standard — camadas

e a sua estratégia de visibilidade global, isto é, se permite a alteração desse estado (método `setOnOff()`). Uma camada permite ser interrogada acerca da sua estratégia de visibilidade global (método `isOnOff()`), do seu estado de visibilidade global e do seu estado de visibilidade em função do seu valor corrente de ampliação (métodos `isVisible()`).

Uma camada permite a obtenção de figuras em função de pontos e geometrias de conversão; no caso duma pesquisa por um ponto, a camada retorna o conjunto de todas as figuras contidas no menor quadrante que contém o ponto; no caso duma pesquisa por geometria, a camada retorna todas as figuras cujo quadrante intercepta a geometria.

Uma camada dispõe de dois tipos de tarefas temporizadas, disponibilizadas pelo *singleton* `LayerSchedulers`, nas quais admite registo de figuras: uma tarefa para gestão de animações e outra para gestão de comportamento intermitente. Neste último, o simples registo de figuras é suficiente para as tornar intermitentes.

A classe `MultiLayer` implementa o conceito de multi-camada, permitindo a adição e remoção de camadas. Esta classe é uma especialização da classe `Layer`, dispondo assim de todas as suas funcionalidades. O processo de desenho de um `MultiLayer` resulta no pedido ordenado de desenho de cada uma das suas camadas. Deste modo, as camadas superiores podem sobrepor zonas já desenhadas por camadas inferiores.

6.8 Modelo de Simbologia

O modelo de simbologia encontra-se definido no pacote `toolkit.symbol`.

6.8.1 Contexto

As funcionalidades apresentadas até ao momento encerram o conjunto de ferramentas genéricas disponibilizadas pela biblioteca de funcionalidades SCADA. Apesar de proporcionarem um número interessante de facilidades, o facto é que nenhuma delas permite ainda a criação de aplicações típicas de um sistema SCADA, ou seja, ainda não foi apresentado nenhum componente especializado que possa validar a tese defendida neste trabalho.

Como tal, esta secção apresenta o projecto de desenvolvimento de um componente especializado para a visualização de diagramas sinópticos. A opção por este componente é fundamentada pelas suas características gráficas complexas, e por ser, na opinião do autor, aquele que dispõe de uma maior número de funcionalidades SCADA, nomeadamente:

- recepção de telemetria;
- execução de controlos;
- recepção de informação de alarmes;
- visualização de diagramas que representam o sistema físico controlado.

Este componente será a base do protótipo apresentado no capítulo 7.

6.8.2 Descrição do Problema

Pretende-se desenvolver um componente de visualização de diagramas sinópticos baseado num motor de simbologia. Um motor de simbologia é uma estrutura que permite a obtenção de símbolos, organizados em estilos, de acordo com regras de apresentação.

Um símbolo constitui uma entidade que sabe representar-se num contexto gráfico numa posição bem definida, podendo ser algo tão diverso como uma imagem, texto, uma geometria ou outro tipo de representação. Os símbolos podem ser animados. Um símbolo pode ser utilizado por várias figuras com vista à obtenção de várias representações do mesmo objecto, eventualmente transformado, num componente visualizador. Uma paleta de simbologia é uma especialização da paleta do modelo de componentes, que define o estado de comportamento intermitente. Uma simbologia representa o conjunto dos símbolos e paletas de simbologia disponibilizados pelo sistema.

Um estilo é constituído por um conjunto de símbolos e paletas obtidas a partir de uma simbologia. Um estilo permite obter o símbolo e a paleta associadas a um modelo, que é baseado num gestor local de atributos. Como tal, um estilo constitui um saco de regras, cujas acções são pares símbolo-paleta. As suas regras definem qual o par símbolo-paleta que deve ser atribuído em função de um determinado modelo.

Os modelos têm um atributo especial que é a sua chave. A chave de um modelo deve ser única. Os modelos são obtidos a partir de um gestor de modelos. Os modelos expõem os seus dados a figuras especializadas, denominadas por manipuladores. O mesmo modelo pode estar associado a vários manipuladores. Qualquer alteração de um modelo deve ser propagada a todos os seus manipuladores.

Um manipulador mantém uma associação com um estilo, do qual obtém o símbolo e a paleta que utiliza para se desenhar num componente visualizador. Um manipulador delega no símbolo o processamento de desenho. Um manipulador mantém uma transformada que aplica ao símbolo cada vez que precisa de ser desenhado. A alteração do modelo de um manipulador implica a obtenção do novo par símbolo-paleta correspondente.

Deve ser disponibilizada uma linguagem XML que permita a construção de documentos com simbologias que são carregadas pelo motor. O mesmo se aplica aos estilos. Deve igualmente ser possível a construção e carregamento de diagramas, baseados em multi-camadas, tendo manipuladores como figuras. Estes diagramas devem igualmente ser construídos de acordo com uma gramática XML especialmente concebida para o efeito.

O componente de simbologia é um componente especializado, que permite a visualização de diagramas sinópticos dispostos em multi-camadas. Este componente constitui uma especialização do componente visualizador que utiliza o motor de simbologia como forma de construção e gestão dos seus diagramas. Este componente depende ainda da existência de um gestor de modelos externo.

6.8.3 Descrição Funcional

A figura 6.22 apresenta o diagrama das classes do motor de simbologia ligadas à representação gráfica. Esta estrutura segue o paradigma MVC (*Model/View/Controller*) [EGV97, p 4].

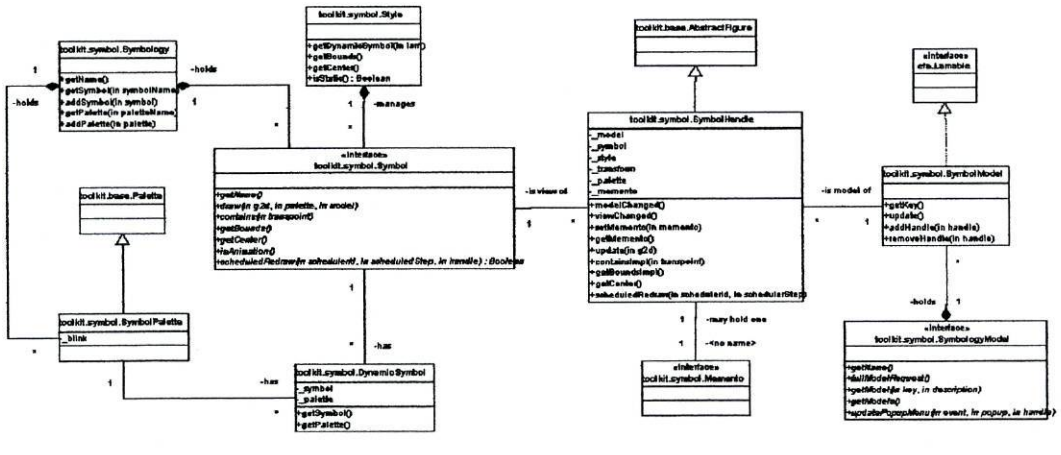


Figura 6.22: Diagrama de classes do pacote toolkit.symbol — estrutura MVC

Neste paradigma, o interface `Symbol` desempenha o papel de vista, já que é responsável pela apresentação gráfica do modelo (`SymbolModel`). Os modelos e as vistas estão associados através do controlador (`SymbolHandle`), aqui também referido como manipulador, que é uma especialização de `toolkit.standard.AbstractFigure`. Qualquer alteração no modelo é transmitida a todos os controladores (`SymbolHandle`) através do método `modelChanged()`, por forma a permitir que estes alterem as suas vistas consistentemente. De forma análoga, uma alteração na vista é reportada através do método `viewChanged()`. Vistas e modelos podem interagir entre si através do controlador.

A relação entre manipuladores (controladores) e símbolos baseia-se no padrão *Flyweight* [EGV97, p 195], onde o manipulador é o cliente e o símbolo o *flyweight* que pode ser partilhado por vários clientes. Este padrão permite a reutilização de objectos pesados por múltiplos objectos mais leves (clientes) que delegam naqueles a responsabilidade de execução de tarefas específicas, neste caso, a responsabilidade de representação gráfica. Uma vez que o símbolo pode ser partilhado, existe a possibilidade deste guardar o seu estado (gráfico) junto do seu manipulador, pela utilização de um *Memento* [EGV97, p 283]. Esta característica pode ser útil para símbolos que implementem animações.

Cada instância de `SymbolHandle` contém uma instância de `java.awt.geom.AffineTransform` que é aplicada ao contexto gráfico corrente antes de cada pedido de desenho ao símbolo correspondente. Esta característica permite que o mesmo símbolo possa ser representado de maneiras diferentes (ampliado, arrastado, rodado, etc) por vários manipuladores.

Cada manipulador mantém uma associação com uma instância de `Style` com vista à obtenção dos símbolos e das paletas, encapsuladas em instâncias de `DynamicSymbol`, que correspondem ao estado do modelo. Esta característica permite que o manipulador possa alterar a sua representação de forma dinâmica. No padrão *flyweight*, o estilo desempenha a função de *Flyweight Factory*.

O facto dos manipuladores poderem alterar a sua representação (símbolo), apresenta, contudo, problemas associados ao mecanismo de gestão espacial, na medida em que não é possível garantir a constância das suas dimensões ao longo do tempo. Como forma de ultrapassar este problema, a classe delega a responsabilidade da definição da geometria (caixa envolvente) não ao símbolo associado, mas sim ao seu estilo. Isto acontece porque o estilo, tendo conhecimento de todos os símbolos que controla, pode responder a pedidos de geometria com a menor caixa envolvente que englobe todos os seu símbolos. Assim, a resposta à execução dos métodos do manipulador `getBoundsImpl()` e `getCenter()` é delegada nos métodos `getBounds()` e `getCenter()` do estilo correspondente. Já o processamento do teste de localização, executado por invocação de `containsImpl()`, é delegado no símbolo corrente.

Os estilos podem ser do tipo estático ou dinâmico. Um estilo dinâmico apresenta as características já discutidas. Um estilo estático é um estilo que gere apenas paletes. Manipuladores baseados neste tipo de estilo mantêm uma associação constante com um único símbolo que lhes é entregue durante a sua construção. Para estes casos, os

manipuladores utilizam directamente os métodos de geometria do símbolo como forma de obtenção da caixa envolvente e da posição de desenho. Esta facilidade permite a existência de símbolos que não pertencem a qualquer simbologia, não sendo por isso partilhados entre manipuladores.

Os modelos de símbolos (`SymbolModel`) são obtidos por classes que especializam o interface `SymbologyModel`.

A figura 6.23 apresenta as classes do modelo que constituem implementações concretas de símbolos. A classe `ShapeSymbol` baseia-se em instâncias de `java.awt.geom.Shape`, a classe `ImageSymbol` representa uma imagem, a classe `LabelSymbol` permite a representação de uma porção de texto fixa, a classe `TextSymbol` permite a representação de uma mensagem de texto que é função do modelo, a classe `ImageWalker` permite animar uma imagem fazendo-a percorrer um caminho determinado e, finalmente, a classe `CompositeSymbol` que implementa o padrão *composite*, podendo, por isso, ser constituída por outras instâncias de `Symbol`.

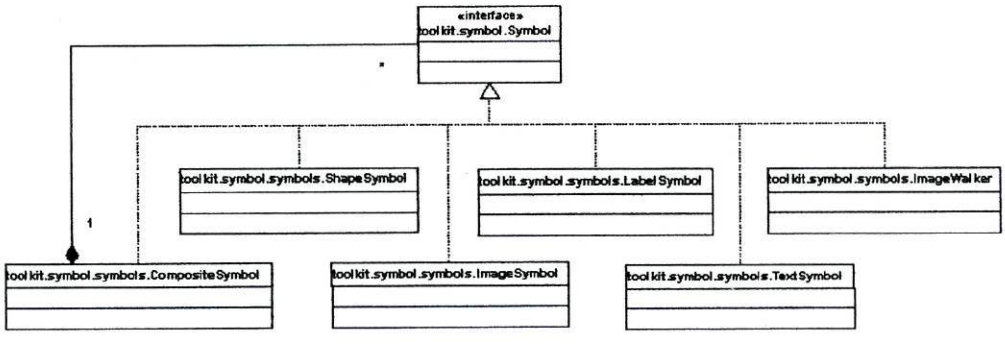


Figura 6.23: Diagrama de classes do pacote toolkit.symbol — símbolos

A figura 6.24 ilustra as classes ligadas à gestão de simbologias e de estilos. A classe `SymbologyManager` permite o carregamento de uma simbologia a partir de um URL. Esta simbologia é mantida até que um novo pedido seja efectuado. O gestor de simbologia utiliza um mecanismo de notificação baseado no sistema de eventos Java. Este sistema permite o registo de ouvintes interessados em saber eventuais alterações de simbologia. A classe `StylesManager` é um exemplo de um "ouvinte de simbologia" uma vez que tem interesse em saber da ocorrência de alterações com vista à actualização dos seus estilos. Este gestor constitui uma especialização de um motor de regras e sabe por isso carregar estilos (regras) também a partir de um URL. Também ele utiliza um mecanismo de notificação por eventos.

Na figura 6.25 apresenta-se o diagrama de classes do componente de simbologia (`SymbologyCanvas`). Este componente é uma especialização do componente visualizador (`efa.base.CanvasComponent`) que tem associações com os gestores de simbologia e de estilos. Aqui se apresenta também a classe `LayerManager` que é capaz de construir multi-camadas a partir de um ficheiro XML, definido por um URL. Esta classe implementa os interfaces `SymbologyListener` e `StylesListener` por forma a

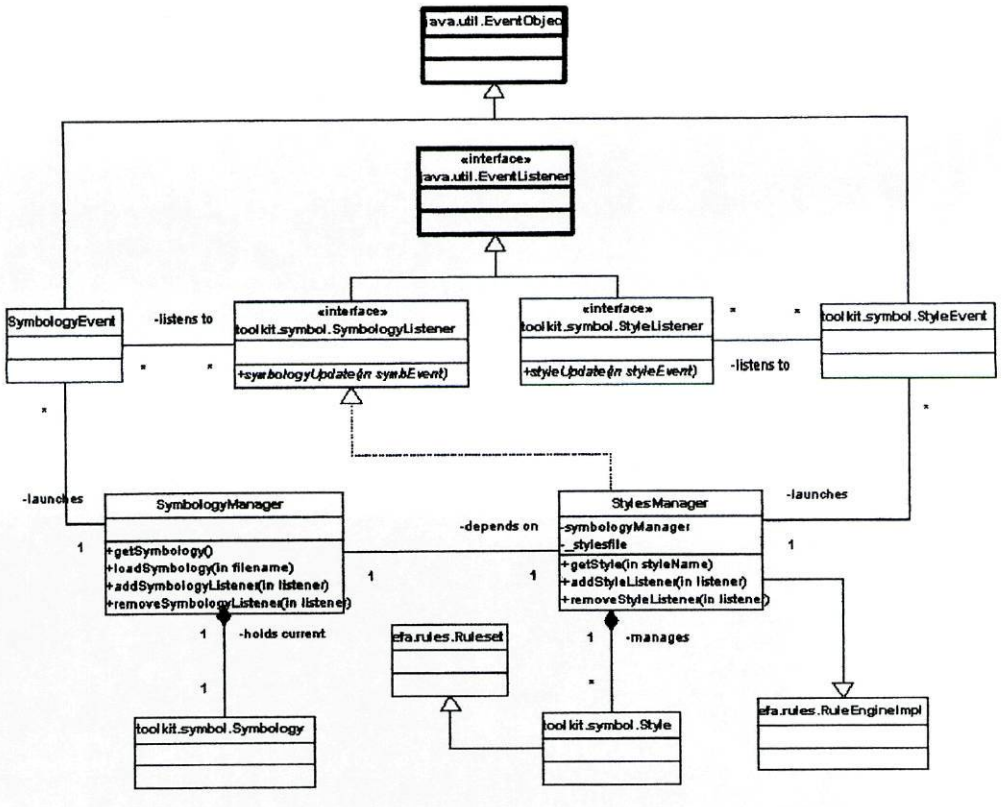


Figura 6.24: Diagrama de classes do pacote toolkit.symbol — gestores

garantir que dispõe de toda a informação necessária para a construção de diagramas. O componente de simbologia mantém ainda uma associação com um gestor de modelos externo (`SymbologyModel`);

6.8.4 Gramáticas XML para Motor de Simbologia

O apêndice D apresenta o esquema XML que contém as definições para a construção de simbologias (`symbology.xsd`). A figura 6.26 ilustra o esquema referido através da sua representação gráfica.

Uma simbologia (elemento `<symbology>`) é constituída por símbolos (elemento `<symbol>`) e/ou paletes (elemento `<palette>`). Uma paleta é identificada por um nome e contém atributos que identificam quais os elementos gráficos por ela definidos, como a côr de fundo, a côr do traço, o tipo de traço, o comportamento intermitente, o tipo de fonte, etc.

Os símbolos podem igualmente ser identificados por um nome e podem ser constituídos por dois tipos de elementos: elementos que mapeiam directamente em implementações concretas do interface `Symbol` (casos dos elementos `<image>`, `<text>`, `<label>` e `<imageWalker>`) ou elementos que codificam informação vectorial (`<circle>`, `<polygon>`, `<polyline>`, `<rect>`, `<ellipse>` e `<line>`). Cada um dos elementos vectoriais é

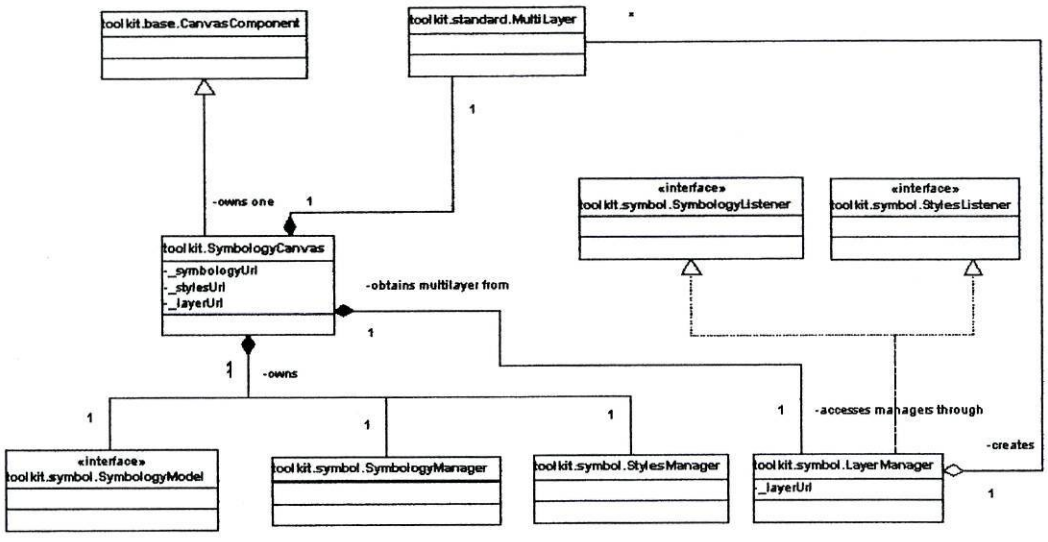


Figura 6.25: Diagrama de classes do pacote toolkit.symbol — componente de simbologia

traduzido internamente numa instância do símbolo `ShapeSymbol`, de acordo com as características do elemento. Um símbolo que contenha mais do que um sub-elemento é traduzido numa instância de `CompositeSymbol` composto pelos símbolos correspondentes. Cada um dos sub-elementos de `<symbol>` tem atributos próprios e atributos genéricos. Os atributos próprios são atributos que definem características específicas dos elementos. Por exemplo, o elemento `<image>` define o atributo `imageUrl` que é utilizado para indicar o URL da imagem a ser carregada. Os atributos genéricos são atributos comuns a todos os sub-elementos de `<symbol>` e definem a transformada que deve ser aplicada às definições do símbolo. Estes atributos são: `tx` (deslocamento no eixo dos x), `ty` (deslocamento no eixo dos y), `tsx` (ampliação no eixo dos x), `tsy` (ampliação no eixo dos y) e `tr` (ângulo de rotação, em graus).

A interpretação de documentos de simbologia, e consequente construção dos símbolos e paletas, é proporcionada pelas classes `DOMSymbologyParser` e `DOMSymbologyParserEngine`, ambas baseadas na API DOM para processamento de XML.

O modelo de simbologia permite o desenvolvimento de novos tipos de símbolos que podem facilmente ser integrados nesta definição (actualização do esquema `symbology.xsd`). Nesse caso, dever-se-á proceder à alteração da classe `DOMSymbologyParserEngine` com vista à introdução da lógica necessária à construção dos novos tipos de símbolos.

No apêndice D é apresentado ainda um outro esquema XML para a definição de diagramas estruturados em multi-camadas (`multilayer.xsd`). A figura 6.27 ilustra o referido esquema.

Neste esquema, uma multi-camada é definida pelo elemento `<multilayer>` que contém o conjunto dos atributos necessários à sua caracterização: O nome (`name`), a paleta por omissão (`palette`), a caixa envolvente (atributos `x`, `y`, `w` e `h`), a política de visualização

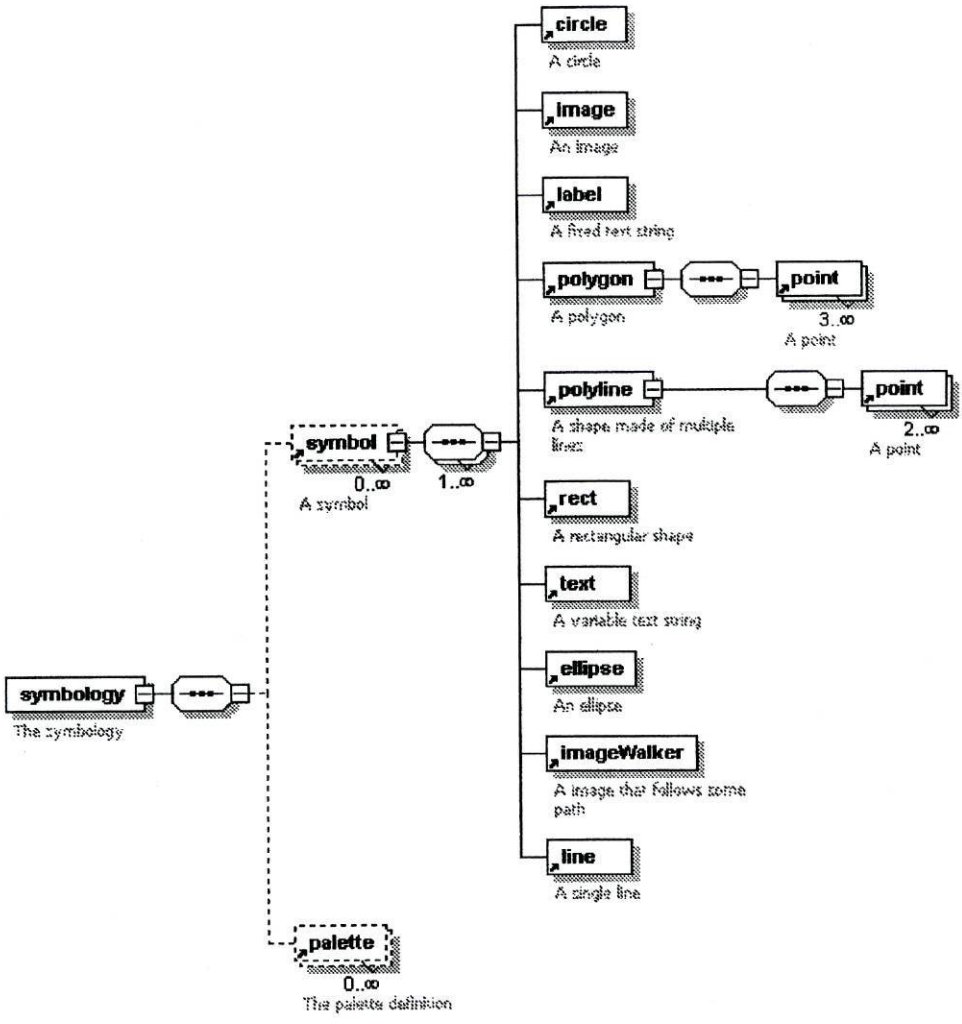


Figura 6.26: Representação gráfica do esquema XML de simbologia

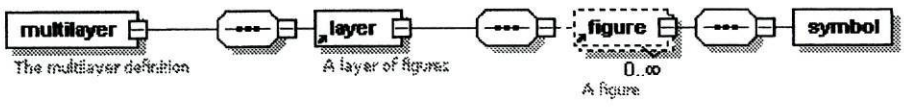


Figura 6.27: Representação gráfica do esquema XML de diagramas

global (`isOnOff`) e um só nível de visualização (atributos `low` e `high`). A opção de utilizar um só nível de visualização teve em vista a simplificação da interpretação dos diagramas.

Um elemento `<multilayer>` é constituído por vários elementos do tipo `<layer>` (camada) que mapeiam em instâncias de `efa.standard.Layer` e que têm o mesmo tipo de atributos que a multi-camada. Uma camada é por sua vez constituída por figuras (elemento `<figure>`). Uma figura é caracterizada pelos atributos `key` (chave do modelo), `model` (texto que identifica uma característica do modelo e é utilizada, juntamente com a chave para a obtenção de uma instância daquele), `<sizable>` (se a figura é variável ou não), `style` (o estilo) e os atributos `tx`, `ty`, `tsx`, `tsy` e `tr`, atrás referidos, para a construção de uma transformada. Esta é a transformada que a figura (manipulador) aplica ao contexto gráfico antes do desenho do símbolo correspondente.

O modelo permite ainda a definição local de um símbolo que deve ser utilizado pela figura (elemento `<symbol>`). Este símbolo segue as mesmas definições dos símbolos de `symbology.xsd` e será utilizado no caso no estilo ser do tipo estático.

O pacote `toolkit.symbol.xml` disponibiliza as classes `DOMLayerParser` e `DOMLayerParserEngine` para a interpretação de diagramas. Também estas classes são baseadas na API DOM de processamento de XML.

6.9 Sumário

Neste capítulo foi apresentado o projecto de *software* que orientou o desenvolvimento da biblioteca de funcionalidades SCADA.

O objectivo principal incidiu na apresentação clara dos objectivos e na definição dos contratos específicos de cada funcionalidade. O restante resumiu-se à descrição técnica dos modelos pela utilização de uma metodologia comum, no caso, a notação UML.

Embora os modelos se constituíssem como problemas ortogonais, à partida, o projecto procurou enquadrá-los numa estrutura lógica, onde cada modelo ia servindo de base à construção de modelos mais específicos, que culminaram com o desenvolvimento de um componente SCADA especializado — o componente de simbologia.

Do projecto transpareceu também uma clara opção pela utilização do XML como linguagem de eleição para a definição de gramáticas externas, que por força das APIs de processamento disponíveis, puderam ser facilmente assimiladas pelas funcionalidades implementadas. Assim foi com o modelo de eventos (ainda no capítulo 5), assim foi com o modelo de expressões, assim foi com o modelo de regras e assim foi também com o modelo de simbologia.

Capítulo 7

Protótipo — ScadaEngine

Este capítulo apresenta o projecto de *software* associado à implementação de um protótipo que permite validar o trabalho desenvolvido — o *ScadaEngine*.

7.1 Contexto

A figura 7.1 apresenta o contexto de utilização do ScadaEngine.

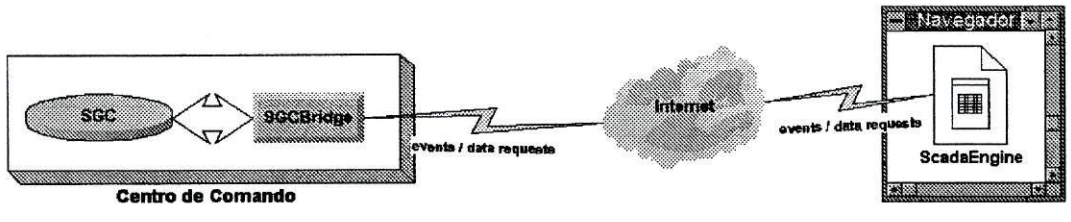


Figura 7.1: Contexto de utilização do ScadaEngine

O *ScadaEngine* é uma aplicação que pode ser visualizada num navegador pela sua inclusão numa *applet* que é descarregada remotamente. No navegador, a aplicação permite o carregamento de diagramas sinópticos para visualização local. Estes diagramas serão actualizados ao longo do tempo por via da recepção de eventos provenientes do centro de comando. Deste modo, o *ScadaEngine* incorpora um ou vários componentes que se registam para publicação e/ou subscrição dos tópicos correspondentes.

No centro de comando existe um componente especial, *SGCBridge*, capaz de interagir com as aplicações do SGC, por um lado, e com o *ScadaEngine*, por outro. No processo de comunicação com o *ScadaEngine*, o *SGCBridge* tem as seguintes responsabilidades:

- Envio dos eventos que correspondem a alterações de estado de entidades analógicas ou digitais;
- Recepção de pedidos de execução de comandos do tipo controlo (digital) ou alteração de valor (analógico).

A comunicação entre estes dois componentes é assegurada pela arquitectura de componentes descrita no capítulo 5.

A figura 7.2 apresenta o diagrama de pacotes associado ao projecto de *software* do ScadaEngine. Nela se verifica a utilização das duas grandes estrutura desenvolvidas neste trabalho: o *framework* de componentes e a biblioteca de funcionalidades SCADA.

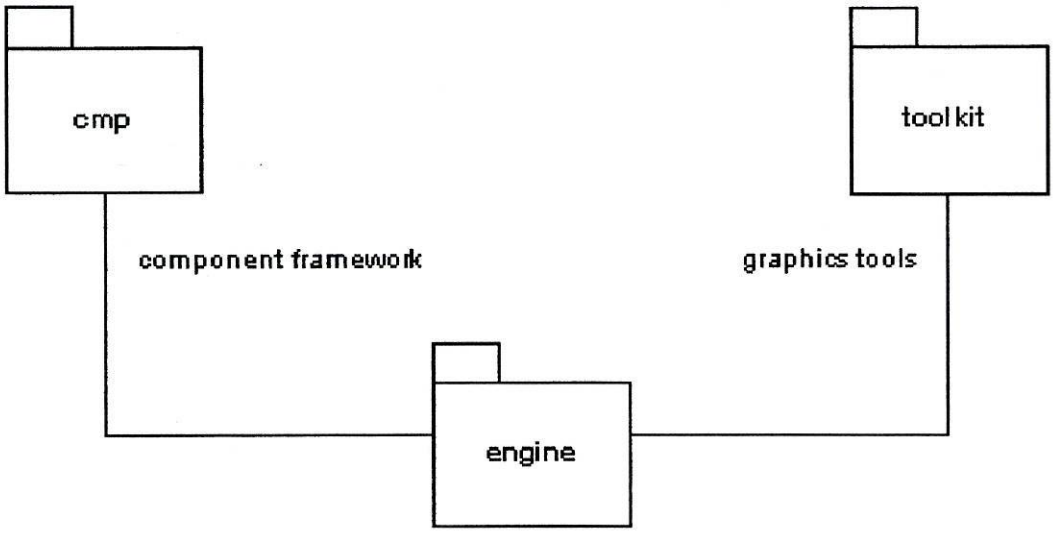


Figura 7.2: Estrutura de pacotes do ScadaEngine

7.2 Descrição do Problema

O *ScadaEngine* é uma aplicação para a visualização de diagramas sinópticos, podendo ser utilizada numa *applet* Java ou como uma aplicação *stand-alone.*

A vista principal da aplicação deve ser constituída por um navegador de diagramas, uma caixa de ferramentas, um gestor de camadas (ligar/desligar) e por uma área de visualização de diagramas.

O carregamento de um sinóptico é efectuado pelo utilizador através da indicação do URL de um ficheiro XML contendo as definições do diagrama: simbologia, estilo e o diagrama propriamente dito. Deve ser possível o carregamento de vários sinópticos que vão sendo mostrados em janelas distintas na área de visualização. A navegação

entre diagramas é garantida pelo navegador de diagramas, que apresenta a lista de todos os diagramas presentes na aplicação.

O gestor de camadas, por seu turno, apresenta todas as camadas do diagrama corrente (seleccionado) e deve permitir alterar o estado global de visualização de cada camada.

Os diagramas podem basear-se em entidades analógicas ou digitais, cujos modelos deverão ser providenciados pela aplicação. Pode existir ainda um outro tipo de modelo para figuras que permitam o carregamento de novos diagramas.

7.3 Descrição Funcional

A figura 7.3 apresenta o diagrama de classes do *ScadaEngine*.

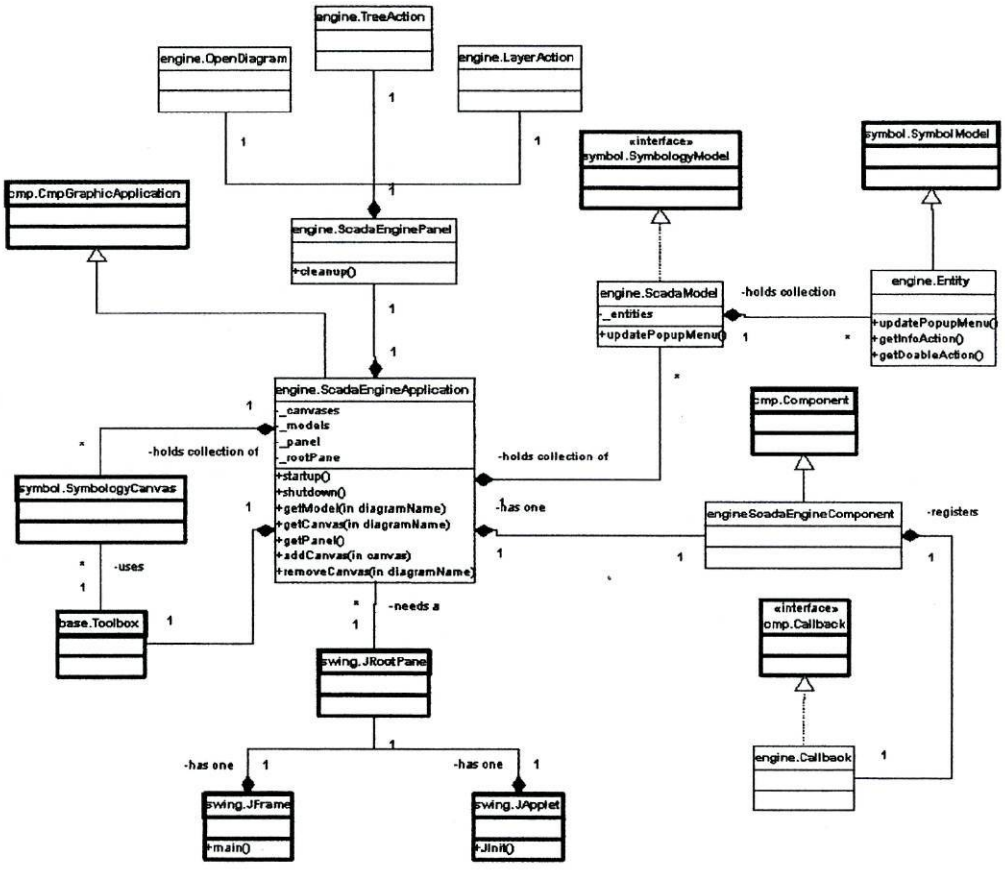


Figura 7.3: Diagrama de classes do ScadaEngine

A classe `ScadaEngineApplication` constitui o "coração" da aplicação, na medida em que funciona como ponto de convergência entre todas as classes do modelo. Esta classe deriva de `cmp.CmpGraphicApplication` e mantém uma associação para a instância de `javax.swing.JavaRootPane` pertencente ao contentor global de visualização, seja

ele uma `javax.swing.JFrame` ou uma `javax.swing.JApplet`. É na instância de `javax.swing.JavaRootPane` que serão desenhados todos os objectos gráficos da aplicação. Este abordagem permite o desenvolvimento da aplicação, independentemente do facto dela poder ser utilizada numa *applet* ou como aplicação *stand-alone.*

A classe `ScadaEnginePanel` constitui o painel principal da aplicação que se subdivide internamente nas diferente áreas de visualização. Esta classe é composta por três outras classes que implementam a lógica própria de cada uma dessa áreas: `OpenDiagram`, `TreeAction` e `LayerAction`.

Ainda do ponto de vista gráfico, a aplicação contém uma instância de `base.Toolbox` que é partilhada por todas as instâncias de `symbol.SymbologyCanvas` existentes, as quais são responsáveis pela visualização dos vários diagramas carregados.

A aplicação inclui ainda dois outros conjuntos de classes: as classes que implementam os modelos das figuras visualizadas nos diagramas e as classes que permitem a sua integração com a arquitectura de componentes.

Do primeiro conjunto de classes fazem parte as classes `ScadaModel` e `Entity`. A classe `ScadaModel` é uma implementação do interface `SymbologyModel`, sendo por isso a entidade responsável pela criação e gestão dos modelos (`SymbolModel`) dos manipuladores utilizados pela simbologia. Este modelos são implementados por especializações da classe `Entity`, e dão corpo aos conceitos de entidade analógica (classe `Measure`), entidade digital (classe `Digital`) e entidade de ligação entre diagramas (classe `Link`). Todas estas classes encontram-se ilustradas no diagrama apresentado pela figura 7.4.

O segundo conjunto de classes diz respeito à infra-estrutura de componentes que permitem a interligação da aplicação com aplicações residentes no centro de comando. Para tal, o modelo define as classes `ScadaEngineComponent` e `Callback`. A implementação do componente restringe-se ao registo de tópicos para a troca de eventos com eventuais aplicações residentes no centro de comando, com vista à obtenção do estado actual das entidades analógicas e digitais. O papel da *callback* é receber esses eventos e comunicalos ao modelo da aplicação (`SymbologyModel`) com vista à actualização da informação gráfica correspondente. A secção 7.4 apresenta a estrutura dos eventos utilizados pela aplicação.

Voltando à figura 7.4, nela são apresentadas as classes que permitem a execução de acções a partir do diagrama. Estas acções são especializações do interface `javax.swing.AbstractAction` podendo, por isso, servir como opções de menus do tipo *popup*, tais como os disponibilizados pelo sistema. A classe `Control` permite a execução de um controlo sobre uma entidade digital e a classe `SetPoint` permite, de forma análoga, a atribuição de valor a uma medida. A lógica de processamento destas classes baseia-se no envio de eventos (ou pedidos de dados) para as aplicações do centro de comando responsáveis pela execução das acções respectivas. A classe `CanvasOpener` associa-se, por seu turno, a uma instância de `Link` e permite o carregamento de um novo diagrama a partir do URL disponibilizado por aquele. Finalmente, a classe `Info` permite a visualização de um painel com informação da entidade seleccionada.

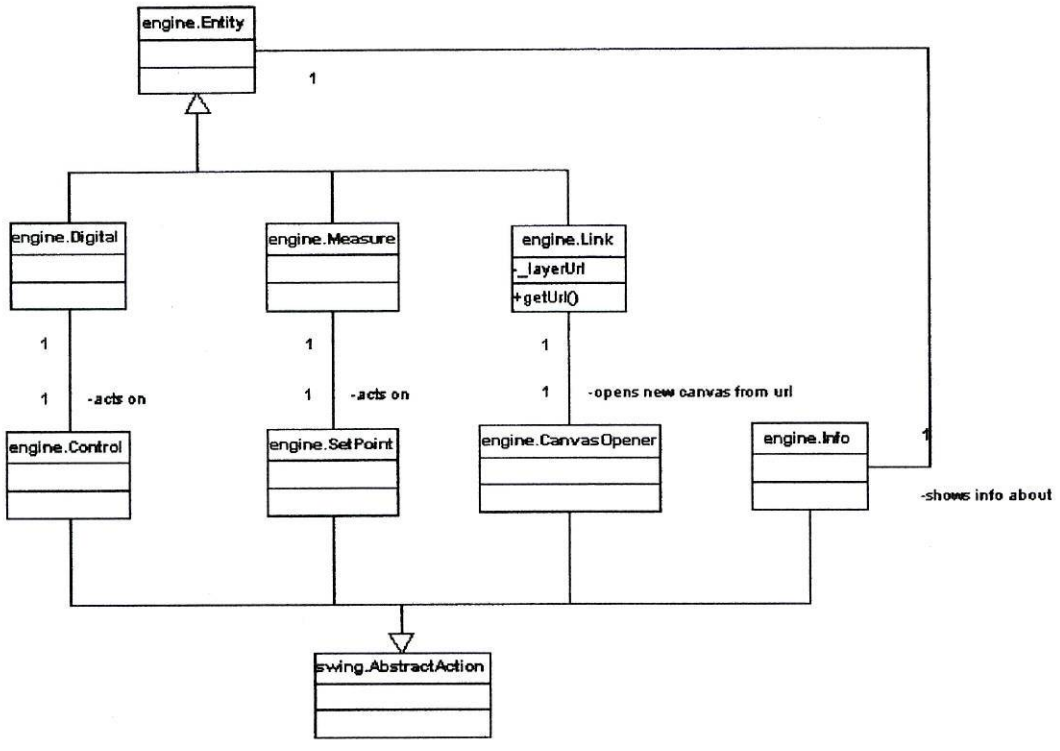


Figura 7.4: Diagrama de classes do ScadaEngine — modelos

7.4 Configurações da Aplicação

Esta secção apresenta o conjunto de informação externa que garante a usabilidade da aplicação. Esta informação agrupa-se em três tipos que serão apresentados em detalhe: metamodelo de eventos, configuração de diagramas e apresentação em HTML.

7.4.1 Metamodelo de Eventos

O metamodelo de eventos é o elemento da arquitectura de componentes onde se estabelecem as definições acerca da estrutura dos eventos trocados entre componentes, e da sua associação com tópicos (ver capítulo 5).

O metamodelo ilustrado na figura 7.5 apresenta somente a informação relevante para esta aplicação. Nele se define o tópico `Scada`, onde pontificam dois eventos: o evento `Analog`, utilizado para a transmissão de informação de entidades analógicas, e o evento `Digital`, que serve a transmissão de informação de entidades digitais. Cada um dos eventos é caracterizado por um identificador (atributo `Id`), o estado de alarme (atributo `Alarm`) e o valor da entidade, que é modelizado pelos atributos `Value` e `On`, referente a entidades analógicas e digitais, respectivamente.

Deste modo, a recepção de um evento `Analog`, por parte do *ScadaEngine*, resultará na alteração gráfica da figura correspondente, em função dos campos `Value` e `Alarm`. O

```
<?xml version="1.0" encoding="UTF-8"?>
<metamodel>
  <topic name="Scada">
    <event name="Analog">
      <attribute name="Id" type="EFAStringValue"/>
      <attribute name="Value" type="EFALongValue"/>
      <attribute name="Alarm" type="EFABoolValue"/>
    </event>
    <event name="Digitals">
      <attribute name="Id" type="EFAStringValue"/>
      <attribute name="On" type="EFABoolValue"/>
      <attribute name="Alarm" type="EFABoolValue"/>
    </event>
  </topic>
</metamodel>
```

Figura 7.5: Metamodelo de eventos para o ScadaEngine

mesmo se aplica ao evento `Digital`.

Este mesmo metamodelo é suficiente para a transmissão de pedidos de execução de comandos da seguinte forma: a execução de um comando digital resulta no envio de um pedido de dados (*data request*) preenchido com um evento `Digital` onde se indica a identificação da entidade que deve ser alterada (atributo `Id`) e o estado pretendido (atributo `On`). Como identificação do pedido, usar-se-á a *string* "*Command*".

No centro de comando existirá um componente sensível a este pedido de dados (O *SGCBridge*) que actuará em conformidade, eventualmente pela execução de um pedido de características semelhantes ao gestor de telecontrolos do SGC (ver capítulo 2).

A execução de uma actualização de valor (*setpoint*) processar-se-á de forma semelhante.

7.4.2 Configuração de Diagramas

Os diagramas são carregados por indicação de um URL contendo informação acerca da localização dos ficheiros de simbologia, estilos e de construção do diagrama. Esta informação é descrita num documento XML com uma estrutura muito simples. A figura 7.6 ilustra um exemplo destes documentos que descreve a localização de um diagrama para a visualização duma hipotética rede de metropolitano.

No apêndice E, apresenta-se o conteúdo dos ficheiros indicados na figura 7.6 e que foram utilizados para validação do protótipo.

```
<?xml version="1.0" encoding="UTF-8"?>
<diagram name="Metro Diagram">
  <symbology url="file:///c:/java/config/engine/symbology.xml"/>
  <styles url="file:///c:/java/config/engine/styles.xml"/>
  <layer url="file:///c:/java/config/engine/metrodiagram.xml"/>
</diagram>
```

Figura 7.6: Informação para carregamento de diagrama

7.4.3 Apresentação em HTML

A parte final do processo de configuração reside na construção duma página HTML para apresentação da *applet* contendo o *ScadaEngine*. O código para o efeito apresenta-se na figura 7.7.

```
<html>
  <head>
    <title>
      ScadaEngine
    </title>
  </head>
  <body>
    <object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" WIDTH = 700
      HEIGHT = 600 ALIGN = middle
      codebase="HTTP://java.sun.com/products/plugin/
      1.2/jinstall-12-win32.cab#Version=1,2,0,0">
      <param NAME = CODE VALUE = "pt.efacec.se.aut.engine.
      ScadaEngineApplet.class">
      <param NAME = CODEBASE VALUE = "." >
      <param NAME = NAME VALUE = "ScadaEngineApplet" >
      <param NAME = ARCHIVE VALUE = "engine.jar" >
      <param NAME = "type" VALUE = "application/x-java-applet;version=1.2">
    </object>
  </body>
</html>
```

Figura 7.7: Página HTML para o ScadaEngine

Os parâmetros da figura 7.7 descrevem as características para o carregamento da *applet*. De salientar a indicação do tipo de *plug-in* requerido pela aplicação (Java 1.2). A figura 7.8 demonstra o aspecto do final da aplicação correndo num navegador. Na figura são facilmente identificáveis as diversas áreas funcionais da aplicação: caixa de ferramentas, navegador de diagramas, gestor de camadas e área de visualização (que

contém dois diagramas carregados).

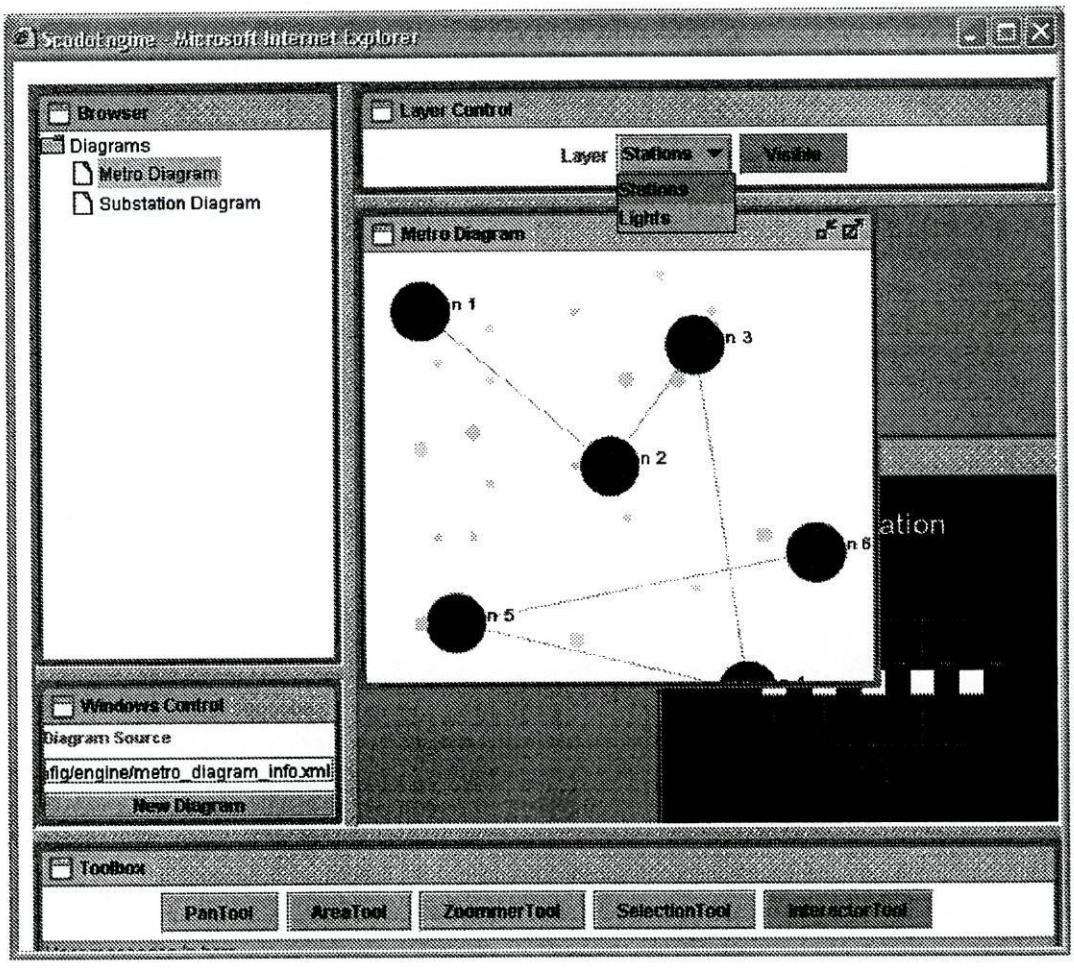


Figura 7.8: ScadaEngine — applet

7.4.4 Utilização noutros Ambientes

Embora tratando-se de um estudo sobre a viabilidade de apresentação de aplicações SCADA na *Web*, importa realçar que a utilização de uma arquitectura de componentes colaborantes, tal como a disponibilizada pelo *framework* de componentes, não restringe a aplicabilidade das soluções apenas ao ambiente *Web*. De facto, a natureza multi-protocolo proporcionada pelo modelo de comunicadores (ver capítulo 5), garante que as aplicações desenvolvidas sobre este *framework* possam rapidamente ser reutilizadas em circunstâncias de comunicação diferentes, reduzindo o esforço de desenvolvimento de soluções específicas a cada contexto de utilização.

A capacidade de reutilização de implementações constitui um factor extremamente interessante, no âmbito da indústria do *software*, revestindo-se como uma vantagem competitiva importante.

A figura 7.9 apresenta o mesmo protótipo, agora a funcionar como aplicação *stand-alone*, a correr em Linux.

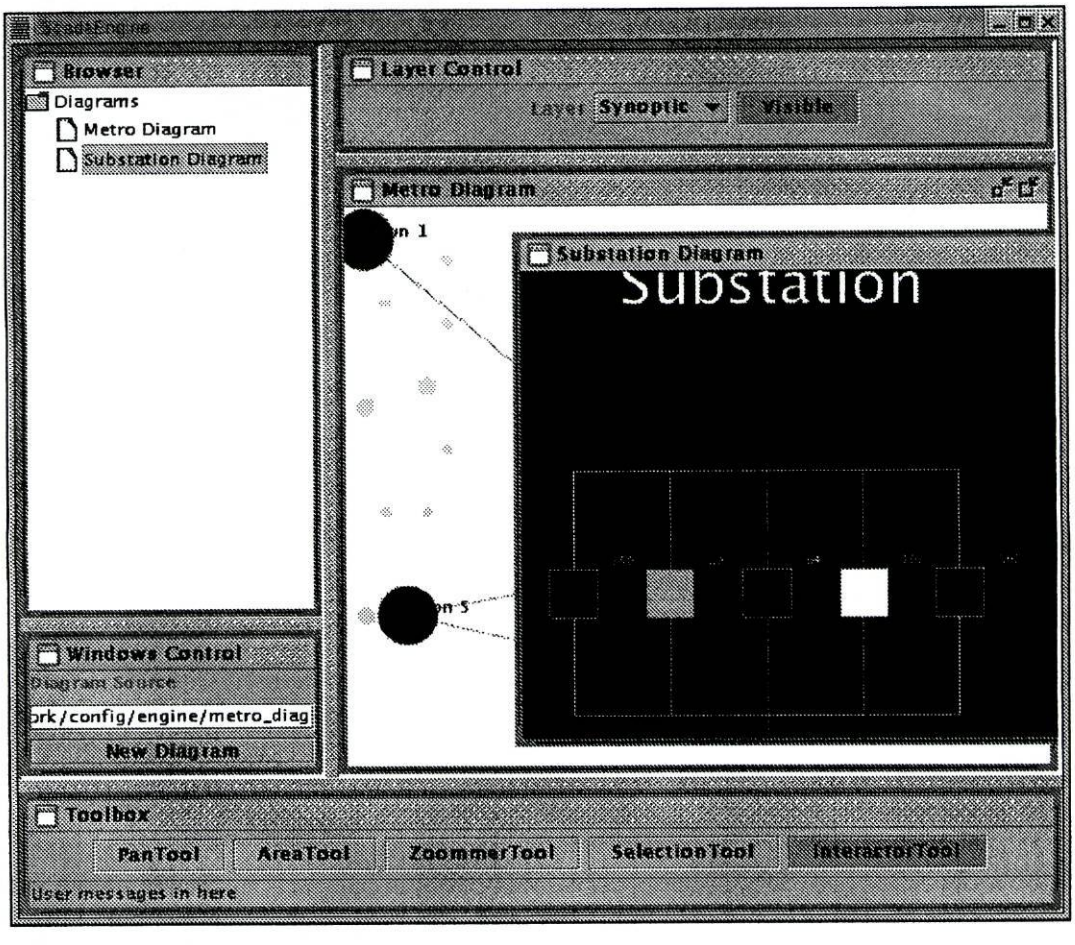


Figura 7.9: ScadaEngine — aplicação stand-alone

7.5 Configurações Gerais

O sucesso da utilização da aplicação depende da existência de algumas bibliotecas exteriores ao ambiente Java *standard*. Estas são as bibliotecas para o processamento de XML, e a própria biblioteca de funcionalidades SCADA desenvolvida neste trabalho. O tamanho total destas bibliotecas é de sensivelmente 3,2 MB (0.2 MB SCADA + 3 MB XML).

Existem duas hipóteses de utilização, a saber:

1. Proceder ao descarregamento das bibliotecas juntamente com a *applet*;
2. Proceder à instalação das mesmas no dispositivo de acesso;

Embora a primeira hipótese se constitua como a mais genérica, o processo de descarregamento das bibliotecas poderá tornar-se numa tarefa penosa, principalmente para acessos através de ligações com baixa largura de banda.

Já a segunda hipótese permite reduzir o tempo de descarregamento somente à aplicação, mas implica garantir a instalação destas bibliotecas[1] em todos os dispositivos individuais de acesso, violando um dos requisitos descritos no capítulo 4. Mesmo assim, considerou-se aceitável a escolha desta opção[2]. Ao fazê-lo, o tamanho total da *applet* a visualizar resume-se a cerca de 25 kB.

7.6 Sumário

Este capítulo apresentou o projecto de *software* para a construção de uma aplicação típica SCADA, com recurso à utilização da biblioteca de funcionalidades SCADA e ao *framework* de componentes Java. A disponibilização destas ferramentas permitiu limitar a construção da aplicação à tarefa de interligação das diversas funcionalidades. Desta forma, o protótipo permite verificar a tese apresentada, onde se defende ser possível a construção de aplicações SCADA pela utilização de componentes gráficos especializados.

Foi também demonstrado a relevância do componente de simbologia como um componente especializado para este tipo de aplicações.

[1] O processo de instalação das bibliotecas é relativamente simples e consiste em colocá-las no sub-directório `jre/bin/ext` do directório onde se encontra o *run-time* Java.

[2] As bibliotecas de processamento de XML passaram a integrar a distribuição *standard* do *Java Development Kit* da *Sun* a partir da versão 1.4.0, o que vem resolver, em parte, esta questão.

Capítulo 8

Testes e Análise de Resultados

Este capítulo apresenta o conjunto de actividades que permitirão validar o resultado do trabalho desenvolvido.

Em primeiro lugar, definem-se as métricas que serão utilizadas como forma de avaliação dos resultados. De seguida apresentam-se dois tipos de análise: análise directa das funcionalidades da biblioteca e análise dos tempos de arranque de uma aplicação real, no caso o *ScadaEngine*.

8.1 Métricas de Avaliação

A dificuldade de elaboração de um sistema de testes não reside no processo de execução dos testes em si, mas sim na escolha das métricas de avaliação que melhor se adaptem à natureza dos problemas versados. Para o caso presente, e tratando-se do desenvolvimento de interfaces gráficos, achou-se por bem enquadrar todos os ensaios numa perspectiva de interacção com o utilizador. Assim, cada um dos testes efectuados será analisado como se os tempos de execução correspondessem ao tempo que decorre entre o pedido de execução de uma acção, e a obtenção da resposta respectiva, por parte da aplicação.

Deste modo, é possível balizar os tempos obtidos de acordo com as métricas definidas por Robert B. Miller[1] (em 1968) e que se mantêm ainda hoje como a referência para a avaliação do grau de interactividade de aplicações que promovem o interface com o utilizador [Nie00, pp 42–46]:

[1]Estas métricas resultam dum estudo que apresentou com um paper na Fall Joint Computer Conference de 1968.

100 milissegundos — Este valor constitui o limite de tempo para o qual o utilizador sente que o sistema reage instantaneamente. Este deve ser o limite para aplicações que permitam a execução de ampliações, arrastamentos ou qualquer outro tipo de manipulação de elementos gráficos em tempo real.

1 segundo — É o limite de tempo para a manutenção do fluxo de pensamento ininterrupto, apesar do utilizador perceber que existe um atraso na resposta. Tempos superiores a 1 segundo requerem a existência de mecanismos de notificação (ao utilizador) da ocorrência de processamento (ex. barra de progresso, mensagem informativa, etc).

10 segundos — Este valor constitui o limite de tempo em que o utilizador ainda mantém a atenção na aplicação.

Para os testes efectuados, considerar-se-ão como bons os resultados obtidos dentro do limite de tempo real (100 milissegundos).

8.2 Ambiente de Testes

Os testes que irão ser apresentados foram efectuados numa máquina com as seguintes características de *software* e de *hardware*:

Processador – Intel@1GHz

Memória – 256 MB

Monitor – Matriz Activa 15"

Placa Gráfica – Silicon Motion Lynx3D com 8 MB de memória

Resolução – 1024x768

Sistema Operativo – Windows XP

***Java Runtime Environment* (JRE)** – 1.3.1

Bibliotecas de Processamento de XML – jaxp-api.jar, dom.jar, sax.jar, xalan.jar, xercesImp.jar e xsltc.jar

Todos os tempos medidos referem-se a tempos totais de execução e não a tempos de processador, podendo por isso variar em função da taxa de ocupação da máquina.

8.3 Testes à Biblioteca de Funcionalidades SCADA

Esta secção apresenta os testes de desempenho às funcionalidades desenvolvidas e uma análise aos seus resultados. Os testes incidirão sobre os modelos de expressões e de regras, e sobre o desempenho do processamento gráfico do componente visualizador.

8.3.1 Modelo de Expressões

Este modelo permite a avaliação do resultado de expressões contra valores contidos em gestores de atributos locais, no caso, eventos. Com vista a facilitar a análise e como forma a garantir uma base de trabalho constante em todos os ensaios, todas as expressões baseiam-se na utilização do mesmo evento (`Alarm`), constituído por três atributos: `Value` (do tipo `long`), `Alarm` (do tipo `boolean`) e `OtherValue` (do tipo `long`).

8.3.1.1 Descrição dos Testes

Os testes que ora se apresentam não pretendem cobrir de forma exaustiva todas as especificidades inerentes ao processamento de expressões, mas antes permitir uma avaliação, minimamente fundamentada, do tipo de resposta esperada em situações de utilização comuns.

Foram definidas 5 expressões como base para a realização dos ensaios, as quais serão doravante denominadas por EXP1, EXP2, EXP3, EXP4 e EXP5. Admitiu-se, para estes ensaios, que o tempo de processamento de uma operação é igual para todos os tipos de operações, exceptuando para as operações lógicas (`AND` e `OR`). Para estes casos, o tempo de avaliação poderá variar, em função do valor do primeiro operando. No caso do operador `AND`, por exemplo, a obtenção do valor lógico *falso* no primeiro operando é suficiente para concluir a operação.

A primeira expressão, EXP1, define uma expressão aritmética simples, com um nível, baseada num evento e tem a seguinte estrutura:

$$EXP1 = (Analog.Value != 20)$$

A expressão EXP2 pretende ser uma "duplicação" de EXP1, ou seja, apresenta uma estrutura de dois níveis, em que o segundo nível é constituído por duas expressões semelhantes a EXP1, que são conjugados pela operação lógica `AND`:

$$EXP2 = (Analog.Value < 40)\ AND\ (Analog.Value == 10)$$

O objectivo da expressão EXP3 é o de introduzir uma variação na lógica de construção realizada até aqui. Assim, EXP3 introduz operadores aritméticos em ambos os "ramos" simples de EXP2, criando uma expressão de três níveis. É importante notar que

apesar de haver mais um nível nesta expressão, o número de operandos baseados em eventos mantém-se igual a EXP2, ou seja, 2.

$$EXP3 = ((Analog.Value + 10) < 40)\, AND\, ((Analog.Value\,/\,2) < 10)$$

A construção da expressão EXP4 segue uma lógica semelhante à utilizada na evolução da EXP1 para a EXP2, ou seja pretende-se "duplicar" a EXP3 pela utilização de um ramo de estrutura semelhante, resultando numa expressão de 4 níveis, onde existem 4 referências a atributos do evento. Apresenta-se a expressão EXP4 em três passos:

$$EXP4.1 = EXP3$$

$$EXP4.2 = ((Analog.OtherValue - 5) == 15)\, AND\, (Analog.Alarm == FALSE)$$

$$EXP4 = EXP4.1\ AND\ EXP4.2$$

Finalmente a expressão EXP5 apresenta uma simplificação de EXP4, mas mantendo o seu significado, pela remoção das operações aritméticas com eventos. A sua apresentação será feita em três passos para simplificação:

$$EXP5.1 = (Analog.OtherValue < 30)\, AND\, (Analog.Value < 20)$$

$$EXP5.2 = (Analog.OtherValue == 20)\, AND\, (Analog.Alarm == FALSE)$$

$$EXP5 = EXP5.1\ AND\ EXP5.2$$

A tabela 8.1 apresenta a lista de ensaios realizados com cada expressão. Os campos `Value`, `OtherValue` e `Alarm` representam os valores presentes no evento `Analog` para cada ensaio. Os ensaios estão numerados para cada expressão. O primeiro ensaio de cada expressão resulta numa avaliação com resultado lógico de *verdadeiro*. O último ensaio de cada expressão utiliza um evento cujos atributos contêm valores que são avaliados como *falso* em todos os níveis da expressão. Estes dois ensaios (o primeiro e o último) constituem casos extremos de funcionamento para cada expressão e serão denominados por *extremo positivo* e *extremo negativo*, respectivamente.

8.3.1.2 Análise dos Resultados

A tabela 8.2 apresenta o tempo gasto por cada expressão (em milissegundos) para a execução do respectivo ensaio, repetido um milhão de vezes.

A figura 8.1 apresenta um gráfico onde se ilustra o resultado correspondente aos extremos positivos de cada expressão. Daqui podemos ver, uma evolução mais ou menos linear para aquelas expressões que foram construídas a partir da duplicação de outras expressões, casos de EXP1 para EXP2, de EXP2 para EXP5 e de EXP3 para EXP4. Vemos também, que a evolução de EXP2 para EXP3 não é tão significativa como a de EXP1 para EXP2. Isto acontece porque o número de operações introduzidas (em termos percentuais) de EXP1 para EXP2 e de EXP2 para EXP5 é superior à

Expressão	Teste	*Value*	*Alarm*	*OtherValue*	Resultado
EXP1	TESTE1	10	-	-	**true**
	TESTE2	20	-	-	**false**
EXP2	TESTE1	10	-	-	**true**
	TESTE2	20	-	-	**false**
	TESTE3	50	-	-	**false**
EXP3	TESTE1	10	-	-	**true**
	TESTE2	25	-	-	**false**
	TESTE3	50	-	-	**false**
EXP4	TESTE1	10	false	-	**true**
	TESTE2	10	true	-	**false**
	TESTE3	10	false	-	**false**
	TESTE4	10	true	-	**false**
	TESTE5	50	true	-	**false**
EXP5	TESTE1	10	false	20	**true**
	TESTE2	10	true	20	**false**
	TESTE3	10	false	25	**false**
	TESTE4	10	true	25	**false**
	TESTE5	50	true	25	**false**

Tabela 8.1: Parâmetros de ensaios ao modelo de expressões

Expressão	Teste	Repetições	Resultado	Tempo (ms)
EXP1	TESTE1	1 milhão	**true**	**811**
	TESTE2	1 milhão	**false**	**811**
EXP2	TESTE1	1 milhão	**true**	**1873**
	TESTE2	1 milhão	**false**	**842**
	TESTE3	1 milhão	**false**	**841**
EXP3	TESTE1	1 milhão	**true**	**2574**
	TESTE2	1 milhão	**false**	**1372**
	TESTE3	1 milhão	**false**	**1362**
EXP4	TESTE1	1 milhão	**true**	**4837**
	TESTE2	1 milhão	**false**	**4836**
	TESTE3	1 milhão	**false**	**4035**
	TESTE4	1 milhão	**false**	**4066**
	TESTE5	1 milhão	**false**	**1512**
EXP5	TESTE1	1 milhão	**true**	**3946**
	TESTE2	1 milhão	**false**	**4016**
	TESTE3	1 milhão	**false**	**3144**
	TESTE4	1 milhão	**false**	**3104**
	TESTE5	1 milhão	**false**	**1172**

Tabela 8.2: Resultado dos ensaios ao modelo de expressões

introduzida de EXP2 para EXP3. De facto, EXP3 e EXP5 apresentam estruturas com mesmo número de níveis (3) mas EXP5 contém mais operações.

O tempo gasto no processamento de uma expressão pode ser sub-dividido em três componentes: o tempo de processamento das operações, o tempo de extracção dos atributos dos eventos e o tempo de extracção dos atributos constantes. Assumindo que o tempo de cada uma destas componentes se mantém constante independentemente do tipo de eventos, tipo de atributos e tipo de operações de facto executadas, é possível obter o valor de cada uma destas componentes a partir dos resultados dos ensaios para o extremo positivo de EXP1, EXP2 e EXP3, pela resolução do seguinte sistema de equações:

$$\begin{cases} op + event + const = 811 \\ 3op + 2event + 2const = 1873 \\ 5op + 2event + 4const = 2574 \end{cases} \tag{8.1}$$

De onde se conclui que:

$$\begin{cases} op = 251 \\ event = 460 \\ const \approx 100 \end{cases} \tag{8.2}$$

Onde `op` constitui o tempo de processamento de um milhão de operações, `event` o tempo de extracção de um milhão de atributos de eventos e `const` o tempo de extracção de um milhão de atributos de constantes.

Utilizando estes valores como meio de cálculo dos tempos de avaliação do extremo positivo para as expressões EXP4 e EXP5, teremos:

$$\tau_{extremopositivoEXP4} = 10op + 4event + 7const = 5050 \tag{8.3}$$

$$\tau_{extremopositivoEXP5} = 7op + 4event + 4const = 3997 \tag{8.4}$$

Conclui-se que a utilização dos valores obtidos em 8.2 para o cálculo do tempo de execução do extremo positivo das expressões EXP4 e EXP5 resultam em tempos finais próximos dos tempos de facto obtidos, com uma margem de erro de 4,2% no caso de EXP4 e 1,3% para o caso de EXP5. O facto dos tempos de execução calculados constituirem majorantes dos tempos de facto obtidos, permitem justificar a utilização dos valores de 8.2 para a efectuar previsões do tempo de processamento de expressões numa base "pessimista".

Ainda referente aos valores de 8.2, verifica-se que a execução de operações baseadas em atributos de eventos é mais cara do que utilizando operações baseadas em atributos constantes. Deste modo, o tempo máximo de execução de uma expressão acontece no caso em que o número de operandos do tipo evento é igual ao número total de operações utilizadas mais um. Utilizando os valores calculados, podemos expressar o tempo máximo de execução de um milhão de expressões como sendo (em milissegundos):

$$\tau_{max1milhao} = 711op + 460 \tag{8.5}$$

Utilizando a equação 8.5 é possível calcular o tempo máximo de processamento de expressões em função do número médio de operações nelas contidas. Assim, e assumindo uma média de 10 operações por expressão, o modelo permite a execução de cerca de 13200 expressões dentro do limite de tempo real.

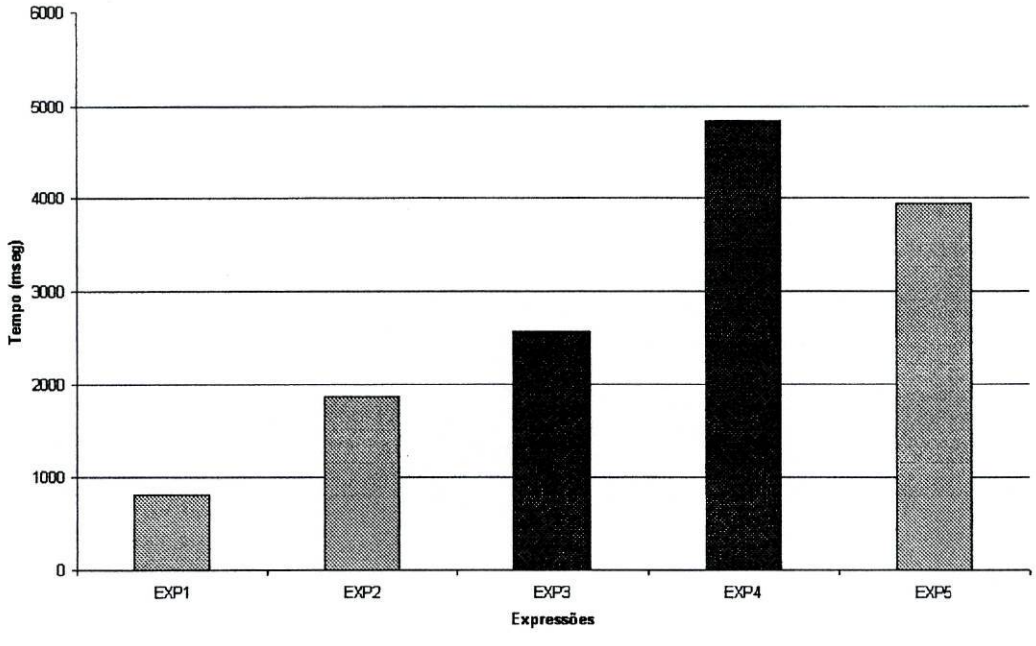


Figura 8.1: Tempo de avaliação de expressões — extremo positivo

De salientar ainda os valores obtidos para os ensaios do extremo negativo . Estes valores revelam o discutido anteriormente, onde o processamento das expressões é interrompido assim que um dos operadores lógicos é capaz de inferir o resultado final (figura 8.2).

8.3.2 Modelo de Regras

O modelo de regras permite a criação de sacos de regras que são avaliados contra instâncias de gestores locais de atributos. Este modelo não é mais do que uma especialização do modelo anterior, pelo que é espectável que os resultados dos ensaios sejam consonantes com os resultados já discutidos para as expressões.

8.3.2.1 Descrição dos Testes

Os ensaios a este modelo baseiam-se em sacos de regras construídos com base nas expressões discutidas na secção anterior.

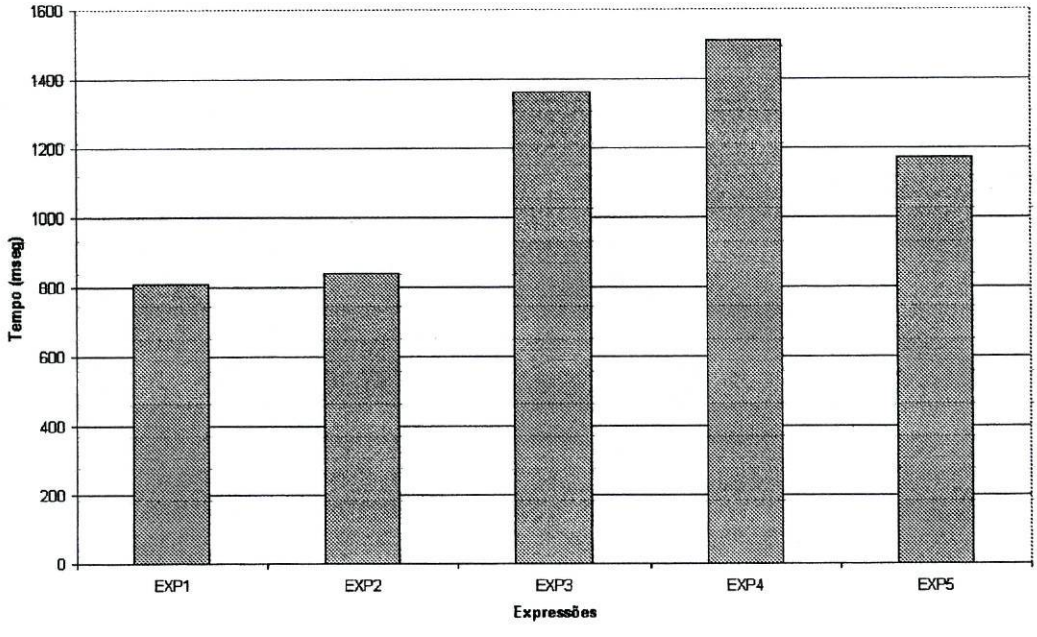


Figura 8.2: Tempo de avaliação de expressões — extremo negativo

Regras				
R1(EXP1), R2(EXP2), R3(EXP3), R4(EXP4)				
Teste	***Value***	***Alarm***	***OtherValue***	**Resultado**
TESTE1	10	false	20	R1, R2, R3, R4
TESTE2	10	true	20	R1, R2, R3
TESTE3	15	true	20	R1, R3
TESTE4	15	false	20	R1, R3, R4
TESTE5	20	false	20	DEFAULT

Tabela 8.3: Parâmetros de ensaios ao modelo de regras — Cenário 1

A tabela 8.3 apresenta as condições dos ensaios realizados no primeiro cenário. Neste cenário, criou-se um saco de regras constituído por quatro regras R1, R2, R3 e R4, cada uma incorporando as expressões descritas para os ensaios anteriores pela mesma ordem, ou seja, EXP1, EXP2, EXP3 e EXP4.

O cenário 2, encontra-se descrito na tabela 8.4, e apresenta como diferença o facto de um dos atributos do evento `Analog` não se encontrar preenchido. Ora, pela definição do modelo de expressões apresentado no capítulo 6, a tentativa de avaliação de uma expressão baseada num evento, dá origem ao envio de uma excepção no caso do atributo desse evento não estar preenchido. Para efeitos do motor de regras, essa situação resulta numa avaliação negativa da regra, no caso, a regra R4. O que se presente medir com este cenário são os tempos introduzidos pela ocorrência de excepções durante o processamento das regras.

Regras				
R1(EXP1), R2(EXP2), R3(EXP3), R4(EXP4)				
Teste	***Value***	***Alarm***	***OtherValue***	**Resultado**
TESTE1	10	false	null	R1, R2, R3
TESTE2	10	true	null	R1, R2, R3
TESTE3	15	true	null	R1, R3
TESTE4	15	false	null	R1, R3
TESTE5	20	false	null	DEFAULT

Tabela 8.4: Parâmetros de ensaios ao modelo de regras — Cenário 2

8.3.2.2 Análise dos Resultados

As tabelas 8.5 e 8.6 apresentam o resultado dos ensaios para os cenários descritos anteriormente.

Regras			
R1(EXP1), R2(EXP2), R3(EXP3), R4(EXP4)			
Teste	**Repetições**	**Resultado**	**Tempo (ms)**
TESTE1	1 milhão	R1, R2, R3, R4	13670
TESTE2	1 milhão	R1, R2, R3	13299
TESTE3	1 milhão	R1, R3	11567
TESTE4	1 milhão	R1, R3, R4	11847
TESTE5	1 milhão	DEFAULT	6329

Tabela 8.5: Resultado dos ensaios ao modelo de regras — Cenário 1

Por análise da tabela 8.5, verifica-se que o tempo de execução das regras é superior ao somatório dos tempos gastos pela execução isolada das expressões que as constituem, o que pode ser atribuído ao tempo de processamento gasto no preenchimento da lista

Regras			
R1(EXP1), R2(EXP2), R3(EXP3), R4(EXP4)			
Teste	**Repetições**	**Resultado**	**Tempo (ms)**
TESTE1	1 milhão	R1, R2, R3	30003
TESTE2	1 milhão	R1, R2, R3	29442
TESTE3	1 milhão	R1, R3	28183
TESTE4	1 milhão	R1, R3	28111
TESTE5	1 milhão	DEFAULT	6370

Tabela 8.6: Resultado dos ensaios ao modelo de regras — Cenário 2

com os resultados das avaliações (lista de acções). Tomando o pior caso como referência (TESTE5) para a construção de um lista só com um resultado, pode esperar-se um aumento de processamento, de 1800 milissegundos por cada milhão de regras bem sucedidas. Na figura 8.3 apresentam-se graficamente os tempos de execução associados ao cenário 1.

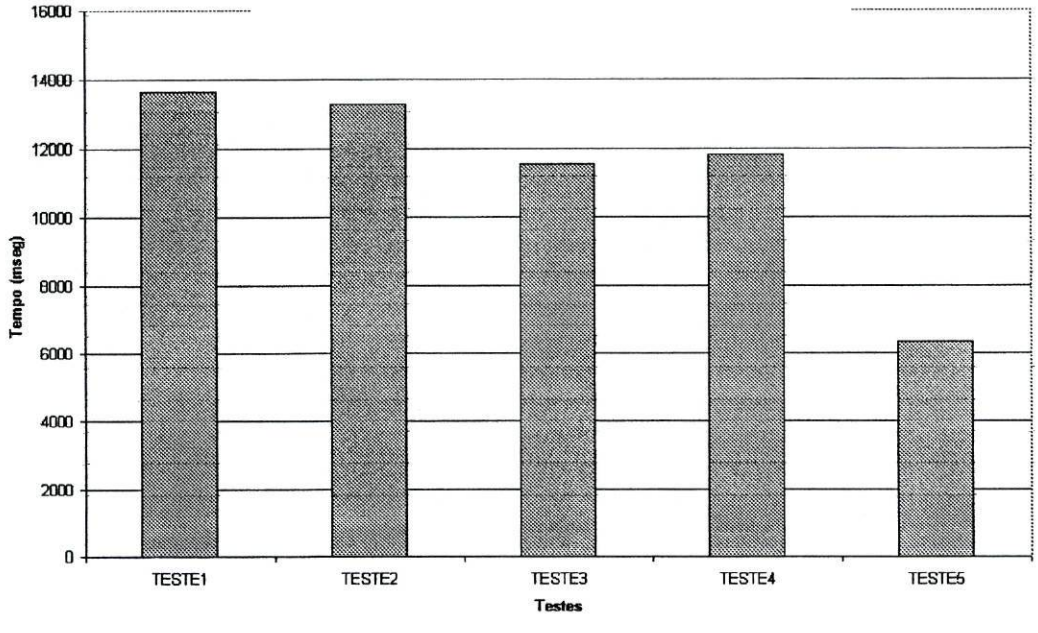


Figura 8.3: Tempo de avaliação de regras — Cenário 1

Na figura 8.4, por seu turno, ilustra-se o tempo gasto pelos ensaios correspondentes ao cenário 2. Verifica-se que a ocorrência de uma excepção na avaliação da regra R4, produz uma degradação considerável no processamento em relação ao cenário 1, que se traduz num aumento de cerca de 16 segundos do tempo de processamento de um milhão de regras. A única excepção corresponde ao TESTE5, onde o tempo é semelhante ao anterior. Tal facto pode ser explicado pela processo de avaliação da expressão EXP4

que, para os valores do evento, consegue ser avaliada antes da ocorrência da excepção.

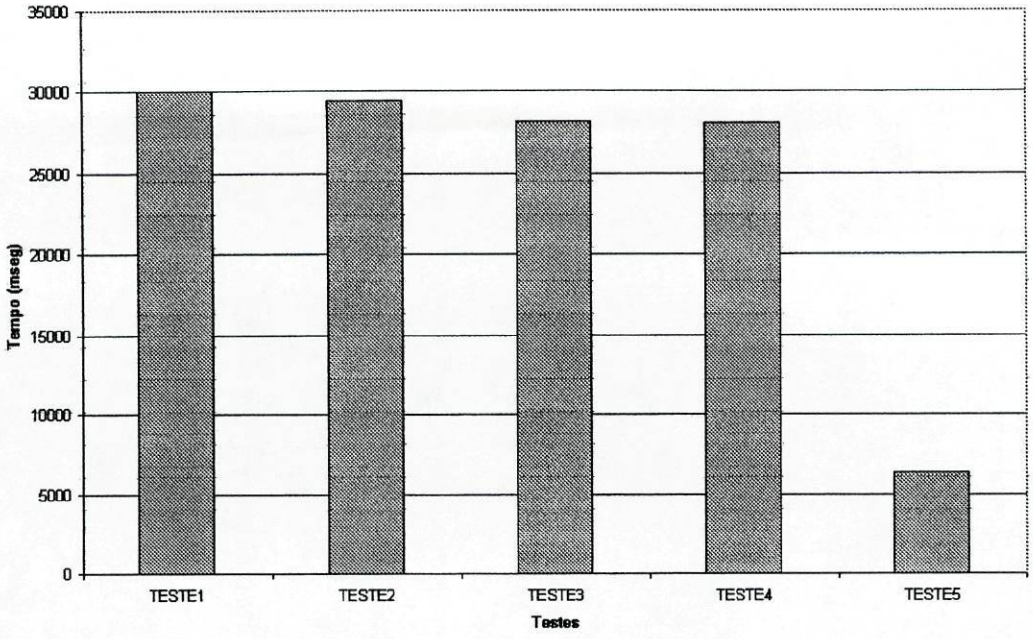


Figura 8.4: Tempo de avaliação de regras — Cenário 2

Como conclusão, poder-se-á dizer que o tempo de processamento gasto na execução de regras é superior ao somatório do tempo gasto na execução das suas expressões em 1800 milissegundos por cada milhão de regras avaliadas com sucesso, a menos das situações de ocorrência de excepções. Nesses casos, e tomando o valor obtido como medida, poder-se-á esperar um aumento do tempo de processamento de cerca de 16 segundos, por cada milhão de regras avaliadas, por cada regra que processe uma excepção.

Estas conclusões dificultam a obtenção de um valor de referência para a avaliação de regras no limite de tempo real, na medida em que esse valor dependerá do tempo gasto no preenchimento da lista de resultados, do número médio de regras dos sacos de regras e da probabilidades de ocorrência de excepções durante a sua avaliação.

Como exemplo, imaginando que os sacos de regras têm em média 5 regras, e cada uma é constituída por cerca de 10 operações, e assumindo que ocorre uma excepção na última operação de cada expressão em metade das avaliações, pode calcular-se o tempo máximo, aproximado, de avaliação de 1 milhão de sacos de regras como:

$$\tau_{saco} = 5 * (10 * 711 + 460 + 16000/2 + 1800/2) = 82350\, milissegundos \quad (8.6)$$

Para este caso, temos que é possível avaliar cerca de 1200 sacos de regras dentro do limite de tempo real.

8.3.3 Processamento Gráfico

Os testes de avaliação de desempenho do processamento gráfico do componente visualizador englobam as actividades de pesquisa nas estruturas de dados, processo de desenho na imagem *off-screen* e o processo de passagem final dessa imagem para o contexto gráfico do dispositivo, ou seja, o ecrã.

A razão de tomar todas estas actividades como uma actividade atómica, em vez de analisar cada uma delas separadamente, baseou-se no facto destas serem as actividades que decorrem desde o momento de actuação, por parte do utilizador, até ao momento de visualização do resultado correspondente, que é precisamente o que importa medir.

8.3.3.1 Descrição dos Testes

Para a realização dos ensaios construiu-se uma aplicação, baseada no componente visualizador, onde são desenhadas geometrias simples (no caso, rectângulos e círculos), distribuídas uniformemente pela área de desenho, com três comportamentos distintos: figuras de tamanho variável estáticas, figuras de tamanho variável animadas (em rotação) e figuras de tamanho fixo. O mesmo objecto gráfico é partilhado por figuras do mesmo tipo, havendo por isso três instâncias de objectos gráficos diferentes. A distribuição das figuras pela área de desenho é conseguida pela aplicação de transformadas diferentes, por cada figura, antes do processo de desenho do componente gráfico correspondente.

A tabela 8.7 apresenta as características de cada um dos 5 testes realizados. Nela são descritas a dimensão da área de visualização (em pontos)[2], o número de cada tipo de objectos, o número total de objectos e os tipos de estruturas de dados utilizadas. Da tabela ressalta o facto do número de objectos de tamanho fixo ser muito menor do que o número de objectos de tamanho variável. O motivo desta diferença reside no facto dos objectos de tamanho fixo não serem geridos por estruturas de organização espacial (ver capítulo 6), fazendo com que a sua utilização, em número elevado, pudesse dificultar a análise dos resultados.

Os testes foram executados pela utilização livre da aplicação, que incluiu as actividades de ampliação, arrastamento, selecção de objectos e variação do tamanho da área de visualização.

8.3.3.2 Análise dos Resultados

Os resultados dos ensaios que se discutem apresentam tempos médios de processamento para várias quantidades de objectos desenhados. Devido à natureza aleatória dos ensaios, não foi possível apresentar resultados, referentes ao mesmo número de objectos, para todos os ensaios. Outra consequência desta situação, é a possibilidade

[2]Esta dimensão corresponde à dimensão da imagem *off-screen*

Teste	Objectos			Total de Objectos	Estruturas de Dados	Área de Desenho (pontos)
	Variáveis Estáticos	Variáveis Animados	Variáveis Estáticos			
TESTE1	500	500	20	1020	Quadtree + Set	1000x1000
TESTE2	2500	2500	50	5050	Quadtree + Set	1000x1000
TESTE3	5000	5000	100	10100	Quadtree + Set	1000x1000
TESTE4	5000	5000	100	10100	Quadtree + Set	10000x10000
TESTE5	10000	0	100	10100	Quadtree + Set	10000x10000
TESTE6	5000	5000	100	10100	Set	10000x10000

Tabela 8.7: Parâmetros dos ensaios ao processamento gráfico

de existirem tempos baseados numa só amostra e outros baseados em várias amostras, levando à ocorrência de perturbações significativas à uniformidades de alguns resultados[3]. De qualquer modo, é a opinião do autor, que os resultados verificados servem como base razoável de avaliação do desempenho desta funcionalidade.

As figuras 8.5, 8.6, 8.7, 8.8 e 8.9 apresentam os gráficos relativos ao resultado dos ensaios 1 a 5. O ensaio 6 (TESTE6) resultou na actividade de desenho constante, de todos os 10100 objectos, a cada pedido de processamento, independentemente dos parâmetros de visualização (ampliação, arrastamento, etc). A média dos valores obtidos para o tempo de processamento neste ensaio cifrou-se nos 330 milissegundos.

Pala análise dos gráficos apresentados, verifica-se a existência de um padrão mais ou menos linear de evolução das suas curvas (a menos das perturbações atrás referidas), o que é aliás comprovado pelos diversos coeficientes de correlação das aproximações lineares calculadas para cada uma (valores de R^2 superiores a 0,8). Será com base nestas "aproximações" que serão discutidos os resultados:

- A análise do gráfico 8.5 permite verificar que a manipulação de 1000 objectos é sempre inferior ao limite de tempo-real, independentemente do número de objectos de facto visualizados. Através da curva de aproximação, podemos inferir que o limite de tempo-real seria atingido para a visualização de cerca de 1700 objectos.

- O aumento do número total de objectos a visualizar (como acontece de 8.5 para 8.6 e de 8.6 para 8.7) degrada o desempenho gráfico, desde que sejam mantidas as áreas de visualização. Para 8.6 o limite de tempo real ronda os 1300 objectos, e para 8.7 este limite é de 930 objectos. O factor principal para esta degradação será o aumento de níveis da *quadtree* de gestão espacial, o que prejudica os tempos de pesquisa.

- O aumento da área de desenho aumenta o desempenho gráfico para o mesmo número total de objectos, tal como se verifica de 8.7 para 8.8. Assumindo uma

[3]Como é o caso da entrada em funcionamento do *garbage collector* que leva à introdução de atrasos no tempo de processamento.

distribuição uniforme (como é o caso) será espectável obter o mesmo número de níveis em ambas as *quadtrees*, uma vez que estas não operam em função da área que gerem, mas sim em função do número de objectos nela contidos. O aumento de desempenho do segundo caso, justifica-se pelo facto de haver menor probabilidade de um objecto ser atribuído a dois ramos de um nó da *quadtree*, já que o seu tamanho permanece o mesmo em ambas as situações. Havendo menos duplicações haverá uma consequente diminuição dos tempos de pesquisa, acelerando o processo de desenho.

- A melhoria de desempenho verificada de 8.8 para 8.9 (aumento dos limites de tempo-real de 1128 para 1730 objectos) pode ser explicada pela inexistência do *thread* de processamento de animações.

A análise conjunta dos resultados, em comparação com o valor médio obtido pelo TESTE6 (330 milissegundos), três vezes superior ao período de tempo-real, permite concluir que a existência de um mecanismo de gestão espacial garante a usabilidade da aplicação em tempo-real, mesmo nos casos de manipulação de um grande número de objectos, desde que o número de objectos presentes na zona visível seja inferior ao limite de tempo-real.

Em função dos testes realizados, poderemos assumir como valor razoável de limite de tempo-real do visualizador, para a representação de geometrias simples, um valor à volta dos 1000 objectos, mesmo para situações de manipulação de um número elevado de objectos (dezenas de milhar). Convém contudo realçar, que este valor será certamente inferior no caso de utilização de objectos gráficos mais complexos, tais como imagens ou composições de vários objectos simples.

8.4 Testes ao Protótipo

Esta secção pretende analisar os tempos associados ao carregamento de diagramas do componente de simbologia. Para tal utilizar-se-á o protótipo descrito no capítulo 7.

8.4.1 Descrição dos Testes

Os ensaios baseiam-se na medição dos tempos de *setup* do *ScadaEngine* na situação de carregamento dos ficheiros de simbologia, estilos e construção do diagrama. Para tal utilizaram-se ficheiros com as seguintes características:

simbologia — Foi criado um ficheiro contendo 1000 símbolos baseados em geometrias simples. O tamanho final do ficheiro é de cerca de 100 kB.

estilos — Este ficheiro definiu 1000 estilos compostos por uma única regra, cada um, associada a cada um dos 1000 símbolos presentes na simbologia. O tamanho

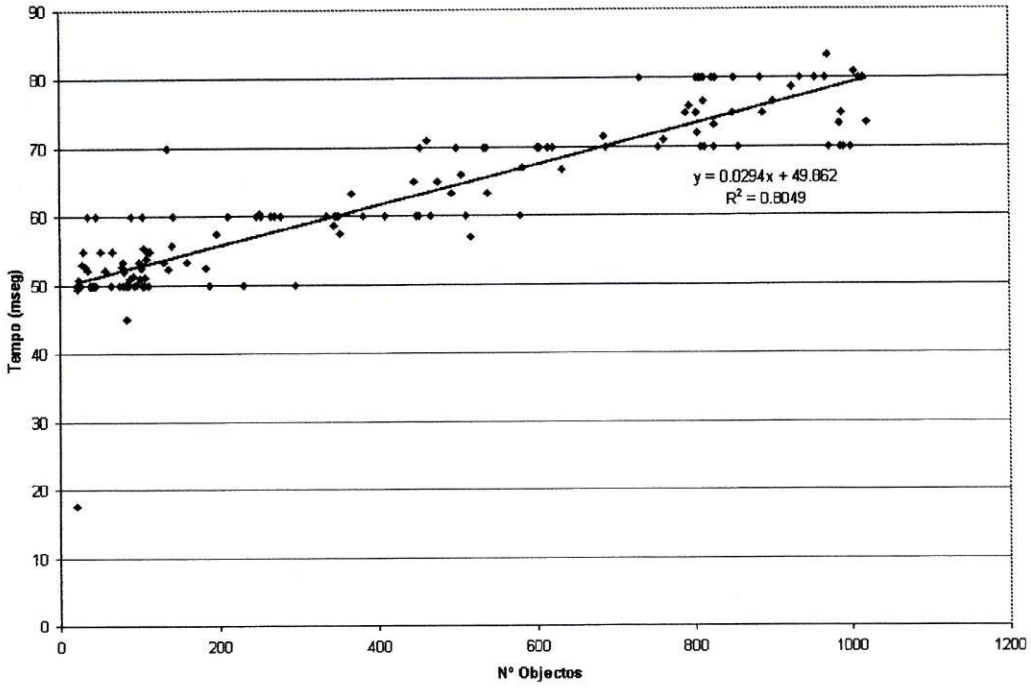


Figura 8.5: Resultados dos ensaios ao processamento gráfico — Teste 1

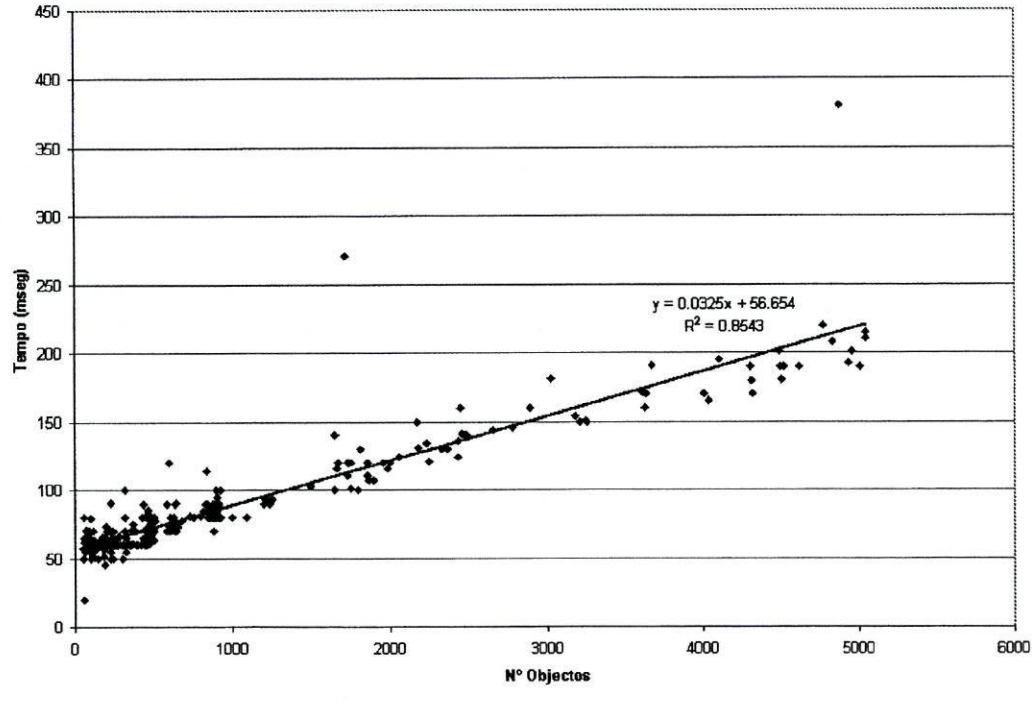


Figura 8.6: Resultados dos ensaios ao processamento gráfico — Teste 2

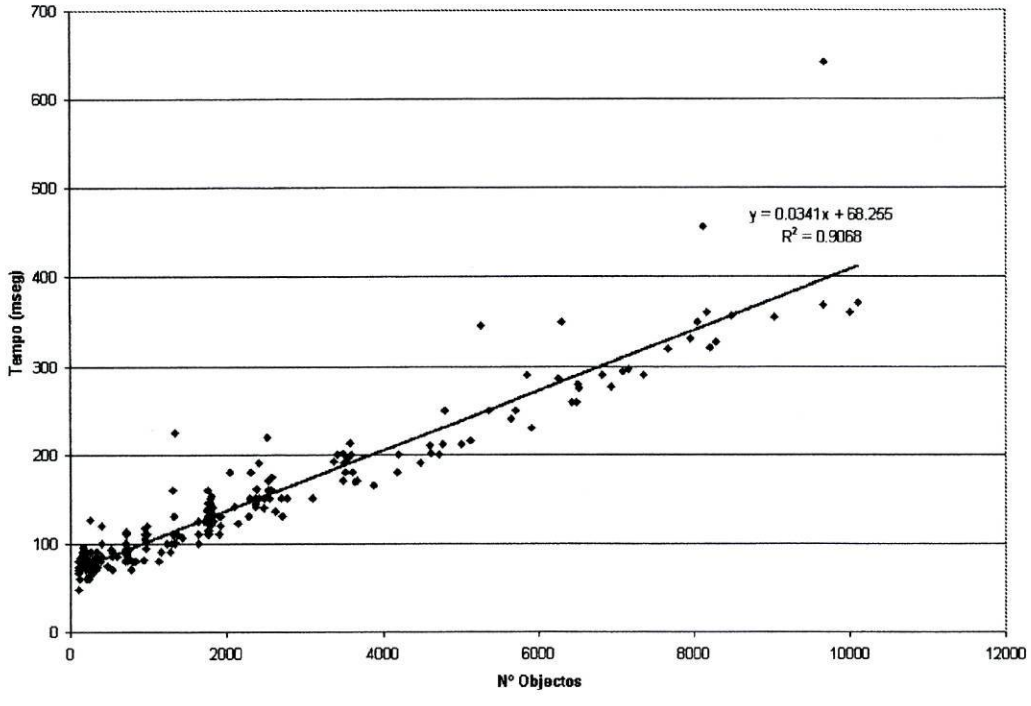


Figura 8.7: Resultados dos ensaios ao processamento gráfico — Teste 3

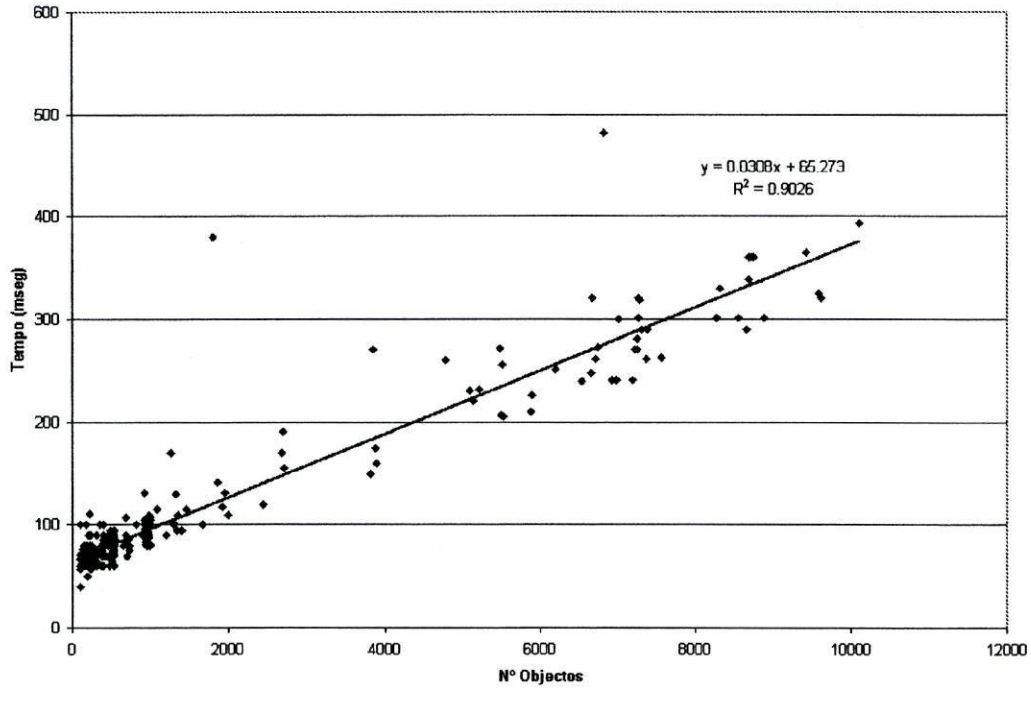


Figura 8.8: Resultados dos ensaios ao processamento gráfico — Teste 4

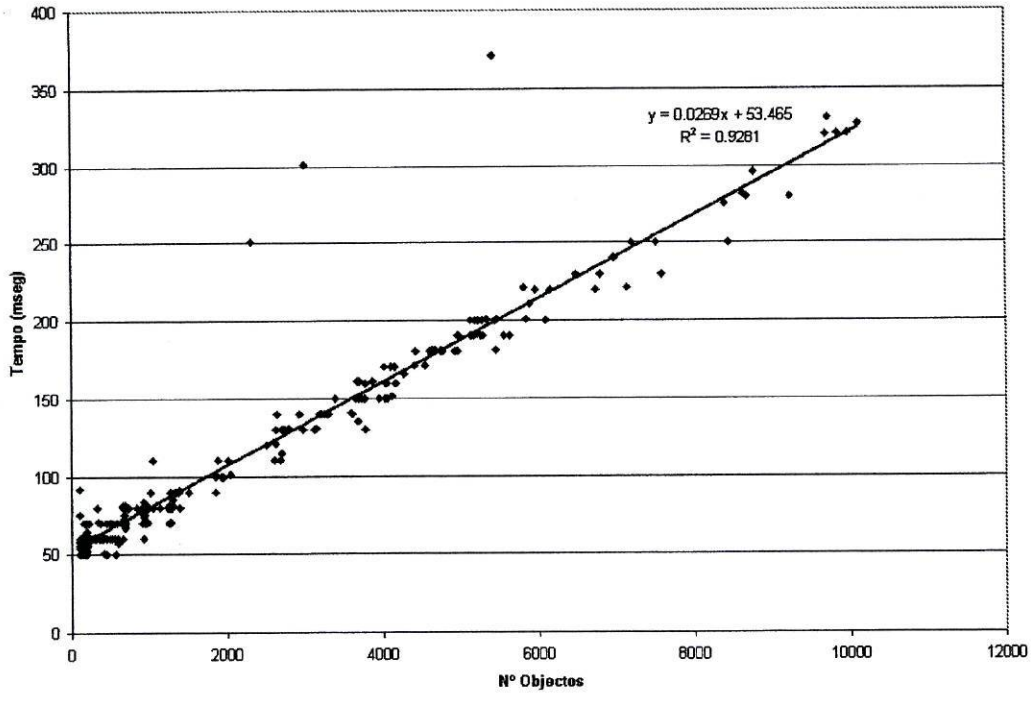


Figura 8.9: Resultados dos ensaios ao processamento gráfico — Teste 5

final do ficheiro é de cerca de 350 kB.

diagrama — O diagrama consistiu na distribuição aleatória de 1000 figuras, cada uma baseada num dos 1000 estilos definidos. O tamanho final do ficheiro de construção do diagrama cifrou-se nos 75 kB.

Os testes consistiram na execução de 5 carregamentos do diagrama e na medição dos tempos associados ao processamento individual de cada um dos ficheiros.

Os teste foram executados localmente, pelo que não entram em consideração com o tempo de acesso aos ficheiros pela rede.

8.4.2 Análise dos Resultados

A figura 8.10 apresenta o resultado dos testes onde se verifica que o tempo de processamento do modelo de simbologia é bastante superior a cada um dos outros dois. O tempo total de carregamento de um diagrama, consiste na soma de todos os tempos parciais e anda à volta dos 7 segundos, o que, mesmo para um número elevado de símbolos e estilos, se mantém dentro do limite de atenção do utilizador.

As conclusões que se podem extrair dos ensaios apresentados é que embora os tempos sejam aceitáveis do ponto de vista de usabilidade, tal situação pode ser agravada se o acesso ao sistema se verificar através de canais de comunicação com baixa largura de

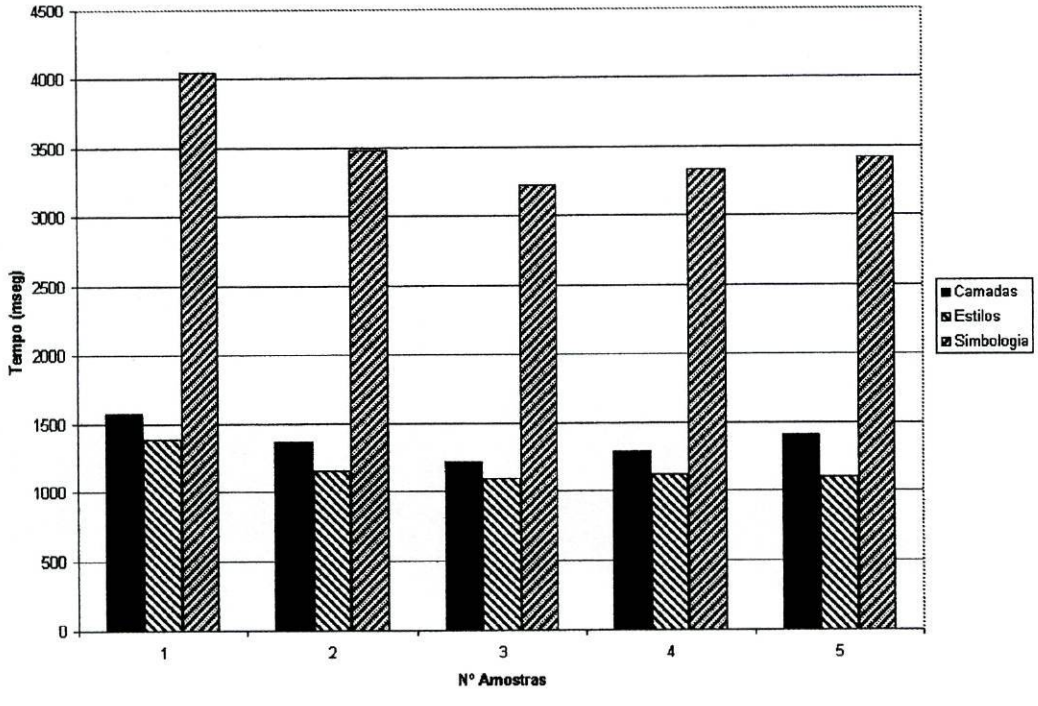


Figura 8.10: Resultados dos ensaios ao protótipo

banda. Contudo, convém também referir que os testes basearam-se num número de símbolos e estilos elevados. De facto, o modelo de simbologia foi pensado para suprir as necessidades de criação de diagramas cujas figuras são baseadas num conjunto restrito de símbolos que serão representados inúmeras vezes. Ora este princípio, prevê que o número de estilos e de símbolos de um sistema sejam bastante menores que as figuras a representar, minimizando assim o impacto dos resultados apresentados.

8.5 Sumário

Este capítulo apresentou os testes de desempenho das funcionalidades disponíveis na biblioteca de funcionalidades SCADA.

A análise dos resultados baseou-se em métricas de avaliação de usabilidade vulgarmente utilizadas na indústria do *software*, as quais definem limites para a interacção com aplicações em tempo real.

Pelos resultados obtidos, apresentaram-se os valores de funcionamento para situações de operação próximas do limite de tempo real. É a opinião do autor que estes valores constituem resultados interessantes, permitindo a utilização destas funcionalidades como suporte ao desenvolvimento de aplicações SCADA, gráficas ou não, desde que o seu projecto tome em consideração as condições de funcionamento que mais beneficiam o seu desempenho.

Uma nota final para os resultados de carregamento de diagramas, onde foram verificados tempos superiores ao limite de tempo real, mas que ainda se mantêm dentro dos limites de atenção do utilizador, para os exemplos apresentados.

Capítulo 9

Conclusões

Neste capítulo revêm-se os pontos principais discutidos neste documento com vista à apreciação geral do trabalho desenvolvido e apresentam-se sugestões para trabalhos futuros.

9.1 Objectivos Iniciais

Recuperando o objectivo apresentado no capítulo 1, este trabalho propunha-se a estudar e demonstrar a possibilidade de disponibilização de aplicações gráficas SCADA na *Web*. Nessa altura definiram-se três áreas de actuação:

- Identificar os tipos de aplicações típicas de sistemas SCADA, analisar as metáforas gráficas mais comuns e caracterizar o tipo de informação que circula entre aplicações;
- Apresentar uma arquitectura apropriada para o desenvolvimento de interfaces gráficos SCADA na *Web*;
- Provar a sua viabilidade pela disponibilização de interfaces *Web* que apresentem as mesmas características das aplicações gráficas SCADA convencionais.

Do consequente estudo, resultou uma lista dos requisitos que deveriam ser verificados pela solução final. Estes requisitos, apresentados no capítulo 4, versaram três áreas distintas:

Requisitos SCADA — Requisitos decorrentes das características típicas das aplicações gráficas SCADA, tal como apresentado no capítulo 2;

Requisitos EFACEC SE/AS — Requisitos obtidos a partir de consultas de opinião aos responsáveis dos sistemas desenvolvidos na EFACEC SE/AS;

Requisitos Comerciais — Requisitos que visam a obtenção de soluções com valor máximo, i.e. mínimo custo e máxima funcionalidade.

9.2 Escolha da Solução e Principais Dificuldades

A definição dos requisitos permitiu fundamentar a escolha da solução final e consequente enquadramento da tese aqui defendida nos seguintes termos:

A implementação de interfaces gráficos na Web para aplicações típicas de sistemas SCADA pode ser conseguida pela utilização de componentes Java especializados que executam localmente no navegador. A utilização destes componentes implica o desenvolvimento de uma arquitectura de software que suporte a sua integração com as aplicações residentes no centro de comando, nomeadamente pela implementação de um mecanismo uniforme de comunicação construído em cima do protocolo HTTP e de outros protocolos por elas utilizados.

Ou seja, defende-se o desenvolvimento de componentes Java especializados para cada tipo de funcionalidade gráfica SCADA. Uma vez desenvolvidos, estes componentes podem ser integrados livremente para a construção de aplicações.

Devido ao tempo limitado para a realização do presente trabalho, concluiu-se não haver a possibilidade de desenvolver componentes que cobrissem a totalidade das áreas de intervenção das aplicações SCADA, pelo que se optou pela construção de uma biblioteca generalista, composta por funcionalidades úteis para o desenvolvimento destes componentes. O resultado foi uma biblioteca de funcionalidades gráficas e não gráficas, e que por isso, apresenta o benefício de poder ser utilizada para o desenvolvimento de outras aplicações, mesmo para aquelas menos vocacionadas para a interacção com o utilizador. Embora limitando o resultado final, que seria a implementação de vários componentes especializados, conseguiu-se a disponibilização de uma infra-estrutura mais abrangente.

A utilidade desta biblioteca ficou comprovada pela implementação de um componente especializado para a visualização de diagramas sinópticos.

A outra actividade desenvolvida, relacionou-se com a definição de uma arquitectura que permitisse a integração das aplicações locais (no navegador) com o sistema residente no centro de comando. Esta arquitectura inspirou-se numa arquitectura de *software* existente na EFACEC SE/AS, que se baseia em componentes distribuídos e colaborantes para a implementação de uma funcionalidade específica. O modelo de colaboração entre componentes centra-se no princípio da publicação/subscrição de tópicos para a troca de eventos. A arquitectura final, materializada no *framework* de componentes, juntou-lhe a capacidade de comunicação sobre diversos protocolos, entre os quais o HTTP.

O sucesso da definição desta arquitectura permitiu separar a tarefa de desenvolvimento

de componentes gráficos especializados, do processo de construção de aplicações.

A principal dificuldade sentida ao longo da execução deste trabalho residiu na multitude de áreas versadas durante o seu desenvolvimento, tanto para a biblioteca de funcionalidades SCADA como para o *framework* de componentes colaborantes. Estas áreas cobriram assuntos tão diversos como os protocolos de comunicação, definição de gramáticas XML e sua interpretação, estudo de modelos de expressões, estudo de modelos de regras ou estratégias de gestão e manipulação gráfica de objectos.

9.3 Cumprimento dos Objectivos?

A tese apresentada define uma estratégia de actuação, baseada nos requisitos estabelecidos, com vista ao cumprimento do objectivo inicial.

Nessa estratégia, a implementação de interfaces SCADA na *Web* depende de dois factores: a disponibilização de componentes gráficos especializados e a existência de uma arquitectura de *software* conforme às especificidades dos sistemas (SCADA e *Web*).

Em relação à arquitectura de *software*, a discussão do capítulo 4 apresentou os princípios básicos tomados em consideração para o desenvolvimento do *framework* de componentes, o qual é descrito em pormenor no capítulo 5. Embora este trabalho não tenha versado a problemática da sua implementação sobre diversos protocolos, procurou-se a definição de um modelo que prevê essa execução, nomeadamente pela disponibilização de comunicadores especializados. Esta abordagem de isolamento entre a lógica de comunicação e a lógica da aplicação permite o desenvolvimento de aplicações passíveis de serem utilizadas noutros ambientes, que não somente a *Web*, constituindo um benefício extremamente interessante para o processo de desenvolvimento de *software*, por via da diminuição de custos decorrentes da capacidade de reutilização de implementações. Uma prova cabal deste princípio é apresentada no capítulo 7, onde se introduz o *ScadaEngine*, uma aplicação SCADA passível de ser integrada numa *applet* para execução num navegador, ou podendo actuar isoladamente como uma aplicação *stand-alone*.

No que respeita aos componentes gráficos, este trabalho reduziu-se à implementação de um único, o componente de simbologia, especializado para a visualização de diagramas sinópticos. Embora não tendo sido possível a disponibilização de componentes para todas as variantes de aplicações gráficas SCADA, cremos que aquele que se apresenta permite verificar o sucesso da estratégia estabelecida. Esta conclusão é fundamentada não só pela constatação da sua aplicabilidade prática, demonstrada no protótipo, como também pelos resultados dos testes de interacção gráfica analisados no capítulo 8, onde se verificou ser viável a sua utilização em situações de operação de tempo-real, mesmo para um número significativo de objectos gráficos representados.

Em alternativa à implementação de vários componentes, este trabalho propôs o desen-

volvimento de uma biblioteca generalista, dotada de ferramentas pensadas para suprir as necessidades de vários componentes especializados e assim minorar, por um lado, a falha no cumprimento do objectivo principal (todas as aplicações) e estabelecer, por outro, uma estrutura de base para o desenvolvimento de qualquer funcionalidade SCADA. O capítulo 8 ofereceu uma análise pragmática a algumas destas funcionalidades (regras e expressões), de onde se pôde concluir ser igualmente viável a sua utilização em aplicações que operam em situações de tempo-real. Aliás, se recordarmos os requisitos SCADA de tempo-real apresentados (capítulo 2), verifica-se que o número de regras e de expressões passíveis de serem processadas, dentro do limite de tempo-real (100 milissegundos), excede, em muito, a taxa de recepção de informação de entidades então apresentada (cerca de 200 por segundo), provando a viabilidade da utilização destas ferramentas nessas situações.

A tabela 9.3 apresenta uma análise da forma como cada uma das soluções escolhidas verificam os requisitos apresentados no capítulo 4, bem como uma descrição sumária das razões ou, em alguns casos, as possibilidades de utilização que conduziram à adopção de cada uma delas. Da sua análise conclui-se terem sido cumpridos, quase na totalidade, os requisitos apresentados.

Em resumo, este trabalho provou a exequibilidade da tese como forma de disponibilização de interfaces SCADA na *Web*, embora tal não tenha sido demonstrado para todos os aspectos das aplicações SCADA. Em alternativa, apresentou-se uma biblioteca generalista, que em função dos resultados obtidos, permite concluir que constitui uma base sólida para a implementação de qualquer tipo de aplicação gráfica SCADA.

9.4 Oportunidades de Melhoria

Nesta secção discutem-se dois tipos de intervenções que visam a melhoria do trabalho realizado: acções correctivas e acções evolutivas.

As acções correctivas constituem alterações ao trabalho desenvolvido que permitem corrigir factores menos conseguidos na implementação apresentada. As alterações propostas no âmbito destas acções são:

- Reestruturação dos modelos desenvolvidos no âmbito da biblioteca de funcionalidades SCADA, com vista à criação de unidades especializadas, encapsuladas em *JavaBeans*. Deste modo, o processo de criação de aplicações poderia resumir-se à associação de componentes *JavaBeans*, eventualmente com recurso a um IDE (*Integrated Development Environment*);
- Revêr o algoritmo de construção da *quadtree*, por forma a evitar a disseminação do mesmo objecto por vários dos seus ramos. Alternativas são a utilização de *quadtrees* de quadrantes variáveis, ou a manutenção de objectos partilhados no nó superior (*MX-CIF quadtree* [Sam95]). Outra opção consiste na utilização de outro tipo de estrutura de organização espacial mais eficaz do que a utilizada;

Solução Apresentada	Razões/Utilizações	Requisitos Cumpridos
Opções Tecnológicas		
Java	Multi-plataforma, orientada ao objecto, conjunto vasto de bibliotecas *standard*, bibliotecas gráficas evoluídas, suportada em navegadores, utilização gratuita	12, 14, 15
XML	Definição de gramáticas *ad hoc*, natureza textual da informação, mecanismos de validação sintáctica, amplamente divulgado, existência de várias bibliotecas de processamento (gratuitas)	9, 14
Componentes Gráficos	Especialização (um componente uma funcionalidade), reutilização (applets, aplicações), interacção local com o utilizador	1, 8, 10
Applets	Execução em navegadores, seguras, gestão centralizada	8, 11, 12, 13[1], 15
Framework de Componentes		
Framework para Aplicações	Infra-estrutura para o rápido desenvolvimento de aplicações, facilidades úteis à generalidade das aplicações (logging, leitura de parâmetros, etc)	-
Modelo de Componentes	Modelo de programação distribuída, encapsula especificidades de comunicação, inter-operável	7, 9, 11
Biblioteca de Funcionalidades SCADA		
Modelo de Gestores de Atributos	Atributos flexíveis, operações entre atributos (polimorfismo), gestores de atributos, meta-informação, eventos, conversão de e para formato texto	-
Modelo de Expressões	Construção, manutenção e execução de expressões baseadas em gestores de atributos, carregamento e conversão de e para XML, regras, filtros	6
Modelo de Regras	Construção, manutenção e execução de regras, baseadas em gestores de atributos, simbologia	4
Modelo de Componentes Gráficos	Esqueleto para o desenvolvimento de componentes gráficos especializados, modelo de interacção gráfica (ferramentas), figuras, contentores e animações	3
Gestão Espacial	Optimização de manipulação de objectos dispostos espacialmente, camadas	5
Implementações Standard	Ferramentas comuns, organização de figuras em camadas, gestão de animações e intermitência de imagens	3, 5
Modelo de Simbologia	Símbolos, estilos e camadas, definição externa de modelos de dados, componente especializado para a visualização de diagramas sinópticos	1[2], 2, 3, 4, 7[3]

Tabela 9.1: Verificação dos Requisitos

- Utilização de SVG como forma de definição dos objectos de simbologia, permitindo assim o carregamento de diagramas construídos noutros ambientes que exportem este formato;
- Definição de expressões com recurso a MathML, que é também uma gramática XML especialmente concebida para a representação de expressões matemáticas;
- Revisão do conceito de símbolo composto, permitindo a utilização de paletas diferentes para cada um dos seus sub-elementos. Esta melhoria permite o desenvolvimento de objectos gráficos mais ricos e apelativos;

As acções evolutivas consistem em propostas de melhoria do trabalho desenvolvido pela introdução de novas funcionalidades. Destas, destacam-se:

- Desenvolvimento de componentes especializados para a visualização de listas (alarmes e entidades) e gráficos de tendência, de modo a completar o leque de funcionalidades comuns às aplicações SCADA típicas;
- Integração do modelo de expressões na arquitectura de componentes colaborantes, permitindo estabelecer critérios de filtragem em função do conteúdo dos eventos. No caso de comunicações por canais de baixa largura de banda, esta melhoria poderá restringir o tráfego de eventos àqueles realmente importantes para as aplicações;
- Desenvolvimento de novas ferramentas (da caixa de ferramentas) que permitam a execução de manipulações mais complexas sobre figuras, tais como ferramentas de desenho que poderão ser utilizadas para a construção de um editor diagramas;
- Desenvolvimento de novos símbolos e animações úteis para a criação de diagramas SCADA;
- Definição de um modelo de classes para os tipos de entidades comuns aos sistemas SCADA, que possa ser reaproveitado por várias aplicações/componentes;
- Desenvolvimento de um editor de diagramas que forneça uma interface gráfica para a construção dos diagramas, símbolos e estilos, permitindo a geração automática dos ficheiros XML respectivos;
- Desenvolvimento de aplicações que permitam a edição gráfica de regras e expressões, culminando com a geração dos ficheiros XML respectivos;

[1] Considera-se o esforço de configuração das bibliotecas desprezável, devido à sua simplicidade, tal como discutido no capítulo 7.

[2] O modelo de simbologia implementa apenas uma das funcionalidades SCADA.

[3] O modelo de simbologia não define de facto uma estratégia para a execução de controlos, antes permite a sua execução através da especialização dos modelo de simbologia, tal como foi demonstrado no protótipo (capítulo 7).

9.5 Sumário

Nesta dissertação defendeu-se a utilização de componentes gráficos Java especializados como forma de disponibilização de interfaces SCADA na *Web*.

Nela foi demonstrada a viabilidade desta abordagem, mas apenas para um tipo específico de aplicações, os diagramas sinópticos. Em alternativa desenvolveu-se uma biblioteca de funcionalidades genéricas, que poderá ser utilizada como forma de obviar o processo de desenvolvimento daqueles componentes.

A estratégia passou também pela definição de uma arquitectura baseada em componentes colaborantes, a qual se revelou interessante não só para o desenvolvimento de aplicações *Web*, como também para a sua utilização noutros contextos. Este factor reveste-se como uma mais-valia significativa, no âmbito da indústria do *software*, onde a capacidade de reutilização de implementações constitui uma vantagem competitiva importante.

Finalmente, conclui-se que o presente trabalho cumpriu, quase na totalidade, os objectivos inicialmente propostos.

Apêndice A

Acrónimos

4GL	Fourth Generation Language
ADO	Active Data Objects
ADSL	Asymmetric Digital Subscriber Line
API	Application Programming Interface
ASP	Active Server Pages
CERN	Conceil Européen pour la Recherche Nucléaire
CGI	Common Gateway Interface
CLV	Comunicação e Localização de Viaturas
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheets
DCOM	Distributed COM
DMS	Distribution Management System
DOM	Document Object Model
DTD	Document Type Definition
EDP	Electricidade de Portugal
EJB	Enterprise JavaBeans
FTP	File Transfer Protocol
GIF	Graphics Interchange Format
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IHM	Interface Homem-Máquina
IETF	Internet Engineering Task Force
IP	Internet Protocol
J2EE	Java 2 Platform, Enterprise Edition
J2SE	Java 2 Platform, Standard Edition
JCA	Java Connectivity Architecture
JCP	Java Community Process
JSR	Java Specification Request

JDBC	Java Database Connectivity
JDK	Java Development Kit
JMS	Java Messaging Service
JMX	Java Management Extensions
JNDI	Java Naming Directory Interface
JRE	Java Runtime Environment
JSP	Java Server Pages
LTE	Electricidade de Lisboa e Vale do Tejo
MTU	Master Terminal Unit
NCSA	National Center for Supercomputing Applications
PAS	Power Application System
PC	Personal Computer
PDA	Portable Digital Assistant
PDF	Portable Document Format
PHP	PHP: Hypertext Preprocessor
PNG	Portable Network Graphics
RDF	Resource Description Framework
RFC	Request for Comments
RMI	Remote Method Invocation
SAX	Simple API for XML
SCADA	Supervisory Control and Data Acquisition
SEE	Sistemas Eléctricos de Energia
SGBDR	Sistema de Gestão de Bases de Dados Relacionais
SGC	Software de Gestão Central
SGI	Sistema de Gestão de Incidentes
SGML	Standard Generalized Markup Language
SI	Sistemas de Informação
SIG	Sistema de Informação Geográfica
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
UDDI	Universal Description Discovery an Integration
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
URT	Unidade Remota Terminal
URR	Unidade Remota Reduzida
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
WWW	World Wide Web
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSL-FO	XSL Formatting Objects
XSLT	Extensible Stylesheet Language Transformations

Apêndice B

Esquema XML para o Motor de Expressões

B.1 Esquema expressions.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="cexp">
    <xs:annotation>
      <xs:documentation>
        The root element for conditions
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:choice>
          <xs:element ref="bexp"/>
          <xs:element ref="rexp"/>
        </xs:choice>
        <xs:element ref="comment" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="bexp">
    <xs:annotation>
      <xs:documentation>
        A Boolean Expression
      </xs:documentation>
    </xs:annotation>
```

```
    <xs:complexType>
      <xs:choice>
        <xs:element ref="and"/>
        <xs:element ref="or"/>
        <xs:element ref="not"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="rexp">
    <xs:annotation>
      <xs:documentation>
        A Relational Expression
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:choice>
        <xs:element ref="diff"/>
        <xs:element ref="equal"/>
        <xs:element ref="greater"/>
        <xs:element ref="less"/>
        <xs:element ref="isNull"/>
        <xs:element ref="isNotNull"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="aexp">
    <xs:annotation>
      <xs:documentation>
        An Arithmetic Expression
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:choice>
        <xs:element ref="add"/>
        <xs:element ref="sub"/>
        <xs:element ref="mul"/>
        <xs:element ref="div"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="and" type="BooleanOperator">
    <xs:annotation>
      <xs:documentation>The operator AND</xs:documentation>
    </xs:annotation>
  </xs:element>
```

```
<xs:element name="or" type="BooleanOperator">
  <xs:annotation>
    <xs:documentation>The operator OR</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="not" type="BooleanOperator">
  <xs:annotation>
    <xs:documentation>The operator NOT</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="greater" type="RelationalOperator">
  <xs:annotation>
    <xs:documentation>The operator 'greater'</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="less" type="RelationalOperator">
  <xs:annotation>
    <xs:documentation>The operator 'lesser'</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="equal" type="RelationalOperator">
  <xs:annotation>
    <xs:documentation>The operator '='</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="diff" type="RelationalOperator">
  <xs:annotation>
    <xs:documentation>The operator '!='</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="add" type="ArithmeticOperator">
  <xs:annotation>
    <xs:documentation>The operator '+'</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="sub" type="ArithmeticOperator">
  <xs:annotation>
    <xs:documentation>The operator '-'</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="mul" type="ArithmeticOperator">
  <xs:annotation>
    <xs:documentation>The operator '*'</xs:documentation>
  </xs:annotation>
</xs:element>
```

```
<xs:element name="div" type="ArithmeticOperator">
  <xs:annotation>
    <xs:documentation>The operator '/'</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="event">
  <xs:annotation>
    <xs:documentation>The event operand</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="attribute" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="const">
  <xs:annotation>
    <xs:documentation>The constant operand</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="type" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:complexType name="BooleanOperator">
  <xs:annotation>
    <xs:documentation>
      The type of all boolean operators
    </xs:documentation>
  </xs:annotation>
  <xs:choice maxOccurs="2">
    <xs:element ref="bexp"/>
    <xs:element ref="rexp"/>
  </xs:choice>
</xs:complexType>
<xs:complexType name="RelationalOperator">
  <xs:annotation>
    <xs:documentation>
      The type of all relational operators
    </xs:documentation>
  </xs:annotation>
  <xs:choice maxOccurs="2">
    <xs:element ref="aexp"/>
```

```
      <xs:element ref="const"/>
      <xs:element ref="lam"/>
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="ArithmeticOperator">
    <xs:annotation>
      <xs:documentation>
        The type of all arithmetic operators
      </xs:documentation>
    </xs:annotation>
    <xs:choice minOccurs="2" maxOccurs="2">
      <xs:element ref="aexp"/>
      <xs:element ref="const"/>
      <xs:element ref="lam"/>
    </xs:choice>
  </xs:complexType>
  <xs:element name="comment" type="xs:string">
    <xs:annotation>
      <xs:documentation>
        Used for writing the expression in the regular format
      </xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="lam">
    <xs:annotation>
      <xs:documentation>A container of values</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence minOccurs="0">
        <xs:annotation>
          <xs:documentation>
            In case of null use the following attribute value
          </xs:documentation>
        </xs:annotation>
        <xs:element ref="const"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="attribute" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="isNull" type="RelationalOperator">
    <xs:annotation>
      <xs:documentation>The operator 'is null'</xs:documentation>
    </xs:annotation>
  </xs:element>
```

```
  <xs:element name="isNotNull" type="RelationalOperator">
    <xs:annotation>
      <xs:documentation>The operator 'is not null'</xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:schema>
```

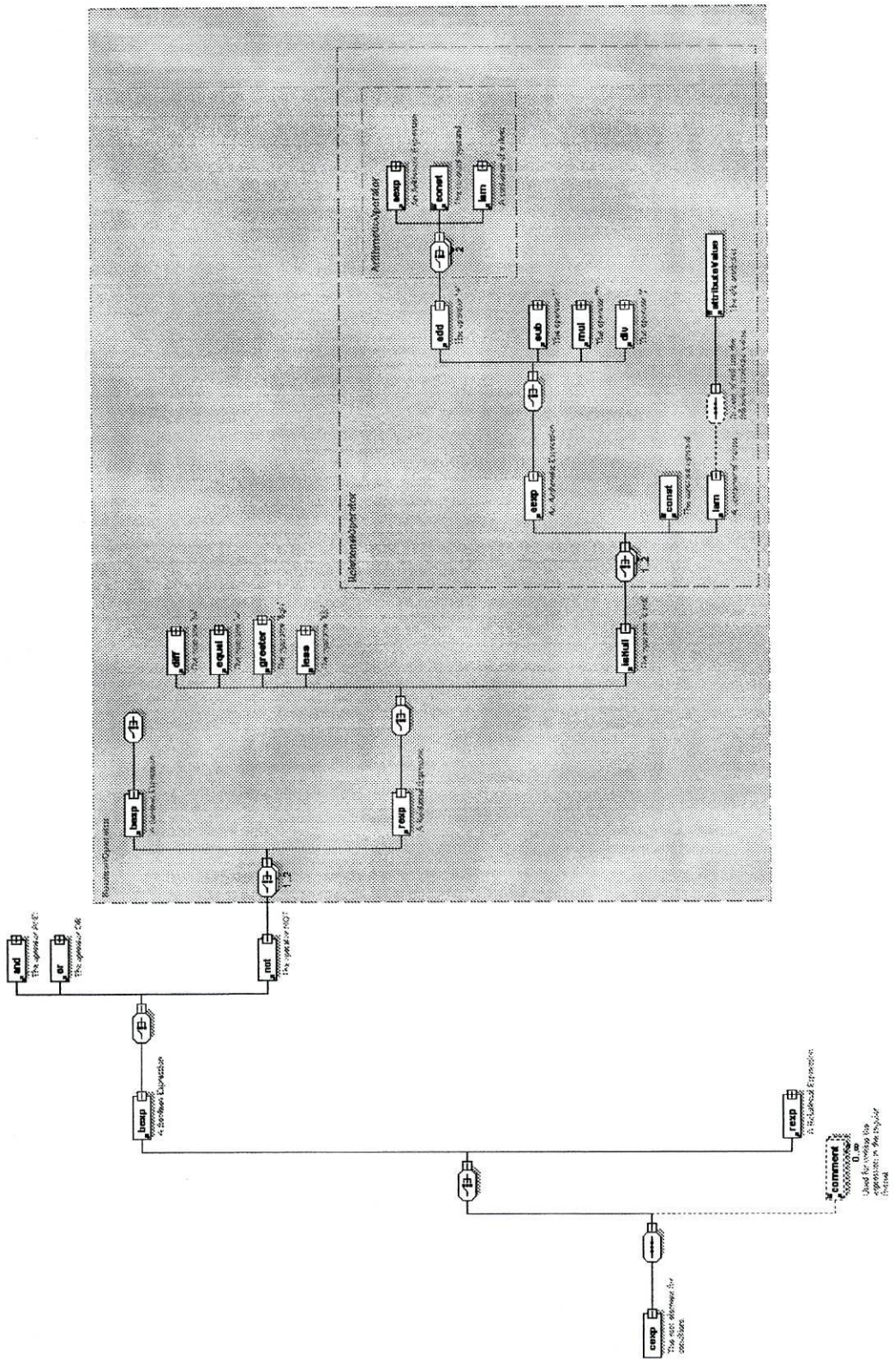


Figura B.1: Representação gráfica do esquema XML de expressões

Apêndice C

Esquema XML para o Motor de Regras

C.1 Esquema rules.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:cond="expressions
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:import namespace="." schemaLocation="expressions.xsd"/>
  <xs:element name="ruleset">
    <xs:annotation>
      <xs:documentation>A set of rules</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="default"/>
        <xs:element ref="rule" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="rule">
    <xs:annotation>
      <xs:documentation>if condition then action</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="if"/>
        <xs:element ref="then"/>
      </xs:sequence>
```

```
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="if">
    <xs:annotation>
      <xs:documentation>Groups conditions</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="cexp">
          <xs:complexType/>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="rules">
    <xs:annotation>
      <xs:documentation>Root element</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element ref="ruleset"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="default" type="xs:string">
    <xs:annotation>
      <xs:documentation>The default action</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="then" type="xs:string">
    <xs:annotation>
      <xs:documentation>The action associated with a rule</xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:schema>
```

Apêndice D

Esquemas XML para o Motor de Simbologia

D.1 Esquema symbology.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="." xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="." elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="symbology">
    <xs:annotation>
      <xs:documentation>The symbology</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="symbol" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="palette" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="symbol">
    <xs:annotation>
      <xs:documentation>A symbol</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element ref="circle"/>
        <xs:element ref="image"/>
        <xs:element ref="label"/>
        <xs:element ref="polygon"/>
```

```
      <xs:element ref="polyline"/>
      <xs:element ref="rect"/>
      <xs:element ref="text"/>
      <xs:element ref="ellipse"/>
      <xs:element ref="imageWalker"/>
      <xs:element ref="line"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="rect">
  <xs:annotation>
    <xs:documentation>A rectangular shape</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="AnySymbol">
  <xs:annotation>
    <xs:documentation>Commom definitions</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute name="tsx" type="xs:double" use="optional"/>
    <xs:attribute name="tsy" type="xs:double" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="point">
  <xs:annotation>
    <xs:documentation>A point</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="polygon">
  <xs:annotation>
    <xs:documentation>A polygon</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="point" minOccurs="3" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="polyline">
  <xs:annotation>
    <xs:documentation>A shape made of multiple lines</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
```

```
        <xs:element ref="point" minOccurs="2" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="circle">
    <xs:annotation>
      <xs:documentation>A circle</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="text">
    <xs:annotation>
      <xs:documentation>A variable text string</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="label">
    <xs:annotation>
      <xs:documentation>A fixed text string</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="image">
    <xs:annotation>
      <xs:documentation>An image</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="palette">
    <xs:annotation>
      <xs:documentation>The palette definition</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="ellipse">
    <xs:annotation>
      <xs:documentation>An ellipse</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="line">
    <xs:annotation>
      <xs:documentation>A single line</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="imageWalker">
    <xs:annotation>
      <xs:documentation>A image that follows some path</xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:schema>
```

D.2 Esquema multilayer.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:cond="symbology" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:import namespace="." schemaLocation="symbology.xsd"/>
  <xs:element name="multilayer">
    <xs:annotation>
      <xs:documentation>The multilayer definition</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="layer"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="layer">
    <xs:annotation>
      <xs:documentation>A layer of figures</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="figure" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="figure">
    <xs:annotation>
      <xs:documentation>A figure</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="symbol"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Apêndice E

Configurações do Protótipo – ScadaEngine

E.1 Definição de Simbologia — symbology.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<symbology name="ScadaSymbology">
  <symbol name="LinkSymbol">
    <circle fill="true" x="0" y="0" r="20"/>
  </symbol>
  <symbol name="LampSymbol">
    <circle fill="true" x="0" y="0" r="10"/>
  </symbol>
  <symbol name="RectSwitchOn">
    <rect fill="true" x="-25" y="-50" w="50" h="50"/>
  </symbol>
  <symbol name="RectSwitchOff">
    <rect fill="false" x="-25" y="-50" w="50" h="50"/>
  </symbol>
  <symbol name="MeasureViewer">
    <text attribute="Value" nchars="10"/>
  </symbol>
  <symbol name="Transformer">
    <circle fill="false" x="0" y="0" r="5" tsx="5" tsy="5"/>
    <circle fill="false" x="0" y="4.5" r="5" tsx="5" tsy="5"/>
  </symbol>
  <palette name="LightOnPalette" bgcolor="white" fgcolor="yellow"
  scolor="cyan" blink="false"/>
  <palette name="LightOffPalette" bgcolor="white" fgcolor="black"
  scolor="cyan" blink="true"/>
```

```
  <palette name="Normal" bgcolor="white" fgcolor="black" scolor="cyan"
  blink="false"/>
  <palette name="NormalSynoptic" bgcolor="black" fgcolor="white"
  scolor="cyan" blink="false"/>
  <palette name="LinkPalette" bgcolor="lightGray" fgcolor="blue"
  scolor="cyan" blink="false"/>
  <palette name="Alarm" bgcolor="lightGray" fgcolor="orange"
  scolor="gray" blink="true"/>
  <palette name="Danger" bgcolor="lightGray" fgcolor="red"
  scolor="gray" blink="true"/>
  <palette name="Synoptic" bgcolor="black" fgcolor="white"
  scolor="red" blink="true"/>
</symbology>
```

E.2 Definição de Estilos — styles.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
  <ruleset name="Lamp">
    <default>LampSymbol.LightOnPalette</default>
    <rule name="LightOn">
      <if>
        <cexp>
          <rexp>
            <equal>
              <lam name="Digital" attribute="On"/>
              <const type="boolean">true</const>
            </equal>
          </rexp>
        </cexp>
      </if>
      <then>.LightOnPalette</then>
    </rule>
    <rule name="LightOff">
      <if>
        <cexp>
          <rexp>
            <equal>1
              <lam name="Digital" attribute="On"/>
              <const type="boolean">false</const>
            </equal>
          </rexp>
        </cexp>
```

```
    </if>
    <then>.LightOffPalette</then>
  </rule>
</ruleset>
<ruleset name="RectSwitchStyle">
  <default>RectSwitchOn.NormalSynoptic</default>
  <rule name="SetRectSwitchOn">
    <if>
      <cexp>
        <rexp>
          <equal>
            <lam name="Digital" attribute="On"/>
            <const type="boolean">true</const>
          </equal>
        </rexp>
      </cexp>
    </if>
    <then>RectSwitchOn</then>
  </rule>
  <rule name="SetRectSwitchOff">
    <if>
      <cexp>
        <rexp>
          <equal>
            <lam name="Digital" attribute="On"/>
            <const type="boolean">false</const>
          </equal>
        </rexp>
      </cexp>
    </if>
    <then>RectSwitchOff</then>
  </rule>
  <rule name="SetAlarm">
    <if>
      <cexp>
        <rexp>
          <equal>
            <lam name="any" attribute="Alarm"/>
            <const type="boolean">true</const>
          </equal>
        </rexp>
      </cexp>
    </if>
    <then>.Alarm</then>
  </rule>
```

```
    <rule name="ResetAlarm">
      <if>
        <cexp>
          <rexp>
            <equal>
              <lam name="any" attribute="Alarm"/>
              <const type="boolean">false</const>
            </equal>
          </rexp>
        </cexp>
      </if>
      <then>.NormalSynoptic</then>
    </rule>
  </ruleset>
  <ruleset name="MeasureStyle">
    <default>MeasureViewer.NormalSynoptic</default>
    <rule name="SetAlarm">
      <if>
        <cexp>
          <rexp>
            <greater>
              <lam name="Analog" attribute="Value"/>
              <const type="long">100</const>
            </greater>
          </rexp>
        </cexp>
      </if>
      <then>.Alarm</then>
    </rule>
    <rule name="ResetAlarm">
      <if>
        <cexp>
          <rexp>
            <less>
              <lam name="Analog" attribute="Value"/>
              <const type="long">101</const>
            </less>
          </rexp>
        </cexp>
      </if>
      <then>.NormalSynoptic</then>
    </rule>
  </ruleset>
  <ruleset name="TransformerStyle" static="false">
    <default>Transformer.NormalSynoptic</default>
```

```
  </ruleset>
  <ruleset name="Line" static="true">
    <default>.Normal</default>
  </ruleset>
  <ruleset name="LineSynoptic" static="true">
    <default>.NormalSynoptic</default>
  </ruleset>
  <ruleset name="Static" static="true">
    <default>.Normal</default>
  </ruleset>
  <ruleset name="StaticSynoptic" static="true">
    <default>.NormalSynoptic</default>
  </ruleset>
  <ruleset name="Link">
    <default>LinkSymbol.LinkPalette</default>
  </ruleset>
</rules>
```

E.3 Definição de um Diagrama da Rede de Metropolitano — metrodiagram.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<multilayer name="Main" palette="Normal" x="0" y="0" w="2000" h="2000"
  isonoff="false" low="0" high="0">
  <layer name="Stations" palette="Normal" x="0" y="0" w="2000" h="2000"
  isonoff="false" low="0" high="0">
    <figure key="TAG1" style="Link"
      model="file:///c:/java/config/engine/substation_diagram_info.xml"
      sizable="false" tx="100" ty="100"/>
    <figure key="TAG2" style="Link"
      model="file:///c:/java/config/engine/substation_diagram_info.xml"
      sizable="false" tx="650" ty="550"/>

  (...)

    <figure key="TAG20" style="Static" model="None" sizable="false"
    tx="130" ty="100">
      <symbol>
        <label text="Station 1"/>
      </symbol>
    </figure>
    <figure key="TAG21" style="Static" model="None" sizable="false"
```

```
  tx="680" ty="550">
    <symbol>
      <label text="Station 2"/>
    </symbol>
  </figure>

(...)

  <figure key="TAG7" style="Line" model="None" sizable="true">
    <symbol>
      <polyline>
        <point x="100" y="100"/>
        <point x="650" y="550"/>
      </polyline>
    </symbol>
  </figure>
  <figure key="TAG8" style="Line" model="None" sizable="true">
    <symbol>
      <polyline>
        <point x="650" y="550"/>
        <point x="900" y="200"/>
      </polyline>
    </symbol>
  </figure>

(...)

  <figure key="TAG12" style="Static" model="Measure" sizable="true">
    <symbol>
      <imageWalker file="file:///c:/java/config/engine/car.png"
        low="0.0" high="100.0" attribute="Value">
        <point x="100" y="100"/>
        <point x="650" y="550"/>
        <point x="900" y="200"/>
        <point x="1050" y="1200"/>
        <point x="200" y="1000"/>
        <point x="1250" y="800"/>
      </imageWalker>
    </symbol>
  </figure>
</layer>
<layer name="Lights" palette="Normal" x="0" y="0" w="2000" h="2000"
isonoff="true" low="0.2" high="100">
  <figure key="TAG101" style="Lamp" model="Digital" sizable="true"
   tx="1100" ty="750" tsx="2" tsy="2"/>
```

```
    <figure key="TAG102" style="Lamp" model="Digital" sizable="true"
    tx="150" ty="250"/>
    <figure key="TAG103" style="Lamp" model="Digital" sizable="true"
    tx="100" ty="500" tsx="2" tsy="2"/>

   (...)

  </layer>
</multilayer>
```

E.4 Definição do Diagrama Unifilar de uma Subestação — substationdiagram.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<multilayer name="Main" palette="Synoptic" x="0" y="0" w="2000" h="2000"
  isonoff="false" low="0" high="0">
  <layer name="Synoptic" palette="Synoptic" x="0" y="0" w="2000" h="2000"
  isonoff="false" low="0" high="0">
    <figure key="TAG200" style="StaticSynoptic" model="None" sizable="true"
    tx="150" ty="200" tsx="5" tsy="5">
      <symbol>
        <label text="Substation"/>
      </symbol>
    </figure>
    <figure key="TAG201" style="RectSwitchStyle" model="Digital"
    sizable="true" tx="100" ty="500"/>
    <figure key="TAG202" style="MeasureStyle" model="Measure"
    sizable="true" tx="140" ty="450"/>

  (...)

    <figure key="TAG211" style="LineSynoptic" model="None" sizable="true">
      <symbol>
        <polyline>
          <point x="100" y="350"/>
          <point x="100" y="450"/>
        </polyline>
      </symbol>
    </figure>
    <figure key="TAG212" style="LineSynoptic" model="None" sizable="true">
      <symbol>
        <polyline>
```

```
          <point x="200" y="350"/>
          <point x="200" y="450"/>
        </polyline>
      </symbol>
    </figure>
    <figure key="TAG213" style="LineSynoptic" model="None" sizable="true">
      <symbol>
        <polyline>
          <point x="300" y="350"/>
          <point x="300" y="450"/>
        </polyline>
      </symbol>
    </figure>

    (...)

    <figure key="TAG225" style="TransformerStyle" model="None" sizable="true"
    tx="300" ty="650"/>
  </layer>
</multilayer>
```

Referências

[9402] JCP 94. *JSR 94 — Java Rule Engine API.* Java Community Process, 2002. Disponível em www.jcp.org/en/jsr/detail?id=94.

[All02] Jeremy Allaire. Macromedia Flash MX — A Next-Generation Rich Client, 2002. White Paper.

[Any] Symantec PC Anywhere. http://www.symantec.com.

[Ber97] Jesse Berst. ActiveX's Dirty Litte Secret, 1997. Disponível em www5.zdnet.com/anchordesk/story/story_813.html.

[BL] Tim Berners-Lee. World Wide Web Seminar. Apresentado na web.

[BL96] Tim Berners-Lee. The World Wide Web: Past, Present and Future. Artigo que seria publicado na edição especial do IEEE Computer que não foi editada. Disponível em http://www.w3.org/People/Berners-Lee/1996/ppf.html, Outubro 1996.

[Boy93] Stuart A. Boyer. *SCADA: Supervisory Control and Data Acquisition.* ISA - The Instrumentation, Systems and Automation Society, 1993.

[BS01] Bill Robins Brent Sleeper. Defining Web Services. Technical report, The Stencil Group, June 2001.

[But01] Mark H. Butler. Current Technologies for Device Independence. Technical report, Hewlett Packard, 2001.

[CH98] Dan Chang and Dan Harkey. *Client/Server Data Access with Java and XML.* John Wiley & Sons, Inc., 1998.

[Dis01] DistributechEurope. *Web Deploying an Integrated GIS/SCADA/OMS Solution*, Novembro 2001.

[DS99] A. Daneels and W. Salter. What is SCADA? *International Conference on Accelerator and Large Experimental Physics Control Systems*, 1999.

[EFA96] EFACEC Sistemas de Electrónica, S.A. *SCADA/DMS - Brief Technical Description*, Fevereiro 1996.

[EFA01] EFACEC Sistemas de Electrónica, S.A. *Apresentação da Empresa*, Novembro 2001.

[EFA02] EFACEC Sistemas de Electrónica, S.A. *Unidade de Negócio Automação de Sistemas de Energia*, 2002.

[EGV97] Ralph Johnson Erich Gamma, Richard Helm and John Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software.* Addinson-Wesley, 1997.

[E.J92] Ralph E.Johnson. Documenting Frameworks using Patterns, 1992. Artigo que terá sido apresentado na OOPSLA'92.

[Fla99] David Flanagan. *Java Foundation Classes in a Nutshell.* O'Reilly, 1999.

[Gom02] Miguel Ferreira Pereira Gomes. Canal SCADA na Web, 2002. Faculdade de Ciências da Universidade do Porto.

[GP02] Charles F. Goldfarb and Paul Prescod. *Charles F. Goldfarb's XML Handbook.* Prentice Hall PTR, 2002.

[HB98] Douglas Blank Hal Berghel. The World Wide Web. *ACM*, 1998.

[HC98] Jason Hunter and William Crawford. *Java Servlet Programming.* O'Reilly, October 1998.

[Hun] John Hunt. From Java to SVG. Disponível em http://appdevadvisor.co.uk/downloads/ada6_2/hunt6_2.pdf.

[IET98] IETF. *Uniform Resource Identifiers (URI): Generic Syntax.* Internet Engineering Task Force, Agosto 1998. RFC2396.

[IET99] IETF. *Hypertext Transfer Protocol - HTTP/1.1.* Internet Engineering Task Force, Junho 1999. RFC2616.

[ILOa] ILOG. http://www.ilog.com.

[ILOb] ILOG. ILOG JRules. http://www.ilog.com/products/rules/engines/jrules.

[JF88] Ralph E. Johnson and Brian Foote. Designing Resuseable Classes. *Journal of Object-Oriented Programming*, 1(2):22–25, 1988.

[Kin97] Charlie Kindel. ActiveX and the Web — Architecture and Technical Overview, 1997. Architect & Chief Technical Advisor Microsoft Developer Relations Group. Disponível em http://www.microsoft.com/com/presentations/.

[LB02] Jonathan Lurie and R. Jason Belanger. The Great Debate: .Net vs. J2EE, Março 2002. Disponível em http://www.javaworld.com/javaworld/jw-0308-j2eenet_p.html.

[LFCH02] Simon Lok, Steven K. Feiner, William M. Chiong, and Yoav J. Hirsch. A Graphical User Interface Toolkit Approach to Thin-Client Computing. In *Proceedings of the eleventh international conference on World Wide Web*, pages 718–725. ACM Press, 2002.

[LSFH00] Sheng Feng Li, Quentin Stafford-Fraser, and Andy Hopper. Integrating Synchronous and Asynchronous Collaboration with Virtual Networking Computing. Technical Report 2000.1, AT&T Laboratories Cambridge, 24a Trumpington Street, Cambridge CB2 1QA, England, 2000. Proceedings of the First International Workshop on Intelligent Multimedia Computing and Networking, Atlantic City, USA, Vol. 2, Pages 717-721, February 27-March 3, 2000.

[Met] Citrix Metaframe. http://www.citrix.com/products/metaframe/.

[MF00] J. P. Martin-Flatin. *Web-Based Management of IP Networks and Systems.* PhD thesis, Escola Politécnica Federal de Lausanne, 2000.

[Mic03] Microsoft. Defining the Basics Elements of .Net, 2003. Disponível em http://www.microsoft.com/net/basics/whatis.asp.

[MP02] Steve Mohr Pieter Siegers Chris Knowles Michael Palermo, Darshan Singh. *Professional ASP.Net 1.0 XML with C#.* Wrox, 2002.

[MV01] Dave Marsh and Paulo Viegas. The BUS Architecture. In *Proceedings of the 2001 Porto Power Tech Conference*, Setembro 2001.

[Nie00] Jakob Nielsen. *Designing Web Usability: The Practice of Simplicity.* New Riders Publishing, 2000.

[Nie01] Txomin Nieva. *Remote Data Acquisition of Embedded Systems Using Internet Technologies: A Role-Based Generic System Specification.* PhD thesis, Escola Politécnica Federal de Lausanne, 2001.

[Ntr] Insignia Solutions Ntrigue. http://www.insignia.com.

[Ope] OpenMap. http://www.openmap.org.

[pESdES] Disponibilizado pela EFACEC Sistemas de Electrónica S.A. Specification for a SCADA/EMS/DMS - Technical Requirements. System Performance an Sizing.

[Pip99] George Pipkin. Java Servlets for Enterprise Applications. *virginia.edu*, III(1), 1999. Disponível em http://www.itc.virginia.edu/virginia.edu/spring99/forecast/home.html.

[Rap] Graphon RapidX. http://www.graphon.com.

[Rip01] Riptech Inc. *Understanding SCADA System Security Vulnerabilities*, Janeiro 2001.

[RR00] Johannes Gehrke Raghu Ramakrishnan. *Database Management Systems.* McGraw-Hill International Editions, 2000.

[Sam95] Hanan Samet. Spatial Data Structures, 1995. Computer Science Department and Institute of Advanced Computer Studies and Center for Automation Research.

[SG99] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline.* Prentice Hall, 1999.

[SLN99] Brian K. Schmidt, Monica S. Lam, and J. Duane Northcutt. The Interactive Performance of SLIM: a Stateless, Thin-Client Architecture. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, volume 34, pages 32–47, dec 1999.

[SM02a] Humphrey Sheil and Michael Monteiro. Rumble in the Jungle: J2EE versus .Net, Part1, 2002. Disponível em http://www.javaworld.com/javaworld/jw062002/jw0628j2eevsnet_p.html.

[SM02b] Stephen Stelting and Olav Maassen. *Applied Java Patterns.* Prentice Hall, 2002.

[SS96] Valerie Quercia Stephen Spainhour. *Webmaster in a Nutshell.* O'Reilly, 1996.

[Sun] Sun Microsystems. *Java 2 Platform, Standard Edition, v 1.3.1 API Specification.* Disponível em http://java.sun.com/j2se/1.3/docs/api/.

[Tar] SCO Tarantella. http://www.tarantella.sco.com.

[Tec] Yasu Technologies. QuickRules 2.5 Beta. http://www.yasutech.com/products/quickrules/festures.htm.

[Tre99] Andrew Treloar. Just another technology? How the dynamics of innovation can help predict the future of the browser. *AwsWeb99*, 1999. Disponível em http://ausweb.scu.edu.au/aw99/papers/treloar1.

[W3C] W3C. *The Extensible Stylesheet Language (XSL).* World Wide Web Consortium. Disponível em http://www.w3c.org/Style/XSL.

[W3C98] W3C. *Cascading Style Sheets, level 2 — CSS2 Specification.* World Wide Web Consortium, 1998. Disponível em http://www.w3c.org/TF/REC-CSS2.

[W3C99a] W3C. *HTML 4.01 Specification.* World Wide Web Consortium, 1999. Disponível em http://www.w3c.org/TR/html4.

[W3C99b] W3C. *Resource Description Framework (RDF) Model and Syntax Specification.* World Wide Web Consortium, 1999. Disponível em http://www.w3c.org/TR/1999/REC-rdf-syntax-19990222.

[W3C99c] W3C. *XML Path Language (XPath) — Version 1.0.* World Wide Web Consortium, 1999. Disponível em http://www.w3c.org/TR/xpath.

[W3C99d] W3C. *XSL Transformations (XSLT) — Version 1.0.* World Wide Web Consortium, 1999. Disponível em http://www.w3c.org/TR/xslt.

[W3C00] W3C. *Extensible Markup Language (XML) 1.0 (Second Edition).* World Wide Web Consortium, 2000. Disponível em http://www.w3c.org/TR/REC-xml.

[W3C01a] W3C. *Extensible Stylesheet Language (XSL) — Version 1.0.* World Wide Web Consortium, 2001. Disponível em http://www.w3c.org/TR/xsl.

[W3C01b] W3C. *Scalable Vector Graphics (SVG) 1.0 Specification.* World Wide Web Consortium, Setembro 2001. Disponível em http://www.w3c.org/TR/SVG11.

[Wie01] Ian Wiese. The Integration of SCADA and Corporate IT. Technical report, Water Corporation of Western Australia, 2001.

[Yan02] Kevin Yank. Web Services Demystified, 2002. Disponível em http://www.webmasterbase.com/printTemplate.php?aid=692.

[ZPMD97] Deborra J. Zukowski, Apratim Purakayastha, Ajay Mohindra, and Murthy Devarakonda. Metis: A Thin-Client Application Framework. In *Proceedings of the 3rd Conference on Object-Oriented Technologies and Systems*, pages 103–114, Berkeley, June 16–20 1997. Usenix Association.