



Universidade do Porto

Faculdade de Engenharia

FEUP



Rogério Dias Paulo

Ferramentas de Engenharia para Apoio ao Projecto de Sistemas Conduzidos por Eventos

- Aplicação ao Domínio da Automação
de Subestações de Energia Eléctrica

Rogério Dias Paulo

Ferramentas de Engenharia para Apoio ao Projecto de Sistemas Conduzidos por Eventos

Aplicação ao Domínio da Automação de Subestações de Energia Eléctrica

Tese submetida à Faculdade de Engenharia da Universidade do Porto para satisfação parcial dos
requisitos de grau de mestre em Engenharia Electrotécnica e de Computadores
(Área de Especialização em Informática e Automação)

Tese realizada sobre supervisão do

Professor Doutor Adriano Silva de Carvalho

Professor Associado com Agregação da Faculdade de Engenharia da Universidade do Porto

Departamento de Engenharia Electrotécnica e de Computadores
Faculdade de Engenharia da Universidade do Porto

Abril de 2004

621.3(043)/PAUa/FER

UNIVERSIDADE DO PORTO	
Faculdade de Engenharia	
BIBLIOTECA M	
N.º	73417
CDU	621.3(043)
Data	13.4.2005

à Cristina

aos meus Pais

Resumo

O sistema eléctrico de energia tem vindo a sofrer recentemente uma evolução acelerada que motiva o aparecimento da terceira geração de sistemas de automação, caracterizada pela abrangência, diversidade e agilidade crescentes das funções de controlo automático. As funções do sistema de automação são suportadas, em todos os níveis funcionais, por arquitecturas distribuídas com sólida integração (i) comunicativa entre dispositivos, (ii) sistémica entre funções e (iii) de projecto entre ferramentas e plataformas de automação. Níveis elevados de integração resultam da colocação em prática de estratégias que visam o incremento da eficiência e eficácia do sistema eléctrico e que promovem também o uso de ferramentas de engenharia capazes de suportar de forma coesa o projecto de sistemas de automação.

O trabalho de tese descrito neste documento compreende o estudo de uma solução para ferramentas de engenharia tendo em vista o suporte à actividade de concepção e desenvolvimento de sistemas de automação para instalações do sistema eléctrico de energia, com particular enfoque nas subestações. Esta dissertação descreve uma linguagem para modelação, simulação e *deployment* de sistemas de automação, bem como orientações para a implementação de ferramentas de engenharia baseadas nessa linguagem.

Os sistemas de automação de instalações são caracterizados segundo as perspectivas funcional, de projecto, dos dispositivos e de parametrização. Estas perspectivas condicionam os requisitos da ferramenta de engenharia, que deverá integrar funcionalidades de desenho e validação de modelos, bem como a sua documentação e *deployment*.

O projecto conduzido por modelos é, pela sua eficiência, uma abordagem de eleição para o desenvolvimento de sistemas de automação. Esta abordagem ao projecto é viável no âmbito de aplicação deste trabalho, dado o aparecimento de modelos sistémicos normalizados a nível internacional, dos quais se destacam os que integram a norma IEC 61850. É proposta neste trabalho, como elemento fundamental de uma ferramenta para suporte ao projecto conduzido por modelos, uma linguagem de modelação orientada ao objecto capaz de suportar o desenho das componentes estática, dinâmica discreta e de interface ao utilizador. Esta linguagem, denominada linguagem de modelação para sistemas de automação (ASML), integra elementos da UML, de redes de Petri, de linguagens IEC 61131-3, bem como elementos de modelação de gráficos interactivos.

A ASML permite a implementação económica de ferramentas de engenharia flexíveis, que pode ser levada a cabo segundo as orientações propostas neste documento. Apresenta-se, como solução de implementação, uma arquitectura com características modulares e baseada em meta-modelos e padrões.

Ferramentas de engenharia construídas segundo esta abordagem possibilitarão o desenvolvimento integrado de sistemas de automação de forma eficaz e contribuirão para controlar o sistema eléctrico do futuro, de forma segura, flexível, ágil e eficiente.

Abstract

Electrical power systems have been undergoing significant changes which have motivated the development of third generation automation systems. This new generation is characterized by growing levels of diversity, agility and broadness of automatic control related functions. The automation system functions are supported, at all functional levels, by distributed architectures exhibiting solid (i) communication integration between devices, (ii) system integration between functions and (iii) project integration between tools and automation platforms. These high levels of integration are the result of strategies envisaging an overall increase in the efficiency and efficacy of power systems. Such strategies also promote the systematic use of engineering tools capable of coherently supporting the development of automation systems.

The thesis herein described involves the study of an engineering tools solution for supporting the design and the implementation of automation systems for electrical power systems with main focus on the substation. This dissertation describes a language for modeling, simulating and deploying automation systems, as well as guidance towards the implementation of tools based on this language.

Station automation systems are characterized from the functional view, the project view, the device view and the parameterization view. These perspectives outline the requirements of an engineering tool, which should be aimed at systems design and validation, as well as documentation and deployment.

Given its efficiency, model-driven engineering is the chosen approach towards automation systems development. Model-driven development is viable in this engineering domain since system models accepted as international standards are now available, such as those proposed by the IEC 61850 standard series. This document encloses a proposal for a language aimed at model-driven development, viewed as the fundamental element of an engineering tool. The language, named Automation Systems Modeling Language (ASML), integrates elements from known languages, such as UML, Petri nets and IEC 61131-3, and also interactive display modeling elements. This object-oriented language is capable of supporting the design of automation systems major logical components: static structure, discrete event dynamics and user interface.

The ASML allows the economic implementation of flexible engineering tools that may be carried out in accordance with the proposals included in this document. A modular meta-model and pattern based software architecture for the development of engineering tools is described.

Engineering tools based on the approach of this thesis would allow the integrated development of automation systems and would present a contribution towards the safe, flexible, agile and efficient control of the power system of the future.

Agradecimentos

Gostaria de agradecer

ao Professor Doutor Adriano Silva de Carvalho, supervisor deste trabalho, pelo seu apoio e disponibilidade durante o desenvolvimento da tese,

ao Engenheiro António Carrapatoso, pelo suporte demonstrado a este projecto e pelo incentivo e orientação ao longo do meu percurso profissional,

aos colegas da EFACEC Sistemas de Electrónica, S.A., particularmente ao Engenheiro Fernando Gomes, ao Paulo Rodrigues, ao Fernando Xavier, ao Pedro Simões e ao Manuel Dias, pelo ambiente de trabalho e apoio proporcionados,

à EFACEC Sistemas de Electrónica, S.A. e à Faculdade de Engenharia da Universidade do Porto, pela disponibilização de recursos,

a todos os Amigos, que sempre me incentivaram e que de forma directa ou indirecta contribuíram para este projecto,

à Cristina por toda a paciência e ajuda, e

aos meus Pais, de forma muito especial, por me compreenderem e apoiarem todos os meus projectos.

Índice Geral

Resumo	i
Abstract	ii
Agradecimentos	iii
Índice Geral	iv
Índice de Figuras	vii
Lista de Tabelas	x
Lista de Acrónimos	xii
Parte I Sistemas de Automação e Ferramentas de Engenharia.....	1
<i>Capítulo 1 Introdução</i>	3
1.1. Âmbito e Objectivos do Trabalho.....	3
1.2. Estrutura da Dissertação	4
<i>Capítulo 2 Caracterização</i>	5
2.1. Sobre o Controlo do Sistema Eléctrico de Energia.....	5
2.2. Os Sistemas de Automação de Instalações do Sistema Eléctrico de Energia	8
2.3. Modelo do Sistema de Automação	17
2.4. Sobre as Ferramentas de Engenharia.....	20
Parte II Linguagem de Modelação.....	29
<i>Capítulo 1 Introdução</i>	31
1.1. Linguagens de Programação para Sistemas de Automação.....	31
1.2. Linguagens Orientadas ao Objecto	32
1.3. Modelação Comportamental.....	33
1.4. Representação Gráfica e Interface ao Utilizador	34
1.5. Introdução à Linguagem de Modelação para Sistemas de Automação	35
<i>Capítulo 2 Sintaxe Abstracta e Semântica</i>	37
2.1. Modelo Reduzido	37
2.2. Arquitectura.....	38
2.3. Sobre a Definição da Sintaxe Abstracta e da Semântica da Linguagem	40
2.4. Visão Geral do Meta-Modelo	40
2.5. Modelo Fundamental	41
2.6. Gestão de Modelos.....	62
2.7. Elementos de Comportamento	64
2.8. Elementos de Interação	87
2.9. Biblioteca de Operações Comuns	103
<i>Capítulo 3 Notação</i>	105

3.1. Visão Geral	105
3.2. Elementos Gráficos Comuns	106
3.3. Classificadores.....	107
3.4. Instâncias.....	109
3.5. Redes de Petri	110
3.6. Expressões e Procedimentos	111
Capítulo 4 Semântica de Domínio para Sistemas de Automação.....	114
4.1. Visão Geral e Relação com o Sistema de Automação e a Ferramenta	114
4.2. Sistema Primário.....	115
4.3. Sistema Lógico	117
4.4. Sistema Secundário.....	118
4.5. Sobre a Semântica para Automação de Sistemas de Energia.....	120
Capítulo 5 Conclusões.....	122
5.1. A ASML e os Sistemas de Automação	122
5.2. Rastreio de Requisitos Funcionais	123
5.3. Evolução Futura da ASML	124
5.4. Síntese.....	125
Parte III Estratégias de Implementação.....	127
Capítulo 1 Sobre a Implementação da Ferramenta.....	129
1.1. Princípios Arquitecturais	129
1.2. Interface ao Utilizador	131
1.3. Repositório de Modelos	135
1.4. Experiências de Implementação Conduzidas no Âmbito do Trabalho de Tese	139
1.5. Conclusões	140
Parte IV Conclusões.....	141
Capítulo 1 Sumário, Resultados e Considerações Finais	143
1.1. Sumário de Resultados.....	144
1.2. Conclusões	145
1.3. Considerações Finais	146
1.4. Orientações para Trabalho Futuro.....	146
Referências	I
Anexo I. IEDs Comuns.....	VII
Anexo II. Funções do Sistema de Automação	IX
Anexo III. Modelo do Sistema de Automação	XIII
Anexo IV. Gramática para Expressões e Procedimentos ASML	XVIII
Anexo V. Elementos Textuais da Notação Gráfica ASML	XXVII
Anexo VI. Gramática para Definição de Caminhos	XXX
Anexo VII. Modelo da Notação Gráfica ASML	XXXIII
Anexo VIII. Classes de Dados Comuns.....	XXXVII

Anexo IX. Modelos dos Protocolos IEC 60870-5-101/104..... XL

Anexo X. Exemplo de Modelo de um IED..... XLII

Índice de Figuras

Figura 1.	Níveis de controlo no sistema eléctrico de energia.	6
Figura 2.	Modelo de dispositivos.....	9
Figura 3.	Modelo genérico de um IED.	10
Figura 4.	Modelo de base de dados de um IED.....	11
Figura 5.	Modelo fundamental de níveis funcionais e de interfaces lógicas.	11
Figura 6.	Modelo para classificação de parametrizações.....	14
Figura 7.	Ciclo de vida de um sistema de automação.	15
Figura 8.	Fases de um projecto de desenvolvimento de um sistema de automação.	16
Figura 9.	Vistas do modelo do sistema de automação.	18
Figura 10.	Modelo lógico do sistema de automação e sua relação com o sistema secundário.	19
Figura 11.	Modelo fundamental de uma função no sistema de automação.	20
Figura 12.	Requisitos funcionais.....	23
Figura 13.	Requisitos para construção de modelos.....	24
Figura 14.	Requisitos para validação de modelos.	25
Figura 15.	Requisitos para parametrização de IEDs.	26
Figura 16.	Requisitos para geração de documentação.	27
Figura 17.	Modelo geral da ASML e suas relações com linguagens existentes.....	36
Figura 18.	Modelo reduzido da sintaxe abstracta.	37
Figura 19.	Meta-modelo da linguagem.	41
Figura 20.	Modelo fundamental da linguagem - <i>Foundation</i>	41
Figura 21.	Modelo base do pacote <i>Core</i>	42
Figura 22.	Elementos adicionais ao modelo base do pacote <i>Core</i>	51
Figura 23.	Concretizações de classificadores do pacote <i>Core</i>	51
Figura 24.	Tipos primitivos da linguagem.	54
Figura 25.	Relações do pacote <i>Core</i>	55
Figura 26.	Pacote <i>DataTypes</i>	58
Figura 27.	Pacote <i>Extension Mechanisms</i>	59
Figura 28.	Gestão de modelos - <i>Model Management</i>	62
Figura 29.	Elementos de Comportamento – <i>Behavioural Elements</i>	64
Figura 30.	Ações e eventos do pacote <i>Common Behaviour</i>	65
Figura 31.	Elemento de modelação de comportamento assíncrono elementar.....	67
Figura 32.	Instâncias do pacote <i>Common Behaviour</i>	68
Figura 33.	Ligações do pacote <i>Common Behaviour</i>	69
Figura 34.	Modelo do pacote <i>Petri Nets</i>	70

Figura 35. Modelo base do pacote *Methods*..... 75

Figura 36. Instruções de controlo do pacote *Methods*..... 78

Figura 37. Instrução de controlo *Case*..... 81

Figura 38. Expressões do pacote *Methods*..... 82

Figura 39. Operadores do pacote *Methods*..... 83

Figura 40. Modelo de elementos de interacção da ASML..... 87

Figura 41. Modelo do pacote *Interaction Elements*..... 87

Figura 42. Modelo do pacote *Foundation*..... 87

Figura 43. Modelo do pacote *Core*..... 88

Figura 44. Modelo do pacote *Signals*..... 92

Figura 45. Modelo do pacote *Geometric*..... 93

Figura 46. Modelo do pacote *Shapes*..... 93

Figura 47. Modelo do pacote *Styles*..... 96

Figura 48. Modelo do pacote *Animation*..... 98

Figura 49. Modelo do pacote *Widgets*..... 101

Figura 50. Pacote *Standard Library*..... 103

Figura 51. Modelo de operações da *Standard Library*..... 104

Figura 52. Modelo da notação da linguagem – pacote *Notation*..... 105

Figura 53. Pacote *Notation*..... 106

Figura 54. Anotações..... 106

Figura 55. Três representações alternativas de um mesmo pacote..... 107

Figura 56. Classificador..... 107

Figura 57. Classificadores..... 108

Figura 58. Relações..... 108

Figura 59. Notação base para instâncias..... 109

Figura 60. Uma notação alternativa para instâncias..... 110

Figura 61. Redes de Petri..... 110

Figura 62. Expressão..... 112

Figura 63. Procedimento..... 113

Figura 64. Visão geral da semântica para automação de sistemas de energia..... 114

Figura 65. Arquitectura da ferramenta..... 115

Figura 66. Modelo ASML para modelação do sistema primário..... 116

Figura 67. Regras sintácticas para modelação do sistema primário..... 116

Figura 68. Um exemplo simples de modelação de dispositivos primários..... 117

Figura 69. Estereótipos e interfaces para modelação do sistema lógico..... 117

Figura 70. Regras sintácticas para modelação do sistema lógico..... 118

Figura 71. Estereótipos para modelação do sistema secundário..... 119

Figura 72. Sintaxe para modelação do sistema secundário..... 120

Figura 73. Arquitectura de três camadas..... 130

Figura 74. Arquitectura modular para implementação de ferramentas de engenharia..... 130

Figura 75. Exemplo de um ambiente IDE. 131

Figura 76. Modelo para o ambiente de interface ao utilizador de um IDE. 132

Figura 77. Arquitectura MVC aplicada à interface ao utilizador de um IDE. 133

Figura 78. Modelo simplificado de nós e acções para implementação do controlador da arquitectura MVC..... 134

Figura 79. Modelo geral do editor gráfico. 135

Figura 80. Modelo do repositório e relações com componentes das camadas funcional e de interface..... 136

Figura 81. Meta-meta-modelo para o repositório..... 137

Figura 82. Interface da API de acesso a modelos. 138

Figura 83. Modelo para a camada de suporte a editores. 139

Figura 84. Vista da plataforma para IDEs implementada em C++..... 140

Figura 85. Sistema de automação. XIII

Figura 86. Sistema físico..... XIV

Figura 87. Equipamento primário vulgarmente encontrado em subestações..... XIV

Figura 88. Equipamento secundário. XV

Figura 89. IED..... XV

Figura 90. Base de dados de IED..... XVI

Figura 91. Sistema lógico..... XVI

Figura 92. Função..... XVII

Figura 93. Parametrização..... XVII

Figura 94. Elementos comuns da notação gráfica (*Link, Note*). XXXIII

Figura 95. Elementos comuns da notação gráfica (*Stereotype, Package*). XXXIV

Figura 96. Representação de classificadores. XXXIV

Figura 97. Representação de objectos. XXXV

Figura 98. Representação de redes de Petri. XXXV

Figura 99. Representação de expressões e procedimentos.....XXXVI

Figura 100. Classes de dados elementares para definição de classes de dados comuns.....XXXVII

Figura 101. Classes de dados comuns para informação de estado.....XXXVIII

Figura 102. Sinais do modelo de controlo.....XXXVIII

Figura 103. Classes de dados comuns para informação de controlo.....XXXIX

Figura 104. Modelo de dados para os protocolos IEC 60870-5-101/104. XL

Figura 105. Modelo para servidores IEC 60870-5-101/104.....XLI

Figura 106. Modelo para clientes IEC 60870-5-101/104. XLI

Figura 107. Exemplo de um modelo simples de um IED.....XLII

Lista de Tabelas

Tabela 1.	Classificação funcional convencional dos sistemas de controlo do sistema eléctrico.....	5
Tabela 2.	Classificação em áreas funcionais do sistema de informação do sistema eléctrico.....	7
Tabela 3.	Interfaces lógicas presentes num sistema de automação.....	12
Tabela 4.	Lista de funções presentes num sistema de automação de instalações do sistema eléctrico.....	13
Tabela 5.	Classes de ferramentas.....	21
Tabela 6.	Características chave do modelo semântico da ASML.....	38
Tabela 7.	Arquitectura de quatro camadas para meta-modelos.....	39
Tabela 8.	Regras para referências a elementos na definição da ASML.....	40
Tabela 9.	Tipos primitivos da linguagem.....	53
Tabela 10.	Elementos referenciáveis em instruções.....	77
Tabela 11.	Operadores unários.....	84
Tabela 12.	Operadores de conversão de tipos primitivos.....	85
Tabela 13.	Operadores binários.....	86
Tabela 14.	Simbologia para relações.....	109
Tabela 15.	Semântica para o modelo do sistema secundário.....	119
Tabela 16.	Rastreio de requisitos da ASML para o desenho de sistemas de automação.....	121
Tabela 17.	Rastreio de requisitos da ASML.....	123
Tabela 18.	Elementos da ASML passíveis de trabalho futuro.....	124
Tabela 19.	A API de acesso a modelos e a sua relação com o meta-meta-modelo.....	138
Tabela 20.	Sumário de resultados do trabalho.....	144
Tabela 21.	IEDs com funções essencialmente comunicativas.....	VII
Tabela 22.	IEDs essencialmente associados à interface com o utilizador.....	VII
Tabela 23.	IEDs essencialmente associados ao processo.....	VIII
Tabela 24.	Funções básicas.....	IX
Tabela 25.	Funções de controlo automático comuns.....	X
Tabela 26.	Funções de suporte.....	XII
Tabela 27.	Símbolos não terminais.....	XVIII
Tabela 28.	Expressões.....	XIX
Tabela 29.	Símbolos terminais para operadores ordenados por ordem crescente de precedência.....	XXI
Tabela 30.	Instruções de controlo.....	XXII
Tabela 31.	Referências.....	XXIII
Tabela 32.	Símbolos terminais para valores constantes de tipos primitivos.....	XXIV
Tabela 33.	Restantes símbolos terminais.....	XXIV
Tabela 34.	Caracteres especiais para uso em constantes do tipo <i>String</i>	XXVI

Tabela 35. Sintaxe e léxico para elementos textuais da notação gráfica da linguagem. XXVII

Tabela 36. Sintaxe para definição de caminhos. XXX

Tabela 37. Léxico para definição de caminhos.....XXXII

Lista de Acrónimos

A/D	<i>Analógico/Digital</i>
AC	<i>Alternating Current</i>
API	<i>Application Programming Interface</i>
ASML	<i>Automation Systems Modeling Language</i>
BNF	<i>Backus Naur Form</i>
CAD	<i>Computer-Aided Design</i>
COS	<i>Component Off-The-Shelf</i>
CPN	<i>Coloured Petri Nets</i>
CRM	<i>Customer Relationship Management</i>
DC	<i>Direct Current</i>
DNP	<i>Distributed Network Protocol</i>
EMF	<i>Eclipse Modeling Framework</i>
ERP	<i>Enterprise Resource Planning</i>
FAT	<i>Factory Acceptance Tests</i>
FBD	<i>Function Block Diagram</i>
FSM	<i>Finite State Machine</i>
GDI	<i>Graphics Device Interface</i>
GEF	<i>Graphical Editing Framework</i>
HMI	<i>Human-Machine Interface</i>
I/O	<i>Input/Output</i>
IDE	<i>Integrated Development Environment</i>
IEC	<i>International Electrotechnical Commission</i>
IED	<i>Intelligent Electronic Device</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
IP	<i>Internet Protocol</i>
MDA	<i>Model-Driven Architecture</i>
MDR	<i>Metadata Repository</i>
MFC	<i>Microsoft Foundation Classes</i>

MOF	<i>Meta-Object Facility</i>
MVC	<i>Model View Controller</i>
OCL	<i>Object Constraint Language</i>
OMG	<i>Object Management Group</i>
PC	<i>Personal Computer</i>
PLC	<i>Programmable Logic Controller</i>
RdP	<i>Rede de Petri</i>
RTNA	<i>Real-Time Network Analysis</i>
SA	<i>Sistema de Automação</i>
SAT	<i>Site Acceptance Tests</i>
SCADA	<i>Supervisory Control and Data Acquisition</i>
SCL	<i>Substation Configuration Language</i>
SFC	<i>Sequential Function Chart</i>
ST	<i>Structured Text</i>
SVG	<i>Scalable Vector Graphics</i>
TCP	<i>Transmission Control Protocol</i>
UML	<i>Unified Modeling Language</i>
URT	<i>Unidade Remota Terminal</i>
WAP	<i>Wide Area Protection</i>
WYSIWYG	<i>What You See Is What You Get</i>
XMI	<i>XML Metadata Interchange</i>
XML	<i>Extensible Markup Language</i>

Parte I

Sistemas de Automação e Ferramentas de Engenharia

Introdução

Enquadramento, âmbito, objecto e objectivos do trabalho

Estrutura da dissertação

Caracterização

Automação do sistema eléctrico de energia, modelos convencionais, situação actual e tendências

Modelo do sistema de automação na perspectiva das ferramentas de engenharia

Requisitos para ferramentas de engenharia

Justificação da estratégia de trabalho adoptada

Conclusões da primeira parte

Capítulo 1

Introdução

O estado da arte dos sistemas de controlo do sistema eléctrico de energia apresenta graus de sofisticação crescentes. Esta sofisticação decorre da reestruturação do sector eléctrico, associada a necessidades de aumento de desempenho e eficiência, de controlo de qualidade no fornecimento energético, de redução dos custos totais dos ciclos de vida e de aumento da flexibilidade e agilidade.

Estes factores têm sido motores do aparecimento de uma nova geração de sistemas, a considerada terceira geração de sistemas de automação. Esta geração de sistemas é marcada por um incremento na abrangência e relevância das funções de controlo automático e por uma componente comunicativa ubíqua e transparente. A quase totalidade dos dispositivos está associada ao sistema de informação da *utility*, independentemente da sua dimensão, funcionalidade ou nível de controlo, e os variados fluxos de informação coexistem e suportam soluções funcionais baseadas em arquitecturas cooperativas. A sustentabilidade destes sistemas assenta em mecanismos que permitem não só a consistência de informação de estado, controlo e gestão entre subsistemas interligados, mas também a consistência de meta-informação. Neste quadro, assiste-se a uma movimentação dos esforços de normalização técnica para o suporte à veiculação de meta-informação, sendo hoje o enfoque principal dessas actividades a definição de modelos de informação e suas representações, movendo o cerne da interoperabilidade da vertente comunicativa para a vertente sistémica.

A actividade de concepção e desenvolvimento de sistemas de automação tem sido sempre suportada por ferramentas de engenharia, com o objectivo da redução de custos e aumento da qualidade. Esta vertente é particularmente importante na passagem para a terceira geração de sistemas de automação e o seu desenvolvimento, em paralelo com o das plataformas de automação, constitui motor de incremento significativo da eficácia e eficiência da globalidade da actividade da engenharia de automação.

1.1. Âmbito e Objectivos do Trabalho

O trabalho apresentado neste documento tem como objecto de estudo as ferramentas de engenharia para o apoio à actividade de concepção e desenvolvimento de sistemas de automação de terceira geração para instalações do sistema eléctrico de energia. O trabalho tem enfoque particular nas fases de análise, desenho e integração do projecto de sistemas de automação.

Quanto ao enquadramento no sistema eléctrico de energia, este estudo compreende os sistemas de automação de instalações como sejam as subestações, o principal enfoque do trabalho, mas também as centrais de produção eléctrica ou os parques eólicos. Dada a intercepção funcional e conceptual poderá revelar-se também parcialmente aplicável a outros elementos do sistema eléctrico ou outros domínios da engenharia de automação.

Desta forma os objectivos deste trabalho são:

- caracterizar os sistemas de automação de instalações do sistema eléctrico de energia;
- ~~encontrar~~ ^{propor} uma solução eficaz para ferramentas de engenharia;
- estudar a implementação dessas ferramentas.

Esta dissertação descreve uma linguagem para modelação, simulação e deployment de sistemas de automação, bem como orientações para a implementação de ferramentas para manipulação dessa linguagem.

1.2. Estrutura da Dissertação

Este documento é constituído por nove capítulos agrupados em quatro partes. A primeira parte, introdutória, compreende dois capítulos. No primeiro capítulo faz-se referência ao âmbito do trabalho e seus objectivos. O segundo capítulo apresenta de forma detalhada o enquadramento do trabalho, caracterizando-se o objecto de estudo na perspectiva das ferramentas de engenharia. Neste capítulo são caracterizados os sistemas de automação do sistema eléctrico de energia, apresentados os requisitos para ferramentas de engenharia e adoptada uma abordagem fundamental para o desenvolvimento do trabalho.

Na segunda parte, composta por cinco capítulos, é apresentada uma proposta de linguagem de modelação para suporte às actividades de engenharia de automação, como elemento fundamental de uma ferramenta. No primeiro capítulo analisam-se as linguagens disponíveis com aplicação neste domínio e justifica-se a adopção do modelo de linguagem proposto. O segundo capítulo compreende uma introdução ao modelo e a definição da semântica e da sintaxe abstracta da linguagem. A definição léxica e a sintaxe concreta são apresentadas no capítulo três. O quarto capítulo da segunda parte apresenta a particularização da linguagem para o domínio da automação de instalações do sistema eléctrico, bem como algumas considerações sobre a sua utilização. O último capítulo apresenta as conclusões da segunda parte enquadradas numa avaliação sumária do modelo proposto.

A terceira parte compreende a descrição de uma proposta de arquitectura e de opções de implementação para ferramentas de engenharia, adoptadas parcialmente em experiências de implementação no âmbito deste trabalho.

Na quarta e última parte apresentam-se as conclusões, orientações para trabalho futuro e considerações finais.

Capítulo 2

Caracterização

2.1. Sobre o Controlo do Sistema Eléctrico de Energia

2.1.1. Os Modelos Convencionais

O sistema eléctrico de energia é um dos sistemas industriais controlados com maior dimensão e a sua concepção, operação e manutenção envolve directamente múltiplas áreas da engenharia. O controlo do sistema eléctrico, entendido no sentido lato, é consequentemente também diversificado e envolve vários domínios. A classificação convencional dos vários sistemas de controlo presentes no sistema eléctrico de energia compreende uma classificação funcional, apresentada na tabela 1, e a conhecida pirâmide de níveis controlo, apresentada na figura 1.

Tabela 1. Classificação funcional convencional dos sistemas de controlo do sistema eléctrico.

Classe	Descrição
Protecção e Controlo	Funções que envolvem essencialmente a monitorização das grandezas fundamentais de operação dos equipamentos primários ou da rede (como correntes, tensões, potências, vibração ou temperaturas) e que garantem que estas se mantêm dentro de gamas de operação seguras e eficientes. Os exemplos típicos de funções desta classe compreendem, entre outros, o controlo de disjuntor, a protecção de linha ou de transformador ou o controlo de grupos geradores. Compreendem essencialmente acções de controlo local de um dispositivo (disjuntor, transformador, alternador, etc.) e acções de protecção convencional.
Automação	Funções de controlo automático. Estas funções abrangem essencialmente o equipamento primário, como por exemplo, a colocação em serviço de subsistemas do sistema eléctrico (reposição de linhas após actuação de protecção, arranque de grupos geradores). Estas funções envolvem essencialmente a automação local a uma instalação, essencialmente caracterizadas por sequências de operações condicionadas e geridas ao nível de centros de operação de rede ou ao nível de instalação.
Monitorização, Controlo e Supervisão	As funções de monitorização e supervisão abrangem a aquisição de dados instantâneos (como a configuração da rede ou o estado operativo de equipamentos) ou estatísticos (como as transferências de energia ou o histórico de operação de equipamentos) e a sua preparação para análise em tempo real (operação do sistema) ou diferida (análise após defeito). O controlo de supervisão engloba a operação de dispositivos e sistemas, de forma localizada, ou da rede, de forma abrangente. Envolve nomeadamente a reconfiguração de rede (planeada ou após incidente), essencialmente através de operadores humanos em centros de operação e gestão de rede.

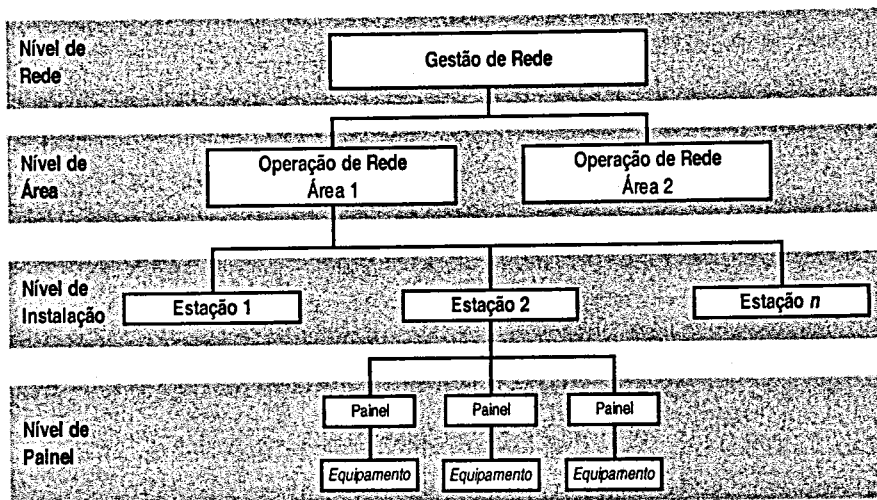


Figura 1. Níveis de controlo no sistema eléctrico de energia.

2.1.2. A Evolução do Sistema Eléctrico

O volume crescente de transacções de energia a grandes distâncias, os fluxos energéticos dotados de grande dinamismo, as exigências de qualidade de energia resultantes das características dos consumidores modernos e o rigor no controlo económico, características de sistemas regulados pelo mercado, introduzem pressões sobre o sistema eléctrico não previstas no seu planeamento inicial [1]. Ao sistema eléctrico são associadas necessidades de qualidade, segurança, disponibilidade, fiabilidade, agilidade e flexibilidade. A consequente reestruturação, hoje em curso, aponta para uma necessidade de incrementar o desempenho do controlo em todos os seus níveis: na qualidade do fornecimento de energia, na gestão dos recursos e nos custos totais de posse. Este incremento impõe, por um lado, novas pressões sobre os requisitos funcionais convencionais e, por outro, introduz novos requisitos para os sistemas de controlo, motivando a sua sofisticação.

A cooperação inter-funcional é uma tendência de evolução comum a todas as funções desempenhadas pelos sistemas de informação. Isto pressupõe um suporte comunicativo sólido, particularmente relevante se associarmos os novos modelos de gestão à perspectiva tecnológica de cooperação funcional. A opção por modelos descentralizados de gestão implica a coexistência de múltiplas entidades independentes na gestão e operação do sistema eléctrico: entidades reguladoras, entidades de gestão económica, entidades detentoras de instalações de produção ou redes de transporte e distribuição, operadores do sistema, organizações de manutenção, integradores de sistemas e fabricantes de equipamentos.

O acelerar da introdução de sistemas de produção distribuída, de elementos de armazenagem de energia, de sistemas flexíveis de transporte AC, de subestações e linhas blindadas, de sistemas DC e o aumento da produção baseada em energias renováveis, entre outros, vêm introduzir uma heterogeneidade tecnológica que promove a sofisticação e a diversidade dos sistemas de controlo.

Neste quadro, assiste-se a um aumento da relevância de funções não convencionais ^[2], como por exemplo (i) as protecções adaptativas ^[3], (ii) os esquemas de protecção distribuídos (WAP), (iii) a reconfiguração de rede e despacho automáticos ^[4], (iv) a monitorização de qualidade de energia, (v) o diagnóstico da operação de equipamentos, (vi) a manutenção local e remota, (vii) a análise de rede em tempo-real (RTNA), (viii) a gestão da segurança de pessoas, instalações e sistemas, (ix) o suporte a equipas de operação e manutenção ou (x) as aplicações de gestão de energia. O SCADA e controlo automático convencionais dão lugar a um sistema de informação integrado e flexível que compreende a integração de várias áreas funcionais (tabela 2).

Tabela 2. Classificação em áreas funcionais do sistema de informação do sistema eléctrico.

Classe	Descrição
Automação	Compreende as funções de automação, controlo e protecção associadas a subestações, centrais de produção eléctrica e redes de distribuição ou transporte. É caracterizada pelo controlo automático sem presença de operadores e suportada por plataformas distribuídas com índices elevados de interoperabilidade e de integração. Esta área funcional inclui, além da automação local da instalação, responsabilidades na automação da rede, passando a envolver, por exemplo, esquemas de automação de grande área ou o despacho automático.
Monitorização, Controlo e Supervisão	Trata-se da monitorização, controlo e supervisão (SCADA) convencionais que se caracterizam por monitorizar e controlar a operação do sistema, com vista à manutenção económica da relação produção/consumo. Envolve a resposta a incidentes, a execução de ordens de reconfiguração de rede, a análise da qualidade de energia ou o apoio a operações de intervenção <i>in loco</i> . Esta área funcional, intimamente associada ao processo, é suportada por sistemas de comando centralizados e distribuídos, fixos e móveis, unidades de interface ao processo e por uma infra-estrutura de comunicações de tempo-real que abrange o sistema eléctrico.
Suporte (Vigilância, Operações e Manutenção)	Apoio a equipas de intervenção para operações e manutenção. Diagnóstico e monitorização da operação do equipamento primário e do sistema de informação, para suporte à gestão da manutenção. Gestão da segurança e vigilância de instalações e equipamentos. Esta área funcional é suportada por sistemas de informação fixos e móveis, mas fortemente integrados com as duas áreas funcionais anteriores.
Gestão de Energia	Aplicações de suporte à decisão, essencialmente de médio e longo prazo, no âmbito da gestão e planeamento do sistema eléctrico, com vista à minimização de perdas de energia e maximização da utilização de recursos. Envolve, entre outros, a previsão de consumos, o cálculo eléctrico e a avaliação de cenários de exploração e contingências.
Mercado e Integração Empresarial	Análise de mercado, bolsa de energia, ERP, CRM, etc.

As novas pressões sobre o sistema eléctrico de energia, seu controlo e gestão levantam novos problemas e desafios tecnológicos à infra-estrutura mais crítica do final do século ^[1], um sistema de grande dimensão, multidisciplinar e de grande relevância para as sociedades modernas.

2.2. Os Sistemas de Automação de Instalações do Sistema Eléctrico de Energia

A automação de instalações segue naturalmente as tendências genéricas de evolução do sistema eléctrico. A automação localizada em subestações em particular, dados os incrementos funcionais e a sua posição privilegiada no sistema como elemento fonte de informação e meta-informação, torna-se uma área de relevância crescente no ambiente tecnológico heterogéneo do sistema eléctrico moderno.

Os sistemas de automação têm sido marcados recentemente por evoluções tecnológicas importantes. O aumento regular da capacidade de processamento dos dispositivos electrónicos, a par de um incremento muito significativo da largura de banda disponível em todas as interfaces comunicativas, possibilitam a implementação de arquitecturas distribuídas em todos os níveis funcionais. O aparecimento de sensores e actuadores inteligentes, baseados em princípios electro-ópticos e com interfaces de comunicações, torna as redes de comunicações do sistema de automação verdadeiramente ubíquas. Novos paradigmas, que compreendem funcionalidades de controlo distribuído da rede eléctrica, tomados possíveis pela disseminação de redes de comunicações de grande área, vêm suscitar a integração inter-instalação, a par da integração local e da integração com centros de operação.

O carácter autónomo dos sistemas de automação, bem como a sua composição multi-dispositivo, torna a integração coesa particularmente relevante. A este nível valoriza-se quer a facilidade de reconfiguração quer a interoperabilidade de dispositivos e sistemas. A importância da integração revela-se na concentração crescente sobre um número reduzido de perfis comunicativos ^[5] com suporte normativo internacional e aberto e em novos esforços de normalização sobre modelos sistémicos. A integração entre dispositivos ao nível da transferência de informação é, neste momento, assegurada por normas com uma base de aplicação muito alargada, como sejam os protocolos da família IEC 60870-5, o protocolo DNP ou a pilha protocolar TCP/IP. Neste contexto, os esforços de facilitação da integração passam para níveis de abstracção superiores. A terceira geração de sistemas de automação toma forma nas actividades de normalização electrotécnica, das quais se destaca o trabalho do TC57 na série de normas IEC 61850 ^[6]. O âmbito de aplicação desta série de normas compreende as redes de comunicação e os sistemas de automação de subestações, embora outras iniciativas alarguem o seu âmbito, como o caso da norma IEC 61400-25 ^[7] aplicável a parques eólicos.

A IEC 61850 ^[8] inclui, ao nível dos modelos de serviços, os modelos tradicionais de transferência de informação genérica ou de processo, modelos para transferência de informação de arquivo e parametrização funcional, bem como de meta-informação, isto é, informação semântica sobre os modelos de dados e funções disponíveis. Quanto aos modelos de dados, a norma concretiza níveis de abstracção superiores aos encontrados nos modelos tradicionais baseados em pontos. Estes modelos, definidos segundo conceitos de orientação ao objecto, incluem o suporte dos dispositivos e funções presentes hoje na subestação, numa perspectiva comunicativa.

A terceira geração de sistemas de automação caracteriza-se assim por: (i) uma tendência para o controlo distribuído do sistema eléctrico, ou seja, para a horizontalização da estrutura hierárquica vertical de controlo, (ii) um enfoque

na integração sistémica, nos modelos de informação e de meta-informação, (iii) uma crescente diversidade, sofisticação e agilidade nos subsistemas tecnológicos e estratégias de controlo.

O incremento de funcionalidade dos sistemas de automação, a diversificação e sofisticação funcionais crescentes, o enfoque na versatilidade das plataformas e a relevância da integração dotam os sistemas de automação de características de parametrização e *software* intensivos. Com efeito, as necessidades de integração comunicativa alargada e transparente implicam que os dispositivos terminais assegurem a definição completa do modelo de informação e meta-informação. O projecto de um sistema de automação envolve assim a integração e parametrização de múltiplos dispositivos e sistemas de comunicações, bem como a definição de múltiplas funções automáticas. O projecto de um sistema de automação tem invariavelmente custos elevados de engenharia, ao nível da especificação, desenvolvimento, integração e validação de sistemas. Estes custos devem sofrer um desagravamento, não obstante as tendências actuais de evolução, pelo que a existência de ferramentas de engenharia para o apoio a esta actividade é um elemento chave na sua redução, bem como na promoção da qualidade do sistema final. Justifica-se assim a relevância e aplicabilidade do estudo de ferramentas de engenharia para apoio à concepção e desenvolvimento de sistemas de automação.

2.2.1. Dispositivos de Automação

O sistema de automação compreende a integração de dispositivos de controlo, que constituem o sistema secundário, e pela sua associação aos equipamentos controlados que constituem o sistema primário. A figura 2 apresenta este modelo do sistema de automação.

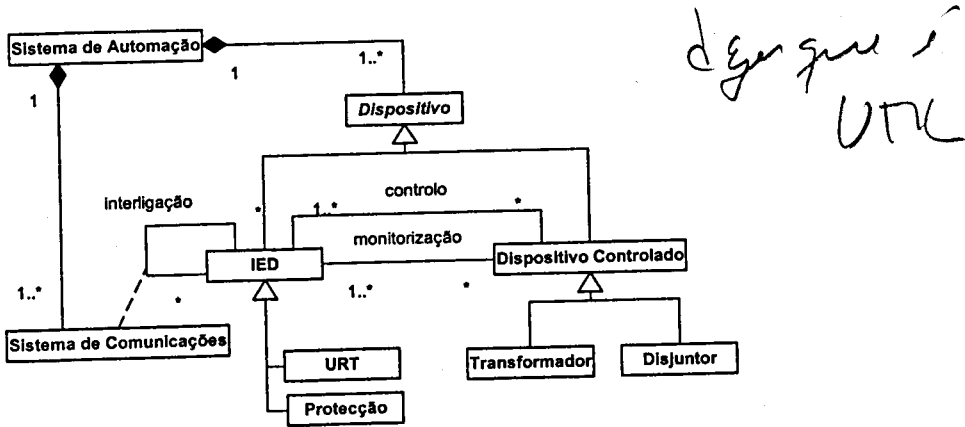


Figura 2. Modelo de dispositivos *na ótica de*

Os dispositivos presentes num sistema de automação, frequentemente em configurações redundantes, são genericamente denominados dispositivos electrónicos inteligentes (IED). Estes dispositivos constituem a plataforma de *deployment* de uma ferramenta de engenharia e podem ser caracterizados pelo modelo apresentado na figura 3. Os IEDs cooperam através de canais de comunicações e executam tarefas de controlo, de interface com operadores humanos e de

interface com o sistema controlado. Exemplos típicos de IEDs incluem URTs, estações de trabalho, protecções, transdutores ou PLCs¹.

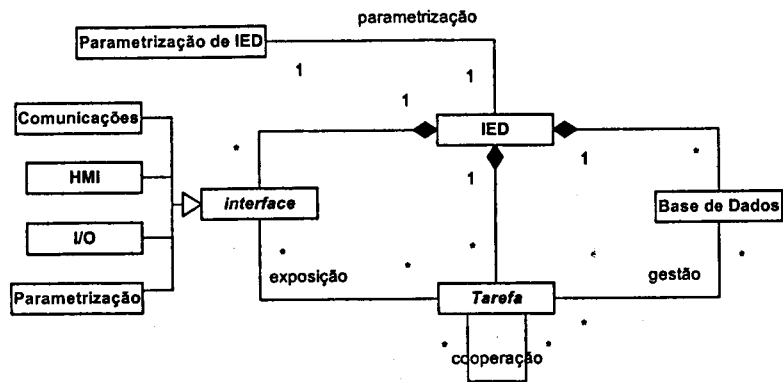


Figura 3. Modelo genérico de um IED.

Independentemente das funções desempenhadas e da sua arquitectura de *hardware* ou *software*, um IED compreende normalmente interface, tarefas, uma base de dados e a sua parametrização. A interface de um IED compreende um conjunto de pontos de interacção, seja com o operador (*HMI*), com o sistema controlado (*I/O*) ou com outros IEDs (*I/O* ou *Comunicações*). Um tipo de interface particular consiste na interface de parametrização do dispositivo, normalmente suportada por um sistema de comunicações entre as ferramentas de parametrização e o IED. Este sistema de comunicações pode ser baseado na transferência de ficheiros ou em protocolos vocacionados para a parametrização. As funções do dispositivo são desempenhadas por um conjunto de tarefas definidas em algoritmos fixos ou programáveis. As tarefas executam funções não só de controlo, mas também de suporte ao dispositivo, como, por exemplo, a implementação pilhas de comunicações ou sistemas de auto-diagnóstico. A imagem que o dispositivo dispõe dos vários processos envolvidos no sistema de automação é concretizada numa base de dados de tempo real (figura 4). As bases de dados de tempo real, na sua maioria, apresentam uma estrutura simples baseada em vectores de pontos de dados com tipos simples. Estas suportam a execução de tarefas, reflectindo a informação recebida através das interfaces ou resultante da execução de funções.

¹ No anexo I podem ser encontradas listas de dispositivos característicos dos sistemas de automação de instalações do sistema eléctrico baseados em [10], [11] e [12].

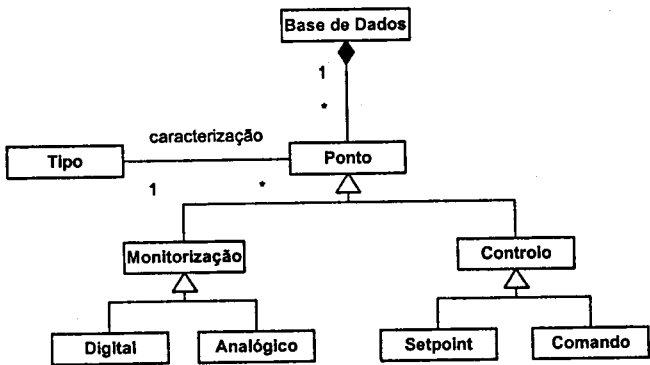


Figura 4. Modelo de base de dados de um IED.

2.2.2. Caracterização Funcional

De forma a facilitar a compreensão funcional de um sistema de automação é relevante introduzir um modelo de níveis funcionais e interfaces lógicas. Apresenta-se na figura 5 um modelo que se julga ser aplicável à generalidade das instalações do sistema eléctrico, baseado no modelo apresentado em [9].

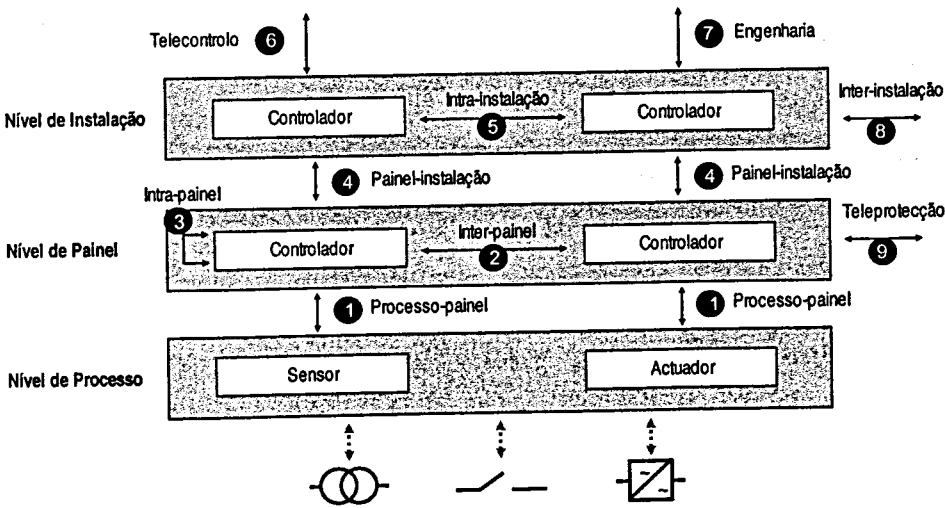


Figura 5. Modelo fundamental de níveis funcionais e de interfaces lógicas.

A figura apresenta uma distribuição e separação lógica de funções em três níveis. O primeiro, denominado nível de processo, compreende as funções que realizam a interface à instalação, por exemplo localizadas em dispositivos de I/O remoto ou sensores e actuadores inteligentes. O nível intermédio, nível de painel, compreende funções limitadas a um subconjunto da instalação e vulgarmente associadas a um painel (de grupo, de linha, de barramento, de transformador ou outros), como por exemplo a protecção diferencial de um transformador. O nível de instalação inclui funções que abrangem toda a instalação, subdividindo-se em duas classes: de processo, tal como a função encravamento distribuído, e de

interface, incluindo o HMI. As interfaces lógicas, sobre as quais se estabelecem ligações cooperativas entre funções, são descritas na tabela 3. Uma identificação e categorização de funções de um sistema de automação, com base em [9] e [10] apresentada na tabela 4.

Tabela 3. Interfaces lógicas presentes num sistema de automação.

Interface		Descrição
1	Processo-painel	Troca rápida de dados de monitorização e controlo, entre o nível de processo e o nível de painel, compreendendo dados de medida dos sensores e dados de controlo para actuadores. A informação de medida é essencialmente instantânea e amostrada, por exemplo proveniente de transformadores de medida.
2	Inter-painel	Troca de dados entre funções de painel localizadas em painéis distintos. Caracterizadas por requisitos apertados no atraso de comunicação e por suportar funções distribuídas rápidas como o encravamento.
3	Intra-painel	Troca de dados entre funções localizadas num mesmo painel.
4	Painel-instalação	Troca de dados entre o nível de painel e o nível de instalação. Compreende múltiplos tipos de informação, nomeadamente de controlo e processo, mas também de gestão.
5	Intra-instalação	Troca de dados entre funções do nível de instalação.
6	Telecontrolo	Troca de dados com sistema remotos de nível superior na hierarquia de controlo do sistema eléctrico, nomeadamente centros de condução ou operação.
7	Engenharia	Troca de dados com postos de engenharia, remotos ou locais, como por exemplo para (tele) diagnóstico, (tele) parametrização ou (tele) reconfiguração.
8	Inter-instalação	Troca de dados entre instalações do sistema eléctrico, essencialmente para suporte a funções distribuídas de controlo da rede eléctrica.
9	Teleprotecção	Troca de dados entre funções de nível de painel localizadas em instalações distintas, por exemplo para suporte a teleprotecção de linhas.

Este modelo ilustra elementos importantes. Por um lado indica uma diversidade significativa que abrange os requisitos tecnológicos das várias funções presentes no sistema de automação e dos sistemas de comunicações que as suportam. Este facto é concretizado nos variados tipos de IEDs e protocolos de comunicações utilizados. Por outro lado, ilustra que a atribuição de funções, quer a níveis funcionais quer a dispositivos, não é necessariamente fixa. Com efeito, num dado sistema concreto estão disponíveis, em regra, várias opções de integração. Estes factores de diversidade, complexidade e variabilidade têm implicado a adopção de ferramentas de engenharia múltiplas e flexíveis. Neste contexto, a adequação de ferramentas de engenharia a um domínio de aplicação lato é factor económico preferencial, pelo que se torna interessante proceder a um trabalho de generalização.

Tabela 4. Lista de funções presentes num sistema de automação de instalações do sistema eléctrico.

Categoria	Funções
Suporte ao sistema de automação	Comunicações, gestão de sistemas de comunicações, sincronização de relógios e auto-verificação de dispositivos.
Configuração e manutenção	Identificação de IEDs, gestão de <i>software</i> , gestão de configurações, gestão do modo de operação, parametrização e gestão de segurança.
Operação e controlo	Interface ao utilizador, interface à instalação, controlo de acesso, actuação, inibição de comandos, disseminação de eventos, comutação sincronizada, activação de parametrizações, gestão de alarmes, gestão de eventos, processamento estatístico, arquivo, recuperação de dados e recuperação de perturbografia.
Automatismos locais de processo	Controlo local, protecção local e encravamento de painel.
Automatismos distribuídos de suporte	Encravamento de instalação e verificação de sincronismo.
Automatismos distribuídos de processo	Falha de disjuntor, adaptação automática de protecções, bloqueio inverso de protecções, deslastre para estabilização de frequência, reposição de carga, regulação de tensão e potência reactiva, comutação de linhas e transformadores, sequências automáticas de comutação, arranque/paragem de grupo, teleprotecção, WAP, etc.

2.2.3. Elementos de Parametrização

Os IEDs apresentam invariavelmente comportamento sujeito a parametrização. A parametrização é composta por listas de parâmetros que podem ser considerados, de forma abstracta e simplificada, tuplos nome/tipo/valor utilizados na adaptação de um conjunto de dispositivos genéricos a um sistema de automação concreto. Os parâmetros de um dado sistema são vulgarmente organizados numa estrutura de classificação hierárquica com múltiplos níveis, frequentemente reflectindo a estrutura do sistema controlado. Esta estrutura compreende, por exemplo no caso de uma subestação, uma hierarquização por nível lógico do sistema: instalação, nível de tensão, painel, equipamento primário. Os parâmetros podem ser caracterizados segundo um modelo de categorias e tipos adaptado de [13], que se apresenta na figura 6. Qualquer parâmetro, quanto à sua origem e função, integra uma de três classes: sistema, processo ou funcional.

Os parâmetros de sistema determinam a cooperação entre IEDs, as suas estruturas internas e as relações de dados entre as várias funções. Compreendem, por exemplo, a definição da composição das bases de dados de tempo-real de dispositivos, a identificação e endereçamento dos pontos de dados ou a parametrização de protocolos de comunicações. Duas classes particulares de parâmetros de sistema, dada a natureza distinta dos seus modelos face às restantes, compreendem (i) a definição de interfaces sinópticos, e (ii) a programação de funções de controlo automático.

Os parâmetros de processo compreendem a descrição da interface entre o sistema controlado e o sistema de controlo, nomeadamente os pontos de interface e seus tipos de dados, conversão de valores, temporização de saídas, etc.

A última classe de parâmetros, parâmetros funcionais, compreende atributos quantitativos ou qualitativos de funções, como por exemplo os valores alvo de controlo ou as condições de disparo de protecções ou de activação de sequências automáticas. Estes parâmetros, normalmente alteráveis *online*, estabelecem perfis ou gamas de operação do sistema. Os parâmetros funcionais podem ser activáveis, isto é, pertencer a grupos de parâmetros que podem ser activados em simultâneo.

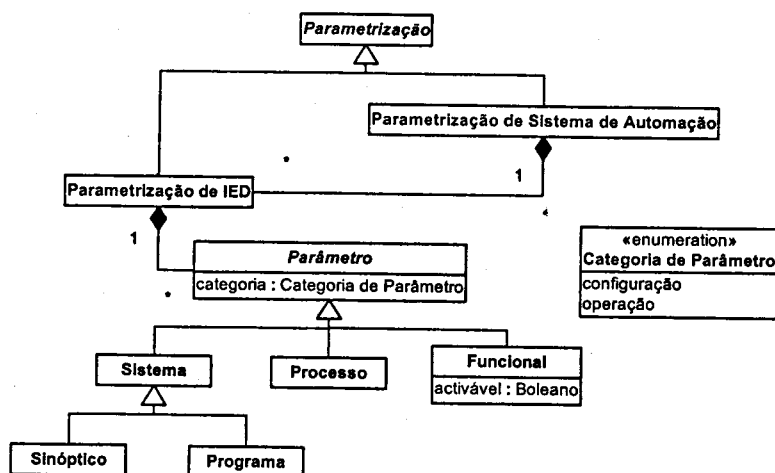


Figura 6. Modelo para classificação de parametrizações.

Os parâmetros, quanto à sua manipulação, podem ser agrupados em duas categorias: configuração e operação. Os parâmetros de configuração definem o comportamento global do dispositivo ou sistema e são atribuídos geralmente durante a parametrização inicial, resultando de decisões importantes de desenho. Estes parâmetros são vulgarmente modificados *offline* e a sua atribuição implica normalmente restrições temporárias no funcionamento do sistema. Os parâmetros de configuração mais comuns são dos tipos sistema e processo. Os parâmetros de operação definem vulgarmente o comportamento parcial de uma dada função do sistema e são normalmente alteráveis durante a operação deste, sem impacto no seu funcionamento, através do HMI ou, remotamente, através de um sistema de comunicações. Os parâmetros de operação mais comuns são dos tipos processo e funcional.

2.2.4. Ciclo de Vida

O ciclo de vida de um sistema de automação acompanha o ciclo de vida da instalação controlada e pode estender-se por algumas décadas (figura 7). Durante o seu período de vida o sistema de automação sofre modificações motivadas essencialmente por alterações na instalação controlada (expansão ou remodelação) ou por remodelação tecnológica dos seus elementos constituintes. Quer o desenvolvimento inicial do sistema quer a incorporação de modificações são implementados através de projectos de desenvolvimento levados a cabo por engenheiros de automação.

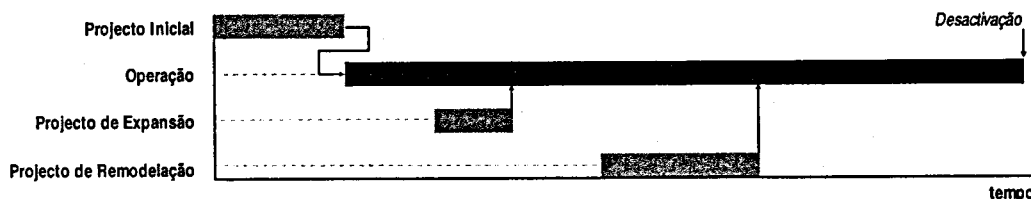


Figura 7. Ciclo de vida de um sistema de automação.

Um projecto de desenvolvimento compreende um conjunto de fases atravessadas pelo fluxo de trabalho, de forma essencialmente sequencial. A figura 8 representa este modelo em cascata.

A fase de análise compreende o estudo e definição dos objectivos funcionais, tecnológicos e económicos, da qual resultam documentos de requisitos na forma de cadernos de encargos ou outros que podem incorporar também algumas soluções arquitecturais e tecnológicas.

A fase de desenho envolve actividades de definição final da configuração de *hardware* do sistema de automação, bem como de definição e de validação das funções lógicas do sistema. O resultado do desenho do sistema compreende vários modelos, como por exemplo, listas de pontos de interface com o processo, desenhos eléctricos de armários e interligações, estruturas de dados ou algoritmos de controlo.

Concluído o desenho, procede-se à integração do sistema em fábrica, o que envolve actividades de interligação e parametrização de dispositivos segundo os modelos definidos na fase de desenho. Sobre o sistema integrado são então realizados ensaios de sistema em fábrica (FAT) que levam à aceitação provisória do sistema.

Após a aceitação dos FATs procede-se à instalação do sistema, envolvendo essencialmente actividades de integração física e interligação do sistema de automação ao processo, mas também de parametrização adicional. Os ensaios da interface entre o sistema de automação e o processo, bem como os ensaios do sistema integrado, caracterizam os SATs e permitem realizar a validação do sistema final.

Uma vez aceites os SATs, o sistema é colocado em exploração, vulgarmente em duas fases, uma fase inicial de exploração experimental, que se considera ser ainda parte do projecto por se realizarem análises do comportamento do sistema de automação, e a exploração definitiva em paralelo com a exploração da instalação.

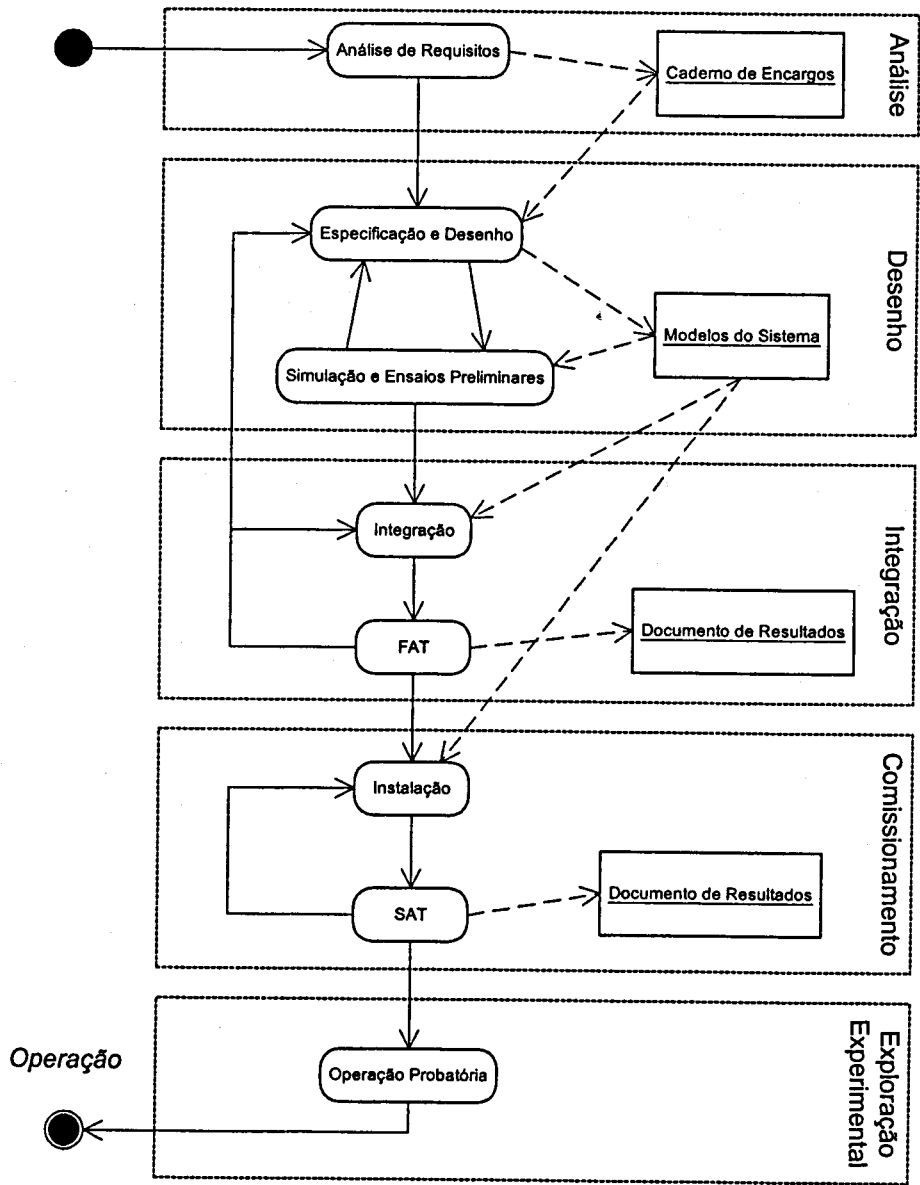


Figura 8. Fases de um projecto de desenvolvimento de um sistema de automação.

2.2.4.1. Actividades de Engenharia

Uma vez caracterizado o ciclo de vida de um sistema de automação e os projectos associados ao seu desenvolvimento, importa enumerar o conjunto de actividades do engenheiro de automação cuja realização pode ser auxiliada por ferramentas de engenharia do domínio de aplicação deste trabalho. Estas compreendem:

- a definição, especificação, implementação e validação das funções lógicas do sistema;
- a definição da configuração de *hardware* do sistema de automação (arquitectura física do sistema, tipo e número de IEDs a integrar) e a atribuição de funções lógicas a dispositivos;
- a integração de dispositivos (interligação e parametrização);
- o ensaio do sistema;
- a documentação de todas as definições (modelos, listas de parâmetros, ligações terminais, entre outros);
- a gestão de versões.

2.3. Modelo do Sistema de Automação

Um sistema de automação pode assim ser visto sobre uma de quatro perspectivas (figura 9): (i) a perspectiva de projecto, compreendendo as actividades associadas ao seu desenvolvimento e a sua evolução ao longo do tempo, (ii) a perspectiva física, compreendendo os dispositivos físicos e sistemas de comunicações associados, (iii) a perspectiva de parametrização e (iv) a perspectiva funcional lógica, compreendendo as funções por ele desempenhadas e as relações entre estas. Estas vistas não são ortogonais e estão fortemente interrelacionadas. O sistema funcional lógico é mapeado no sistema físico através da sua parametrização no decurso do projecto onde são modelados.

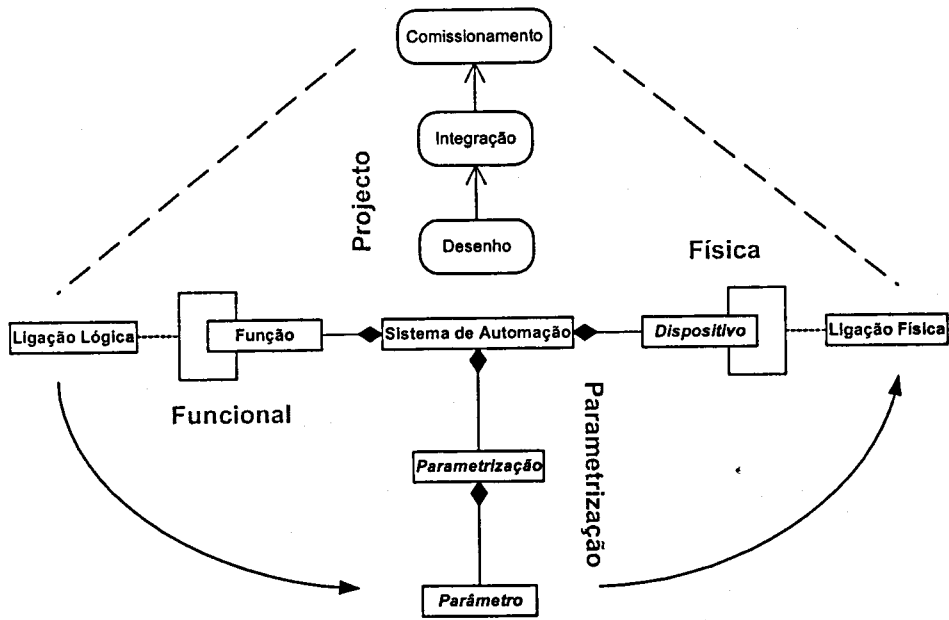


Figura 9. Vistas do modelo do sistema de automação.

Este modelo apresenta características importantes na perspectiva das ferramentas de engenharia. O sistema físico compreende o sistema primário, o sistema secundário e a interligação entre estes. O sistema primário e as suas características tecnológicas são o elemento condicionante do sistema de automação, pelo que a sua caracterização é chave no desenvolvimento do projecto, sendo particularmente relevante nas fases de análise e desenho. O sistema primário é relativamente estável ao longo do ciclo de vida do sistema de automação. Os sistemas de comunicações, os IEDs e a interface destes com o sistema primário constituem o sistema secundário. O sistema secundário é o alvo da parametrização e apresenta alguma mutabilidade ao longo do ciclo de vida do sistema de automação. Numa perspectiva de longo prazo, o ciclo de vida de produto dos elementos constituintes do sistema secundário é, em regra, significativamente mais curto do que os princípios funcionais da automação de sistemas de energia, tanto na vertente das estratégias de controlo como na tecnologia do sistema primário. Os modelos funcionais lógicos, implementados no sistema secundário (figura 10), são particularmente relevantes no desenvolvimento do sistema de automação uma vez que apresentam uma relação muito estável com o sistema primário.

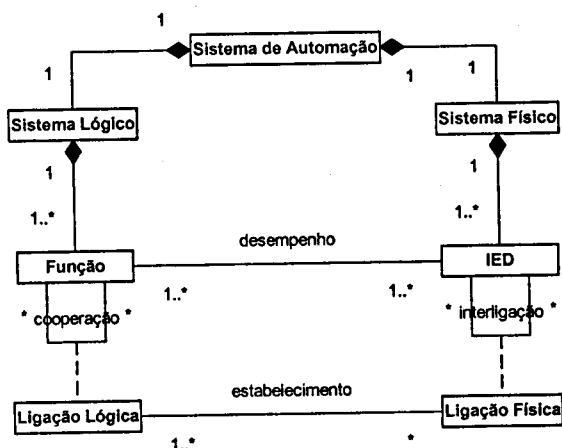


Figura 10. Modelo lógico do sistema de automação e sua relação com o sistema secundário.

Uma abordagem ao desenho numa perspectiva lógica apresenta assim vantagens interessantes: (i) permite um estudo centrado nas funções de automação, o que possibilita a obtenção de modelos mais adequados, (ii) suporta consistentemente a reutilização do trabalho de engenharia e (iii) permite que o ciclo de vida da ferramenta de engenharia, bem como dos modelos por esta criados, suplantem o ciclo de vida dos IEDs.

Uma análise das funções do sistema de automação (anexo II) permite concluir que estas se podem caracterizar segundo três componentes distintas: estrutural estática, dinâmica e de interface. A componente estrutural estática compreende a definição de dados e relações entre dados, englobando a definição das interfaces da função, bem como das estruturas internas de dados de tempo real. Esta componente é caracterizada pela existência de estruturas de dimensão reduzida, baseadas em tipos de dados simples e relações complexas, nomeadamente entre funções.

À componente dinâmica, ou comportamento, corresponde a evolução dos dados ao longo do tempo, caracterizando-se pela evolução conduzida por eventos discretos, com presença de concorrência (sequenciação, paralelismo, gestão de recursos e sincronização), requisitos de tempo-real e características de assincronicidade elevada. Embora a monitorização periódica de grandezas contínuas seja relevante para os algoritmos de controlo, estes evoluem essencialmente de forma discreta e esporádica, característica comum às acções sobre o sistema controlado. Quanto à implementação dos algoritmos associados à componente comportamental de uma função, podem ser implementados de forma fixa, como é o caso das funções de protecção típicas, ou programável pelo engenheiro de automação.

A componente de interface, que engloba a interacção com o utilizador, caracteriza-se por ser rica e incluir, entre outros, representações sinópticas dinâmicas, listas de eventos, alarmes e gráficos de tendência. Os sinópticos são, na perspectiva das ferramentas de engenharia, os elementos de interface mais relevantes e compreendem quer a visualização do sistema quer a acção do operador sobre este. Os sinópticos são, no seu essencial, representações dinâmicas interactivas, bidimensionais e vulgarmente vectoriais, verificando-se, nas aplicações mais recentes, a integração de outros elementos como áudio e vídeo, constituindo aplicações hiper-média em tempo real.

Este modelo de informação simples, representado na figura 11, é considerado fundamental, dado que cada uma das funções do sistema de automação pode ser caracterizada com recurso à integração de cada uma das suas três componentes e que o desenho das três componentes representa, a par das ligações lógicas, a parte essencial da actividade do engenheiro de automação no desenho de uma dada função. O modelo completo do sistema de automação, introduzido neste capítulo e adoptado como fundamental no desenvolvimento deste trabalho, é apresentado na totalidade e com detalhe adicional no anexo III.

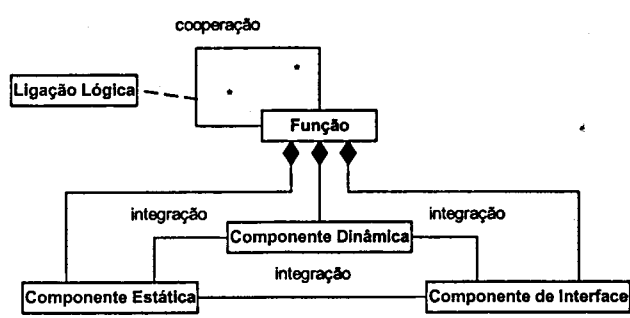


Figura 11. Modelo fundamental de uma função no sistema de automação.

2.4. Sobre as Ferramentas de Engenharia

2.4.1. Ferramentas de Engenharia Comuns

As ferramentas de engenharia com utilização frequente hoje neste domínio de aplicação são variadas e classificáveis em três classes: ferramentas de modelação, ferramentas de programação e ferramentas de parametrização. Esta classificação é descrita na tabela 5. Ferramentas que integrem estas três vertentes não são comuns. Estas vulgarmente compreendem a associação de ferramentas distintas de cada categoria individual em pacotes de *software* para programação e parametrização de IEDs de uma dada família de produtos de um mesmo fabricante.

O aparecimento de ferramentas que integram a modelação, a programação e a parametrização de sistemas de automação é, no entanto, uma tendência natural. O desenvolvimento de modelos como os propostos na norma IEC 61850 e nos trabalhos internacionais associados vem criar uma arquitectura de referência para sistemas neste domínio. Este facto, associado à introdução de funcionalidades de transferência de meta-informação, conduzirá, por outro lado, à diminuição da disparidade de mecanismos e interfaces associados à parametrização de dispositivos. Estes factores possibilitam a construção de ferramentas de engenharia abrangentes, necessárias numa perspectiva económica da actividade de engenharia de automação inserida no novo modelo de mercado do sistema eléctrico.

Tabela 5. Classes de ferramentas.

Classe	Descrição
Modelação	As ferramentas de modelação são tendencialmente genéricas e não específicas do domínio de engenharia de automação. Baseiam-se em linguagens de alto nível como a UML ou as redes de Petri e permitem descrever, modelar ou validar sistemas na sua vertente estática e dinâmica. CPN Tools ^[14] , Artifex ^[15] e Poses++ ^[16] são exemplos de ferramentas com génese em redes de Petri. Alguns exemplos de ferramentas UML são os ambientes ArgoUML ^[17] , Poseidon ^[18] e Visual Paradigm ^[19] . Ferramentas diagramáticas genéricas, como o Visio ou aplicações de CAD, são também utilizadas no desenho e especificação de sistemas de automação.
Programação	As ferramentas de programação estão fortemente relacionadas com as plataformas de implementação e essencialmente orientadas para a componente dinâmica. Na sua maioria são baseadas nas linguagens IEC 61131-3 ou similares. Grande parte dos fabricantes de IEDs disponibiliza ferramentas deste tipo, das quais o ambiente ISAGraph ^[20] é um exemplo característico.
Parametrização	As ferramentas de parametrização apresentam interfaces de manipulação de modelos individuais de IEDs. Estão fortemente dependentes dos dispositivos e incluem essencialmente formulários de parametrização de elementos estáticos, embora possam também incluir editores gráficos para o HMI ou editores de linguagens para programação de comportamento dinâmico.

2.4.2. Requisitos Genéricos para Ferramentas de Engenharia de Automação

As ferramentas de engenharia para sistemas de automação devem apresentar uma orientação sistémica, ou seja, abranger a modelação e implementação do sistema, ao invés da orientação ao dispositivo de automação. Dado o âmbito abrangente, pretende-se que as ferramentas suportem o desenvolvimento de sistemas complexos, que promovam a agilidade face a alterações do projecto e que apoiem a actividade criativa do engenheiro.

Este propósito suporta a opção por ferramentas essencialmente vocacionadas para o desenvolvimento de modelos. A orientação a modelos serve vários propósitos: (i) conduz a representações inteligíveis pelos vários actores intervenientes no projecto, (ii) permite gerir sistemas complexos, (iii) possibilita o estudo de soluções alternativas antes da tomada de decisões de implementação, (iv) promove a reutilização de trabalho de engenharia durante e entre projectos, e (v) possibilita o desenvolvimento de sistemas de automação independentemente da plataforma de implementação. O desenvolvimento conduzido por modelos, que tem vindo a tornar-se prática corrente na engenharia de *software*, é concretizado na iniciativa MDA ^[21] da OMG. No âmbito da engenharia de automação verifica-se também esta orientação, que se concretiza, entre outros no aparecimento de modelos associados a normas como a IEC 61850 ou a IEC 61499 ^[22].

Para abordar sistemas de automação são normalmente utilizados métodos de projecto dos tipos *bottom-up* (composição de subsistemas parciais), *top-down* (refinamento progressivo) ou híbridos que deverão ser igualmente suportados pelas ferramentas de modelação. Para tal devem estar disponíveis mecanismos que permitam gerir modularidade: abstracção, divisão, agregação e estruturação de modelos interdependentes.

Em sistemas de *software* intensivo a frequência de ocorrência de falhas tende a baixar com a utilização. Este factor, associado ao facto que os sistemas de automação de instalações do mesmo tipo apresentam grandes semelhanças, torna a reutilização de modelos uma das características chave na redução de custos de projecto e no aumento da qualidade do sistema final. A solução deve assim ser fortemente orientada para a reutilização.

O desenvolvimento de sistemas de automação é frequentemente realizado por equipas de engenheiros e por organizações de integração de sistemas. A optimização dos mecanismos de comunicação e a manutenção de um *know-how* organizacional são, nestas organizações, factores determinantes de sucesso. Por esta razão a representatividade e facilitação da comunicação entre os vários intervenientes no processo de desenvolvimento deve ser endereçada pela ferramenta, pelo que a manipulação de modelos deverá ser realizada através do manuseamento de artefactos gráficos, preferencialmente baseados em linguagens conhecidas.

A disponibilidade de funcionalidades de validação precoce do sistema, nomeadamente da componente comportamental e da consistência da parametrização estática, é também requisito interessante na medida em que o domínio de aplicação apresenta características de dimensão e dinamismo elevados e que a implementação final envolve custos significativos em equipamentos e instalações normalmente não disponíveis na fase de desenho. A análise de modelos permite avaliar opções de desenho numa fase inicial do projecto, conduzindo a uma validação precoce de requisitos e opções de desenho e implementação. A validação pode ser auxiliada pela existência de representações alternativas, por análise estática, por análise formal e por simulação.

Os objectivos genericamente aplicáveis ao desenvolvimento de ferramentas de engenharia devem ser necessariamente observados, nomeadamente: a promoção da qualidade da implementação final, a contenção na quantidade de recursos necessários para o seu desenvolvimento, o impacto na produtividade do utilizador final, em suma, a rentabilidade económica do seu desenvolvimento.

2.4.3. Requisitos Funcionais

Neste ponto são identificados os requisitos funcionais comuns para ferramentas de engenharia. Os requisitos funcionais incluem quatro grandes áreas, ilustradas no diagrama de casos de uso da figura 12: a construção de modelos, a validação de modelos, a parametrização e a geração de documentação. Alguns destes requisitos não serão, no âmbito deste trabalho, analisados na sua totalidade ou com todo o detalhe necessário à sua implementação. A sua identificação é no entanto necessária, uma vez que as opções arquitecturais a tomar no âmbito do desenvolvimento de uma ferramenta são condicionadas pela totalidade dos requisitos aplicáveis.

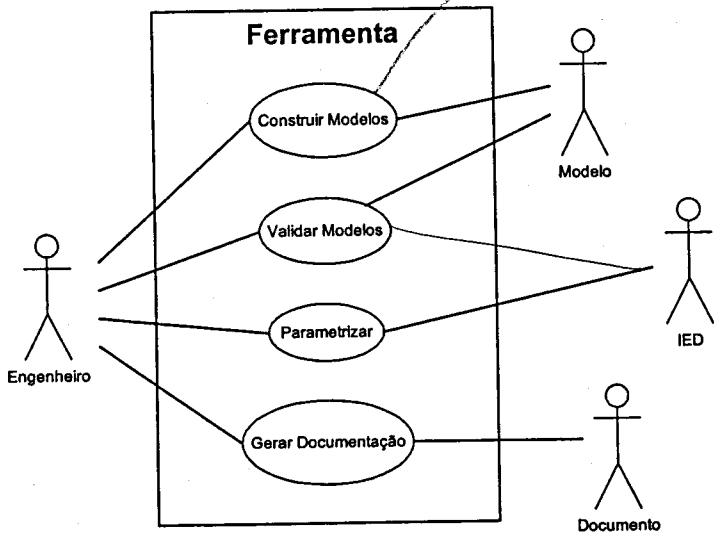


Figura 12. Requisitos funcionais.

2.4.3.1. Modelação

A construção de modelos (figura 13) deve ser realizada com recurso a uma linguagem gráfica única e abrangente. A linguagem deve ser capaz de gerar modelos estruturais estáticos, de comportamento e de interface, de acordo com o modelo para funções de automação. Pretende-se que esta linguagem seja adequada para a modelação de sistemas de automação em geral, mas adequada também para modelação dos vários tipos de dispositivos e sistemas comunicativos. A interface para criação de modelos deverá permitir o desenho de múltiplas vistas gráficas para um mesmo modelo e suportar o desenho iterativo, ou seja, a manipulação de modelos incompletos ou semanticamente incorrectos.

Uma linguagem com estes requisitos será capaz de gerar modelos suficientemente genéricos, mas a sua aplicação principal será a construção de modelos (i) físicos (de sistemas, dispositivos, interligações comunicativas ou esquemas de parametrização), de sistemas primários e de sistemas secundários, e (ii) lógicos, de funções e ligações lógicas. Deverá ser também possível modelar a atribuição de funções lógicas a dispositivos físicos, bem como de ligações lógicas a sistemas de comunicações ou outro tipo de ligações físicas.

A ferramenta deverá ainda apresentar funcionalidades de gestão de modelos como sejam a identificação e, eventualmente o rastreio, de versões, a partilha de modelos e a gestão de bibliotecas de modelos interdependentes. A integração de funcionalidades de importação e exportação de modelos, ou de partes de modelos, deve ser prevista.

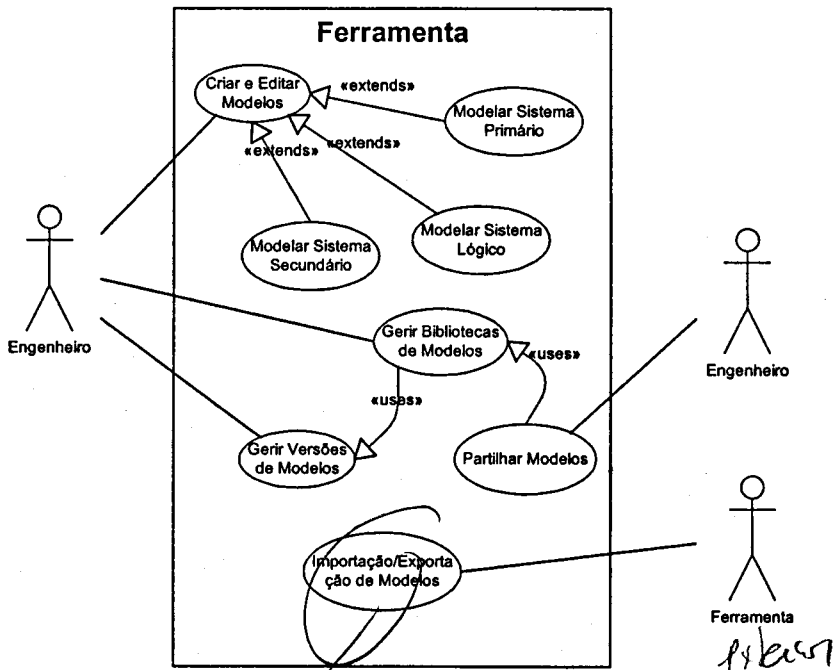


Figura 13. Requisitos para construção de modelos.

2.4.3.2. Validação

importar / exportar

A validação de modelos (figura 14) é um requisito genérico que pode ser decomposto em funcionalidades distintas, dependendo da vertente do modelo a validar. Em geral pretende-se validar modelos de forma precoce, ou seja, antes da disponibilização do sistema final.

A validação estática do modelo deverá ser aplicada quer na análise de erros semânticos na utilização da linguagem de modelação quer na análise da componente estrutural estática de um dado modelo. Esta validação deverá ser preferencialmente realizada *online*, ou seja, em paralelo com a edição do modelo. A validação estática pretende garantir a compatibilidade entre sub-modelos distintos, detectando erros em interfaces comuns. Nesta vertente da modelação pretende-se que as linguagens adoptadas possuam, de raiz, características que permitam prevenir a ocorrência deste tipo de erros.

A validação da componente comportamental pode ser auxiliada por funcionalidades de simulação ou de análise formal. A validação comportamental de modelos por análise formal implica que a sua definição seja realizada recorrendo a linguagens também formais, vulgarmente apresentando reduzida capacidade de modelação. É reconhecido que o poder de modelação de uma linguagem diminui com o aumento da precisão semântica e capacidade analítica. Com efeito, a análise formal de determinadas propriedades só é por vezes possível através da introdução de restrições sintácticas e semânticas que apresentam frequentemente um custo demasiado elevado na capacidade de modelação sistemas de alguma

complexidade. Uma alternativa ao uso dos métodos analíticos de uma linguagem formal é a simulação de eventos discretos. A simulação é um método de análise pesado, no entanto é simples, aplicável a modelos complexos e não introduz limitações significativas ao poder de modelação das linguagens. A simulação não apresenta garantia de determinação correcta de propriedades comportamentais, no entanto, é um método conveniente e aplicável em muitos sistemas práticos. A simulação de eventos discretos apresenta ainda outra vantagem: é um método de validação que pode ser aplicado a qualquer linguagem de modelação comportamental com alguma precisão semântica. Assim estabelece-se como requisito preferencial a validação com recurso a simulação. Os mecanismos de simulação devem compreender a simulação automática e a simulação interactiva, com pontos de paragem e passo-a-passo.

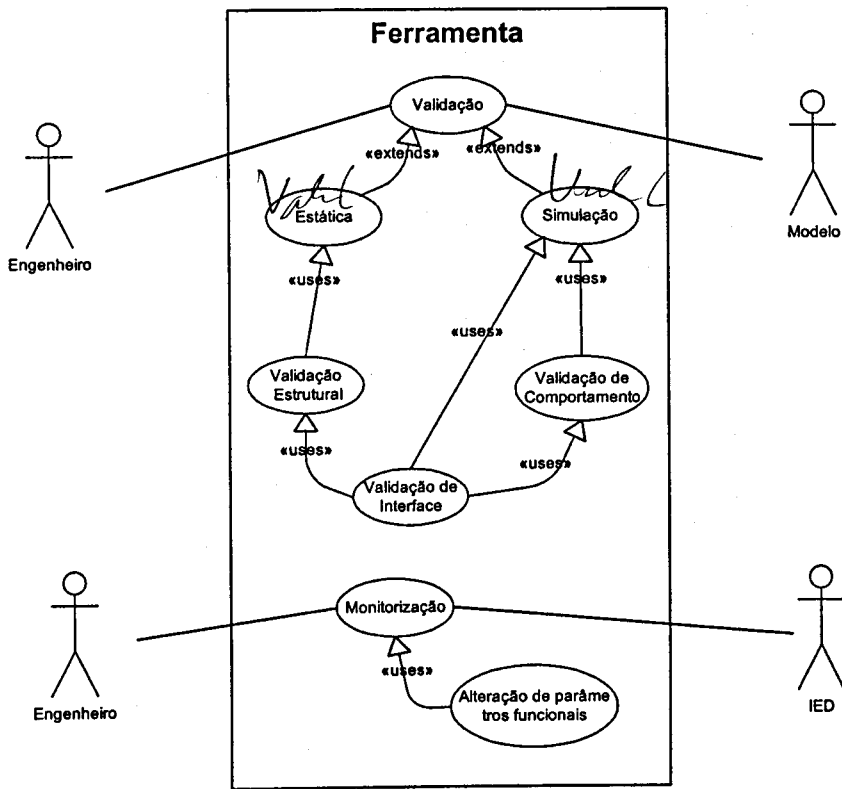


Figura 14. Requisitos para validação de modelos.

A validação precoce da componente de interface deverá ser levada a cabo também por simulação. A simulação da componente de interface deve ser associada à simulação comportamental.

O ensaio do sistema (ensaios preliminares, SATs ou FATs) pode também ser auxiliado pela ferramenta de engenharia. Este requisito compreende a monitorização da operação do equipamento em ensaio, incluindo a alteração de parâmetros funcionais e a alteração simulada de valores do processo.

2.4.3.3. Parametrização

A parametrização (figura 15) compreende a transformação de modelos criados com a ferramenta em modelos dependentes dos dispositivos. Diferentes famílias de dispositivos apresentam diferentes mecanismos de parametrização suportados por formatos de dados e mecanismos de acesso distintos. Para aplicação a uma dada família de IEDs é vulgar a utilização de formatos de dados comuns, um mesmo mecanismo de acesso e uma máquina virtual comum.

Os parâmetros distinguem-se, a este nível, pela sua categoria: operação ou configuração. Os parâmetros de configuração, que compreendem a parte mais significativa do modelo, são o objecto deste requisito genérico. Os parâmetros de operação serão alterados através da monitorização de operação do IED, caso a funcionalidade esteja disponível.

A parametrização envolve, em qualquer caso, a transferência de conjuntos de parâmetros para os IEDs e a verificação da aceitação destes das acções de alteração de parâmetros. Envolve ainda a gestão de versões de parametrização, incluindo pelo menos a identificação de versões. Poderá ainda ser disponibilizada, dependendo do dispositivo, a possibilidade de recuperação de modelos a partir da parametrização residente no dispositivo.

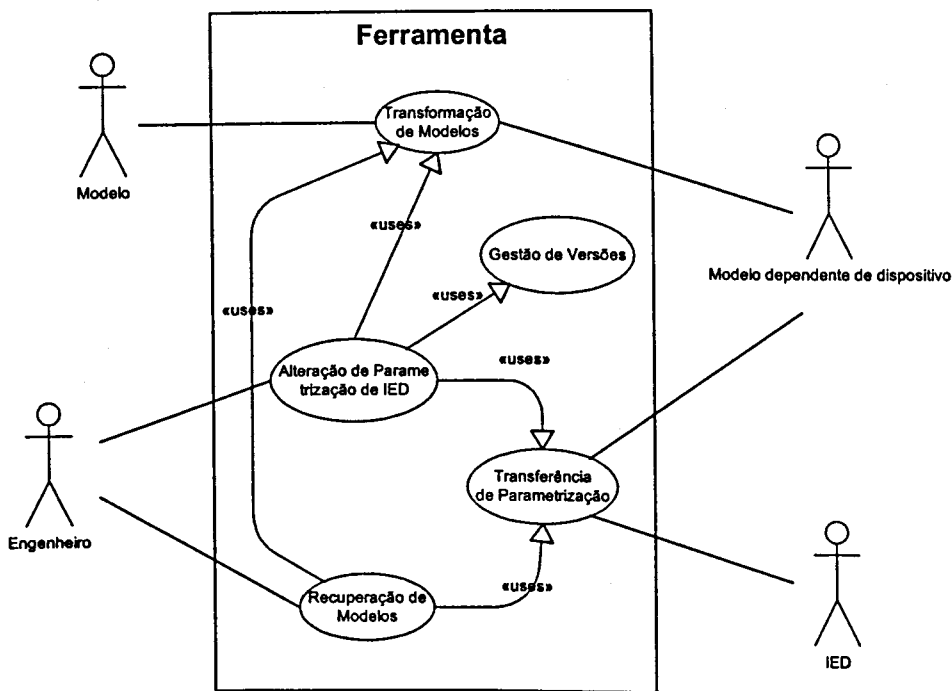


Figura 15. Requisitos para parametrização de IEDs.

2.4.3.4. Documentação

A construção de múltiplos tipos de documentos associados às várias fases do projecto de desenvolvimento do sistema de automação é sempre necessária para a discussão, aprovação, bem como para a documentação e a referência futura do sistema de automação. Dado o carácter gráfico e abrangente da ferramenta, a geração de elementos documentais (figura 16) merece estar a esta associada como requisito funcional de primeiro nível.

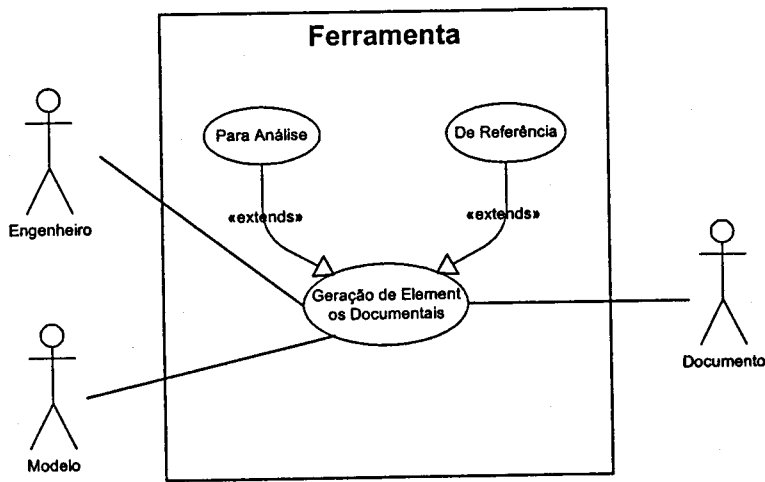


Figura 16. Requisitos para geração de documentação.

Deve ser possível incluir os elementos documentais gerados em documentos estruturados, construídos por ferramentas externas. Estes elementos documentais incluem, entre outros, diagramas unifilares, diagramas de configuração e arquitectura do sistema secundário, diagramas de interligações comunicativas, algoritmos de controlo, quadros sinópticos, esquemas de protecção e listas de pontos de interface.

2.4.4. Conclusões

Definidos o âmbito e domínio de aplicação, bem como os seus requisitos principais, conclui-se que uma solução de ferramentas de engenharia deve apresentar uma interface para definição de modelos de sistemas de automação e gerar, por métodos essencialmente automáticos, os produtos da actividade de engenharia. Destes destacam-se as estruturas de parametrização de dispositivos e esquemas de configuração de sistemas físicos. Dada a integração das componentes lógicas de um sistema de automação, a ferramenta deverá proporcionar um ambiente de desenvolvimento integrado e coeso, capaz de manipular igualmente as vertentes estática, dinâmica e de interface. A definição de uma única linguagem de modelação capaz de lidar com estes modelos, correspondendo ao elemento comum entre esta e o engenheiro de automação, é assim uma orientação interessante para o desenvolvimento de uma ferramenta e aquela que se adopta neste trabalho.

use mais auto

Parte II

Linguagem de Modelação

Introdução

Linguagens utilizadas em sistemas de automação
Justificação e introdução ao modelo da linguagem

Sintaxe Abstracta e Semântica

Modelo reduzido e modelo completo da sintaxe abstracta e da semântica

Notação

Sintaxe concreta e léxico

Semântica de Domínio para Sistemas de Automação

Modelo e semântica de domínio
Relação com o modelo de sistemas de automação
Sobre a aplicação da linguagem

Conclusões

Rastreio de requisitos e avaliação da ASML
Trabalhos futuros e conclusões da segunda parte

Introdução

Os requisitos essenciais para a ASML decorrem dos requisitos apresentados para ferramentas de engenharia, na parte I da dissertação. Estes apontam para uma linguagem gráfica capaz de representar convenientemente as várias componentes do modelo de sistemas de automação. Importa acrescentar também que a linguagem deve ser dotada de características tais que os modelos segundo ela definidos sejam independentes da plataforma de implementação final, mas suficientemente concretos e simples para que a sua manipulação, nomeadamente a sua simulação e a sua transformação para modelos dependentes dos dispositivos, seja passível de realização por métodos automáticos. Foi preocupação do autor posicionar a linguagem para que esteja, por um lado, enquadrada nos procedimentos, métodos e linguagens adoptados na prática actual e, por outro, enquadrada nas tendências actuais de desenvolvimento de sistemas de informação.

As linguagens mais comumente utilizadas no projecto de sistemas de automação são essencialmente linguagens de programação, orientadas para a implementação. Este facto advém do menor peso em tempo de engenharia dispendido em actividades de modelação nos projectos convencionais, bem como da preponderância convencional da componente comportamental. Neste grupo incluem-se linguagens monolíticas, estruturadas e não estruturadas, classificadas como imperativas ou declarativas do tipo lógico, no que se refere à construção de algoritmos. Algumas destas linguagens apresentam notação gráfica, como por exemplo os diagramas de escada e os diagramas de fluxos, ou notação textual, constituídas por listas de instruções próximas do *assembly* ou então com níveis superiores de estruturação. Incluem-se também neste conjunto algumas linguagens declarativas com notação gráfica, baseadas em máquinas de estado finitas e generalizadas na *grafcet* ^[23], para definição de comportamento conduzido por eventos. Estas linguagens estão hoje representadas e completamente integradas na parte três da série de normas internacional IEC 61131 sobre a

Linguagens de Programação para Sistemas de Automação

programação de PLCs [24]. A base de utilização desta norma é muito alargada, o que se reflecte no suporte sólido da maioria dos fabricantes de PLCs.

As linguagens hoje utilizadas são orientadas para as arquitecturas de dispositivos e essencialmente vocacionadas para a componente dinâmica do sistema de automação. Apresentam tipos de dados simples e capacidade de modelação estática reduzida. A sua utilização tem vindo a evoluir no sentido do aumento da importância da modelação, o que se pode verificar no uso cada vez mais frequente das linguagens baseadas em diagramas de estados e blocos.

1.2. Linguagens Orientadas ao Objecto

A importância das linguagens orientadas ao objecto está comprovada ao nível dos sistemas de informação e a sua aplicação tem vindo a revelar-se também adequada no âmbito da modelação de sistemas de automação. Com efeito, as linguagens orientadas ao objecto apresentam um conjunto de características interessantes face às linguagens do tipo monolítico.

Os sistemas de automação têm vindo a sofrer modificações importantes ao nível da quantidade de informação manipulada e da complexidade de relações entre dados, que têm vindo a crescer à medida que mais funcionalidades são integradas. As linguagens orientadas ao objecto estão particularmente vocacionadas para o suporte à manipulação da componente estrutural estática, permitindo a sua organização através dos conceitos de classe, relação e objecto. A orientação ao objecto promove uma relação forte entre dados e comportamento, o que apresenta vantagens na modelação. O objecto como unidade de informação e comportamento é um conceito intuitivo que aproxima a organização dos modelos da dos sistemas reais, promovendo a coesão de subsistemas fortemente dependentes e relaxando as relações entre subsistemas inerentemente independentes.

Ao nível da reutilização a orientação ao objecto apresenta melhorias face às linguagens convencionais na medida em que se baseia no encapsulamento e em relações de herança. Estas características limitam o impacto das alterações e promovem a reutilização imediata de modelos. Por outro lado, as relações de dependência são mais visíveis, mesmo explícitas, melhorando a identificação do impacto da introdução de alterações. Em suma, a estabilidade dos modelos construídos face a alterações de requisitos é promovida.

O nível de encapsulamento e abstracção presente nas linguagens orientadas ao objecto permite o estabelecimento de interfaces limitadas e as notações normalmente simples e utilizáveis em todo o processo de desenvolvimento do sistema têm um impacto directo na escalabilidade de modelos e sistemas.

Os modelos orientados ao objecto apresentam consistência entre as vistas de análise, desenho e implementação, isto é, a implementação final apresenta concretizações do desenho que por sua vez apresenta concretizações da análise de

requisitos. Com efeito, as classes e objectos identificados na fase de desenho têm uma tradução quase imediata na concretização, o que promove a implementação directa dos modelos.

As linguagens orientadas ao objecto apresentam ainda suporte para concorrência. Os objectos são inerentemente concorrentes, representando comportamentos próprios. Mecanismos como passagem de mensagens ou eventos permitem modelar a comunicação e sincronização entre objectos.

O uso generalizado da modelação orientada ao objecto em vários domínios de aplicação, a necessidade de interligação com vários sistemas de informação, nomeadamente sistemas de informação empresariais, fortemente orientados ao objecto, bem como o aparecimento de arquitecturas de referência orientadas ao objecto são também factores que corroboram a adopção deste tipo de modelação na linguagem base de uma ferramenta de apoio ao projecto de sistemas de automação.

1.2.1. A UML

A linguagem para definição de modelos orientados ao objecto mais generalizada é a UML ^[25] ^[26]. Esta linguagem visual surge da agregação das várias escolas da modelação orientada ao objecto incorporando conceitos de: definição estrutural estática, definição de comportamento dinâmico, gestão de modelos e extensão da linguagem. A definição estrutural, nomeadamente no caso das vistas estáticas, é de particular importância e é nesta, juntamente com os conceitos de orientação ao objecto que a suportam e os mecanismos de extensão, que reside o sucesso do par orientação ao objecto – UML. A vista estática assenta nos conceitos de objecto, classe e relações entre classes, permitindo representar as características mais relevantes das linguagens orientadas ao objecto: classificação, herança, abstracção, realização e associação. A unificação na UML das linguagens de modelação orientadas ao objecto, a generalização do seu uso e a evolução dos modelos de dados em sistemas de automação para modelos orientados ao objecto descritos em UML são factores que justificam a sua adopção, mesmo que parcial, na representação da componente estática no domínio de aplicação em causa.

1.3. Modelação Comportamental

Para modelação de comportamento discreto estão disponíveis e são utilizadas linguagens imperativas estruturadas ou declarativas, nomeadamente, as máquinas de estados finitas e as redes de Petri.

As linguagens imperativas baseiam-se na definição de sequências de instruções, ciclos, invocação de funções ou operações, envio de mensagens ou de equações lógicas. Nestas incluem-se linguagens textuais e algumas linguagens gráficas como os diagramas de blocos, os diagramas de escada e os diagramas de sequência UML. Estas linguagens podem ser facilmente incluídas nos modelos orientados ao objecto na definição de métodos e de interacções entre objectos. No caso das linguagens gráficas é possível visualizar e manipular adequadamente expressões lógicas ou instruções e, no

caso dos diagramas de sequência UML, a sequência de invocações ou envio de mensagens. Não é porém possível com este tipo de linguagens representar conveniente a concorrência nem caracterizar facilmente estados de operação.

As linguagens baseadas em máquinas de estado finitas (FSM), das quais são exemplo a *grafcet*, os diagramas de estados UML, os diagramas de actividades UML e os *Sequencial Function Charts* IEC 61131-3, permitem a definição de estados globais e de regras precisas de evolução de estado. As FSM são o elemento de modelação de comportamento conduzido por eventos mais vulgarizado e permitem a modelação de acções sequenciais, de relações de precedência e paralelismo, bem como de sincronização, definindo comportamentos determinísticos. As grandes vantagens das máquinas de estado encontram-se na sua capacidade de visualização de comportamento e de representação sem ambiguidade.

As redes de Petri (RdP)^[27] são generalizações formais das máquinas de estado finitas que permitem ainda a modelação de acontecimentos síncronos e assíncronos, de comportamentos não determinísticos, de exclusão mútua de recursos e de condições lógicas de transição. As RdP assentam em estados locais, em oposição aos estados globais das FSM, o que evita o problema da explosão de estados relacionado com a modelação de sistemas complexos. A visualização dos estados globais de um sistema não é, porém, imediata nas RdP, sendo necessário para tal recorrer à análise da rede. As RdP permitem, no entanto, a modelação de sistemas concorrentes e de grande dimensão e complexidade com uma notação simples e modelos compactos, possibilitando ainda a determinação por métodos analíticos (sem recurso a simulação) de propriedades tais como: (i) a atingibilidade de um dado estado global partindo de um estado global inicial (esta análise é imediata nas FSM), (ii) a existência de bloqueios globais (*deadlocks*) ou bloqueios locais, (iii) a identificação de limites de capacidade, (iv) a identificação de comportamentos reversíveis e não reversíveis e (v) a análise de desempenho. A simulação de eventos discretos e a obtenção de uma implementação final a partir de um modelo de RdP são, tal como com nas FSM, imediatas.

As RdP e as FSM genéricas, embora apresentem uma semântica precisa, são incompletas e não permitem por si só a geração de modelos interpretados executáveis ou simuláveis. Para tal é conveniente incluir uma linguagem para definição de acções e condições. Embora as RdP e as FSM apresentem extensões que permitem obter estruturação e modularidade e, consequentemente, reutilização (RdP hierárquicas e sub-estados), os mecanismos de herança de comportamento são menos evoluídos do que aqueles disponíveis na modelação estrutural estática orientada ao objecto.

1.4. Representação Gráfica e Interface ao Utilizador

As necessidades de projecto da componente de interface dos sistemas de automação abrangem fundamentalmente os sinópticos, uma vez que outros elementos de interface são vulgarmente sujeitos apenas a parametrização baseada em modelos de dados simples. O desenho de esquemas e diagramas representativos do sistema de automação é necessário também para a componente documental do projecto, sendo os requisitos para desenho diagramático um subconjunto dos requisitos associados ao desenho de sinópticos acrescentado de uma componente de conectividade.

Não é utilizada de forma sistemática, no domínio da engenharia onde se insere este trabalho, uma linguagem comum vocacionada particularmente para aplicação à modelação desta componente, que é normalmente assegurada por editores gráficos WYSIWYG dependentes de aplicações HMI proprietárias. A construção de modelos gráficos bidimensionais vectoriais, estáticos ou dinâmicos, é, porém, um domínio bem tratado da computação gráfica com áreas de aplicação diversificadas. Nesta área destaca-se, entre outras iniciativas, o aparecimento da SVG (*Scalable Vector Graphics*) [28] como linguagem para transacção de modelos gráficos vectoriais animados multi-plataforma.

1.5. Introdução à Linguagem de Modelação para Sistemas de Automação

A ASML é uma linguagem de domínio, declarativa e orientada ao objecto, embora um seu subconjunto apresente características imperativas. A sua definição compreende a agregação de subconjuntos de linguagens conhecidas, modificados com vista à aplicação ao domínio em causa. O seu modelo assenta no uso dos elementos de modelação estrutural estática UML 1.5 [29], nomeadamente os diagramas de classes e os diagramas de objectos. Estes compreendem os elementos fundamentais da linguagem, que suportam a modelação das várias componentes do sistema de automação.

Nos diagramas de classes assenta a modelação de todos os elementos do sistema de automação, nas suas vertentes lógica, física e de parametrização. A base de modelação para suporte ao desenvolvimento conduzido por modelos é suportada pela linguagem com recurso a modelos de classes. Estes modelos, reutilizáveis e evolutivos, compreenderão: (i) as arquitecturas de domínio existentes, nomeadamente no âmbito da IEC 61850 ou outros, (ii) os modelos dos IEDs, estabelecidos pelos fabricantes, e (iv) classes comuns, construídas e partilhadas na organização de integração de sistemas.

Os diagramas de objectos são utilizados como vistas estáticas do estado de um sistema num dado instante. A sua semântica mais importante é, no entanto, a definição de todas as instâncias de classes existentes num dado sistema, bem como do seu estado inicial. O conjunto de todas as instâncias compreende a definição da configuração e parametrização de um sistema de automação específico.

A modelação comportamental assenta na integração de redes de Petri ordinárias hierárquicas interpretadas, estendidas com arcos inibidores e transições temporizáveis². A escolha de redes de Petri assenta essencialmente no facto destas serem consistentemente utilizadas na modelação de sistemas de eventos discretos [23] [30], em sistemas de automação de vários domínios [15] [27] [31] e também no caso particular da automação do sistema eléctrico [32] [33] [34] [35] [36]. A opção por redes de Petri ordinárias, em oposição a redes de Petri orientadas ao objecto, como por exemplo as redes de Petri coloridas [36] [38], foi tomada com a consideração de que as segundas seriam subutilizadas neste domínio de aplicação. Um modelo para definição de expressões e procedimentos, utilizáveis por vários elementos da linguagem, é também

² Uma identificação de classes de redes de Petri, bem como uma introdução à sua teoria, podem ser encontradas em [15].

incluído. O modelo escolhido é semelhante ao de duas das linguagens definidas na IEC 61131-3, a *Structured Text* e a *Function Block Diagram*.

A modelação da componente de interface é realizada através de elementos de interacção que correspondem a elementos gráficos bidimensionais hierárquicos animados, com conectividade e interactividade, utilizáveis na definição de notações alternativas à notação base da linguagem e na definição da função de HMI de um sistema de automação. Os elementos de interacção são definidos exclusivamente neste documento e o seu meta-modelo incorpora conceitos encontrados na SVG 1.1 [39], bem como aqueles encontrados em aplicações de HMI [40] [41].

De forma a dotar a linguagem de alguma capacidade de gestão e expansão, incluem-se também elementos de extensão e de gestão semelhantes aos definidos na UML. Um modelo da linguagem que inclui a sua relação com as linguagens conhecidas que estão na sua génese é apresentado na figura 17.

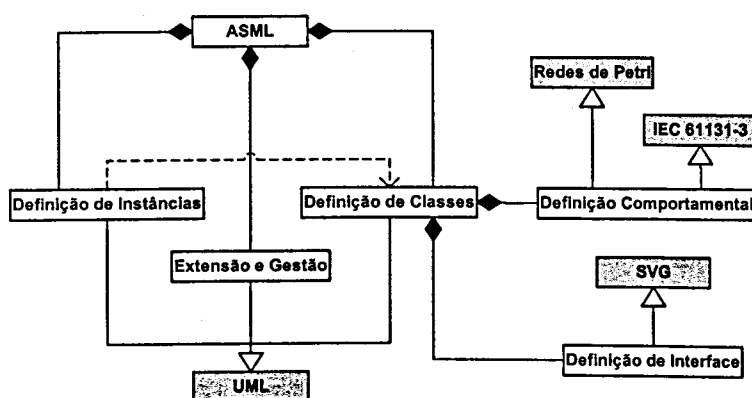


Figura 17. Modelo geral da ASML e suas relações com linguagens existentes.

Dado que a ASML não apresenta, na sua definição essencial, características que a tornem directamente aplicável na definição de modelos de sistemas de automação incluem-se ainda (i) uma biblioteca de funções essenciais para cálculo matemático e manipulação de cadeias de caracteres e (ii) um modelo semântico baseado nas arquitecturas de referência actuais para sistemas de automação do sistema eléctrico de energia.

Capítulo 2

Sintaxe Abstracta e Semântica

Neste capítulo é definido o modelo semântico e de sintaxe abstracta da ASML que incorpora conceitos da UML, da IEC 61131-3 e de redes de Petri. Este modelo será aplicável na definição do *schema* da base de dados de uma ferramenta de engenharia para sistemas de automação, bem como na definição da sua interface ao utilizador. Os modelos criados pelo engenheiro de automação são instâncias deste modelo que podem ser armazenadas na base de dados da ferramenta através da qual são manipulados e transformados.

2.1. Modelo Reduzido

De forma a facilitar a compreensão da sintaxe abstracta da ASML, apresenta-se na figura 18 um modelo reduzido que é composto pelos elementos de modelação concretos mais relevantes, permitindo uma visão de detalhe intermédio.

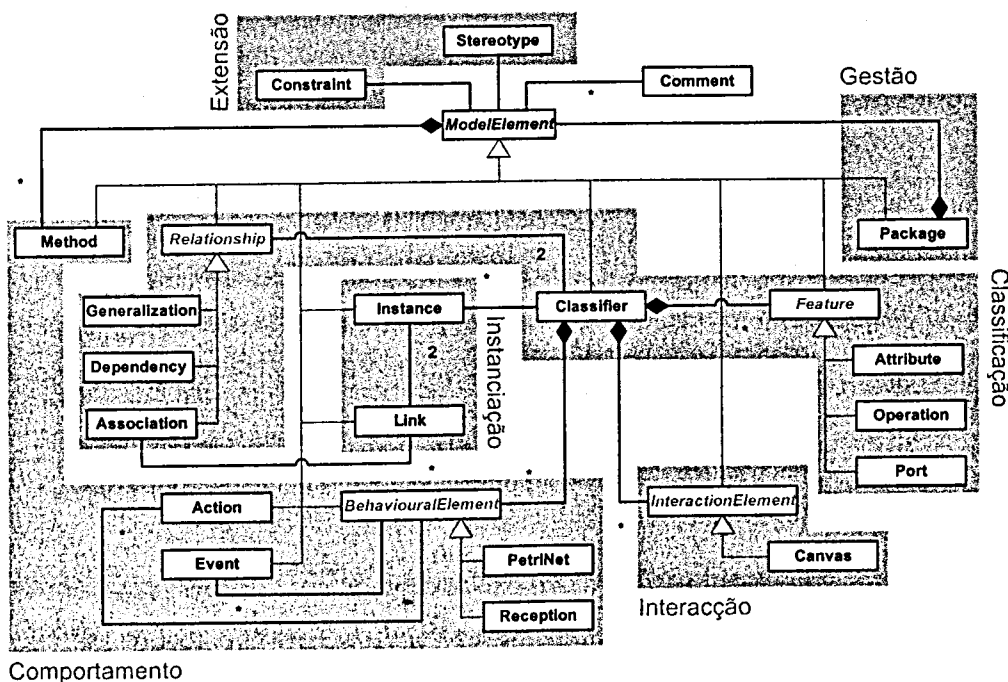


Figura 18. Modelo reduzido da sintaxe abstracta.

O modelo semântico pode ser resumido nas características chave apresentadas na tabela 6.

Tabela 6. Características chave do modelo semântico da ASML.

Subconjunto	Características
Classificação	Classificação generalizável. Estruturação de dados e comportamento. Classificação em atributos (estado), portos (sinais) e operações (procedimentos). Classificação de comportamento e interface com o utilizador.
Interacção	Modelo generalizável de gráficos bidimensionais vectoriais e matriciais, hierárquicos, animados e com conectividade. Herança de atributos visuais na definição de elementos gráficos. Interactividade baseada em animações gráficas e em modelo de eventos do utilizador. Elementos de interacção para modelação da interface com o utilizador e para extensão do léxico gráfico da linguagem. Herança de interacção baseada em elementos herdados e sobreponíveis.
Comportamento	Modelo assíncrono baseado em acções e eventos temporais, associados a sinais ou a alterações do modelo de dados. Modelo síncrono de condições, operações e acções baseada em procedimentos e expressões definidos numa linguagem imperativa. Modelação de estruturas de controlo baseada em redes de Petri (redes ordinárias hierárquicas interpretadas, estendidas com arcos inibidores e transições temporizáveis) e em execução cíclica ou esporádica de procedimentos. Herança de comportamento baseada em elementos herdados e sobreponíveis.
Instanciação	Definição de instâncias de classificadores e associações.
Gestão	Estruturação hierárquica de elementos de modelação com controlo de visibilidade. Identificação de versões. Importação de bibliotecas.
Extensão	Elementos de extensão semântica aplicáveis a elementos de modelação: classificação por estereótipo, definição de atributos semânticos adicionais e restrições.

2.2. Arquitectura

A ASML pressupõe a arquitectura de quatro camadas apresentada na tabela 7, idêntica àquela sobre a qual é posicionada a UML.

Tabela 7. Arquitectura de quarto camadas para meta-modelos.

Camada	Descrição	Exemplos
M3 Meta-meta-modelo	A infra-estrutura para especificação de meta-modelos, ou seja, a linguagem para modelação da linguagem de modelação.	MOF
M2 Meta-modelo	Uma instância de um meta-meta-modelo, isto é, uma linguagem para especificação de modelos.	A linguagem definida neste documento.
M1 Modelo	Uma instância de um meta-modelo, ou seja, um modelo que especifica informação de um dado domínio de aplicação.	Os modelos de sistemas de automação trabalhados pelo engenheiro de automação com recurso à ASML.
M0 Objectos do dispositivo	Uma instância de um modelo, isto é, um conjunto de informação num domínio específico e concreto.	O resultado da transformação de modelos M1 para o modelo de parametrização de um dado IED.

Nesta arquitectura os modelos de uma camada de nível inferior correspondem a concretizações de modelos da camada imediatamente acima. A camada objectos do dispositivo (M0) corresponde a um sistema de elementos com existência própria cujas características estão definidas em elementos de modelação do nível imediatamente superior, por exemplo uma função automática de corte por tensão zero e reposição que corre num dado controlador programável. Os sistemas deste nível correspondem a modelos dependentes dos dispositivos obtidos por transformação automática de modelos M1, definidos pelo engenheiro de automação.

A camada modelo (M1) corresponde aos modelos criados com a ASML. Um modelador de sistemas de automação que constrói uma dada classe de controladores cria um modelo deste nível.

A camada meta-modelo (M2) corresponde à linguagem que é utilizada pelo modelador para criar modelos M1. Este documento define um meta-modelo de nível M2, ou seja, uma linguagem de modelação.

A camada meta-meta-modelo (M3) contém elementos de modelação que permitem definir meta-modelos de linguagens. Note-se porém que a definição da linguagem de nível M2 deste documento não recorre directamente a um meta-meta-modelo como o MOF, mas sim utiliza uma descrição em UML. Esta linguagem pode ser vista, no entanto, como um modelo concreto do MOF.

2.3. Sobre a Definição da Sintaxe Abstracta e da Semântica da Linguagem

A sintaxe abstracta e a semântica da linguagem proposta são, por razões de inteligibilidade, definidas na sua totalidade e de forma não formal. A definição da sintaxe abstracta recorre a diagramas de classes UML, alguns dos quais adaptados da especificação UML 1.5, e a definição semântica é incluída em descrições textuais em linguagem natural das produções sintácticas presentes nos diagramas. Não se apresentando como uma especificação formal, foi preocupação do autor utilizar sempre termos claros e concisos nesta definição e contextualizá-los sempre que os termos utilizados possam ter interpretações múltiplas. No entanto, dada a utilização da UML para definir uma linguagem baseada na UML, torna-se por vezes pouco clara a distinção entre meta-meta-elementos, meta-elementos e elementos. Para melhorar a inteligibilidade do texto são seguidas, sempre que possível, as regras definidas na tabela 8.

Tabela 8. Regras para referências a elementos na definição da ASML.

Elemento	Regra	Exemplos
Meta-meta-elementos (Nível M3) (elementos UML usados para definir a linguagem)	Referências a meta-meta-elementos são evitadas. Para referir meta-meta-elementos são utilizados os nomes em Português dos elementos UML.	o atributo as associações
Meta-elementos (Nível M2) (elementos da própria linguagem)	Os meta-elementos, essencialmente meta-classes, são referidos pelo seu nome (em língua inglesa), em itálico, e é-lhes atribuído o género feminino (de meta-classe).	a <i>Classifier</i> a <i>Stereotype</i>
Elementos (Nível M1) (instâncias de elementos da linguagem)	Para referências a instâncias de meta-classes são utilizados os nomes em língua portuguesas das meta-classes associadas ou, quando tal interfere na legibilidade do texto, pelo nome da meta-classe, mas com o género masculino (de elemento).	uma classe um <i>ModelElement</i>

2.4. Visão Geral do Meta-Modelo

O meta-modelo sintáctico e semântico da linguagem é decomposto em quatro pacotes, tal como ilustrado na figura 19, onde são também apresentadas as interdependências mais importantes. Cada um destes pacotes apresenta um conjunto de elementos de modelação com funções suficientemente distintas: (i) *Foundation*, que estabelece o modelo fundamental da linguagem, (ii) *Model Management*, que define elementos para organizar modelos definidos pela linguagem, (iii) *Behavioural Elements*, onde são definidos os elementos de modelação comportamental e (iv) *Interaction Elements*, elementos de interacção com o utilizador.

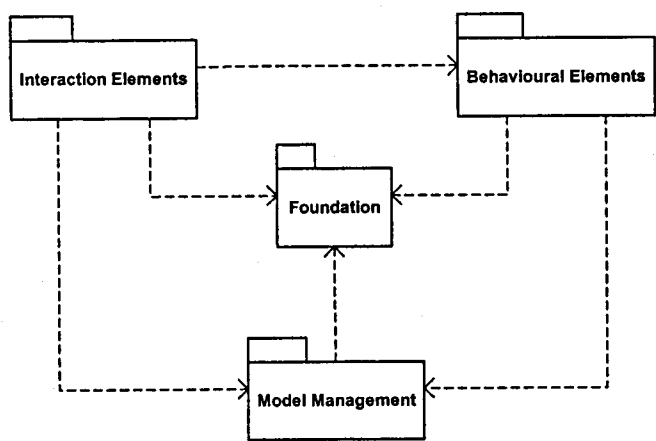


Figura 19. Meta-modelo da linguagem.

2.5. Modelo Fundamental

O modelo fundamental da ASML corresponde ao pacote *Foundation* (*Core*, *Data Types* e *Extension Mechanisms*). Na figura 20 ilustram-se os pacotes deste modelo da linguagem. O modelo fundamental compreende os elementos básicos, os elementos de modelação estrutural, as meta-classes de suporte ao meta-modelo e os elementos de extensão à linguagem.

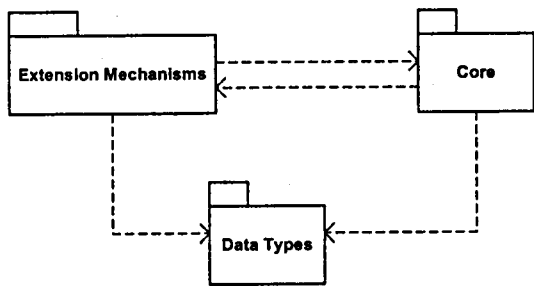


Figura 20. Modelo fundamental da linguagem - *Foundation*.

2.5.1. Pacote Core

As figuras 21 a 25 apresentam o modelo do pacote *Core*. Este modelo apresenta os meta-elementos básicos do meta-modelo da linguagem, bem como os meta-elementos utilizados na modelação estrutural estática.

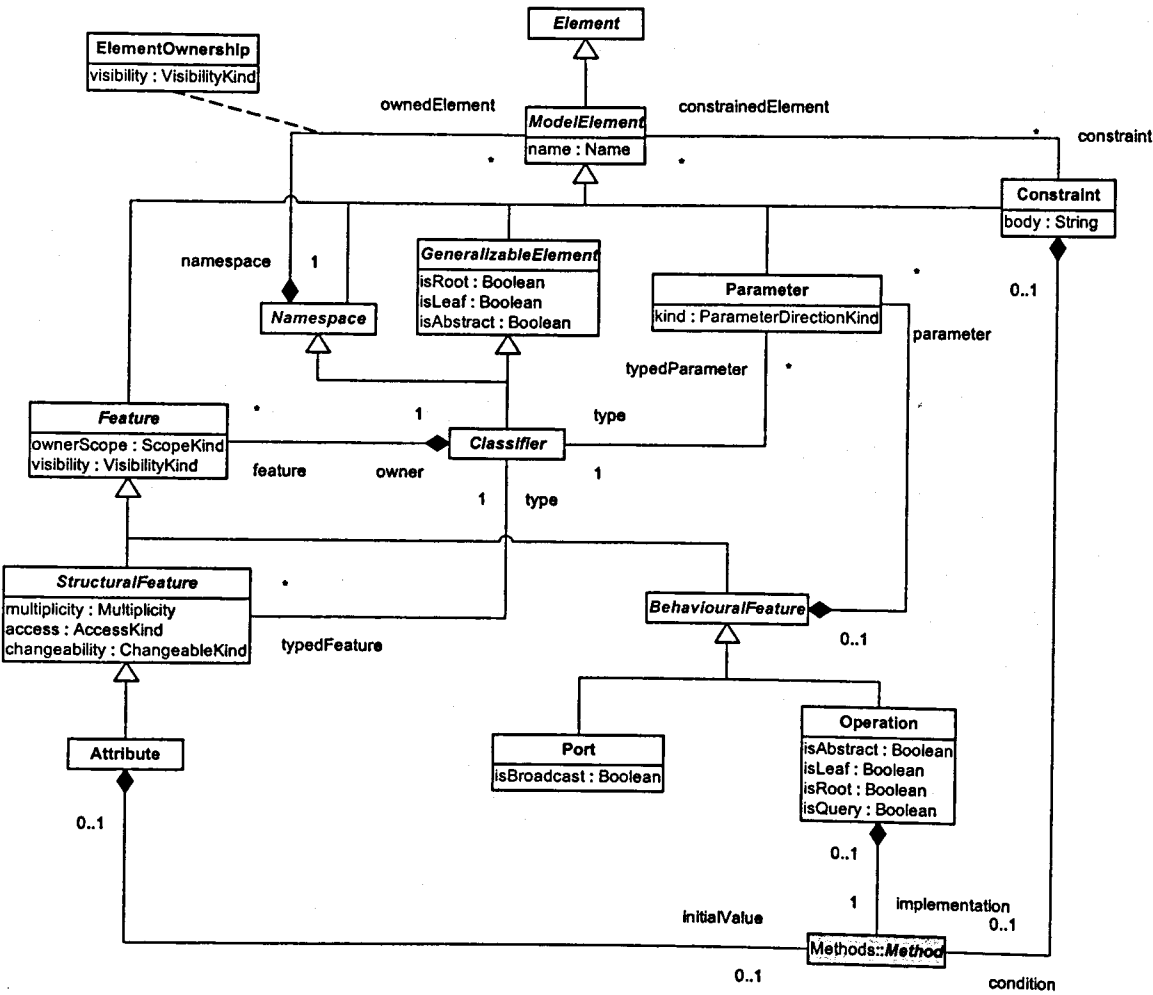


Figura 21. Modelo base do pacote Core.

2.5.1.1. Element

A *Element*, elemento, é uma meta-classe abstracta que generaliza qualquer elemento da linguagem.

Etiquetas	
Documentation	Comentário, explicação ou descrição do elemento. A qualquer elemento pode estar associado um texto explicativo passível de utilização na geração automática de documentação de um projecto.

2.5.1.2. ModelElement

Um *ModelElement* é definido neste meta-modelo como um elemento genérico com nome. A *ModelElement* é, no meta-modelo da ASML, uma meta-classe abstracta que não pode ser instanciada.

Atributos	
<i>name</i>	O nome do elemento de modelação deverá ser, em regra, não vazio e único no seu contexto (a permissão para definição de nomes vazios, ou seja, inexistentes, será explícita na definição da meta-classe concreta do elemento, caso contrário, nomes vazios não são permitidos).
Associações	
<i>clientDependency</i>	Designa todos os <i>Dependency's</i> dos quais o elemento é cliente.
<i>constraint</i>	Indica o conjunto de todos os <i>Constraints</i> que restringem ou afectam o elemento.
<i>namespace</i>	Designa o <i>Namespace</i> que inclui o elemento de modelação. Todos os elementos de modelação estão contidos num e um só <i>Namespace</i> excepto os elementos de topo ou aqueles que são parte integrante de outros elementos. O caminho composto pela concatenação dos meta-atributos <i>name</i> de todos os elementos desde o <i>Namespace</i> de topo até ao elemento corresponde ao identificador único do elemento no universo de modelação.
<i>stereotype</i>	Designa o conjunto de todos os <i>Stereotype's</i> que qualificam o elemento de modelação.
<i>supplierDependency</i>	Designa todos os <i>Dependency's</i> dos quais o elemento é fornecedor.
<i>taggedValue</i>	Designa o conjunto de todos os <i>taggedValue's</i> associados ao elemento.
<i>annotation</i>	O conjunto de <i>Comments</i> que estão associados ao elemento.
<i>presentation</i>	Designa o conjunto de elementos de apresentação que constituem a notação para o elemento. Cada elemento de modelação pode ser representado por um ou mais elementos notacionais (gráficos ou textuais).

Embora um *ModelElement* não seja um *GeneralizableElement* (e portanto não possa participar numa relação de herança num modelo M1), os seus descendentes generalizáveis herdam dessa meta-classe. Os elementos seguintes são herdados pelos descendentes do *ModelElement* que são da classe *GeneralizableElement*.

Características Herdadas	
<i>constraint</i>	Ao descendente aplicam-se todas as restrições dos ascendentes.
<i>presentation</i>	O descendente deve ser apresentado de forma idêntica ao ascendente. No entanto a apresentação pode ser modificada quer na definição de notações alternativas, quer recorrendo às propriedades de interacção.

Características Herdadas (continuação)	
<i>stereotype</i>	O descendente mantém todas os estereótipos dos seus ascendentes embora podendo incluir esterótipos adicionais. Estão de igual forma restringidos pelos <i>Constraint's</i> associados aos <i>Stereotype's</i> dos ascendentes e podem incluir qualquer <i>TaggedValue</i> a eles associado.
<i>taggedValue</i>	As etiquetas de um elemento ascendente num dado modelo definido nesta linguagem são herdadas pelos seus elementos descendentes e não podem ser alteradas.
Características Não Herdadas	
<i>clientDependency</i> <i>supplierDependency</i>	Dependências não podem, por definição, ser herdadas.
<i>name</i>	O nome de um elemento é por definição único e não pode ser herdado.
<i>namespace</i>	Um elemento pode ser incluído num qualquer <i>Namespace</i> independentemente da sua ascendência no modelo.

2.5.1.3. Constraint

Um *Constraint* é uma condição semântica ou restrição concreta. No meta-modelo um *Constraint* aplicado a um ou mais elementos de modelação impõe uma condição que deve ser verdadeira de forma que o modelo seja bem formado. Algumas restrições de um dado modelo podem ser utilizadas para validação do modelo uma vez que se devem verificar para qualquer estado possível do sistema modelado.

Atributos	
<i>body</i>	Uma descrição textual da restrição.
Associações	
<i>condition</i>	O predicado que se deve verificar para que o modelo definido esteja bem formado. Esta associação corresponde a um método de leitura com um único parâmetro de retorno (de tipo booleano) que define concretamente o predicado de forma a permitir a validação automática do modelo.
<i>constrainedElement</i>	O conjunto de elementos afectados pelo <i>Constraint</i> .

2.5.1.4. GeneralizableElement

Um elemento cuja meta-classe seja descendente de *GeneralizableElement* pode participar em relações de herança. A *GeneralizableElement* é, no meta-modelo da linguagem, uma meta-classe abstracta que não pode ser instanciada.

Atributos	
<i>isAbstract</i>	Estabelece se, no modelo, podem ou não ser geradas instâncias definidas por este elemento. Caso este atributo seja verdadeiro o elemento é considerado abstracto e apenas poderão ocorrer instâncias de elementos descendentes que sejam não abstractos. Caso contrário podem ocorrer instâncias deste elemento e este denomina-se concreto. Assume-se que se um elemento do modelo é abstracto então não possui informação de modelação suficiente para gerar instâncias. Por defeito este meta-atributo tem o valor falso.
<i>isLeaf</i>	Indica se o elemento pode ou não ser especializado. Caso este atributo seja verdadeiro o elemento de modelação não pode ser especializado, ou seja, não pode possuir descendentes. Por defeito este meta-atributo tem o valor falso.
<i>isRoot</i>	Indica se o elemento pode ou não ser generalizado. Caso este atributo seja verdadeiro o elemento de modelação não pode ser generalizado, ou seja, não pode possuir ascendentes. Por defeito este meta-atributo tem o valor falso.
Associações	
<i>generalization</i>	Designa o conjunto de <i>Generalization's</i> cujos elementos <i>parent</i> são ascendentes imediatos do elemento.
<i>specialization</i>	Designa o conjunto de <i>Generalization's</i> cujos elementos <i>child</i> são descendentes imediatos do elemento.
Características Não Herdadas	
<i>isRoot</i> , <i>isLeaf</i> e <i>isAbstract</i>	Não podem ser herdadas uma vez que definem o próprio mecanismo de herança.
<i>generalization</i> e <i>specialization</i>	Herdadas implicitamente pela hierarquia de herança.

2.5.1.5. Namespace

Esta meta-classe generaliza elementos que contêm outros elementos. Um *Namespace* é um elemento que agrupa, de forma não ordenada, um conjunto de elementos de modelação de qualquer meta-classe. Um elemento contido num *Namespace* deverá ter um *name* único no universo desse *Namespace* independentemente da sua meta-classe, exceptuando aqueles cujo *name* seja vazio.

Associações	
<i>ownedElement</i>	Designa todos os elementos que compõem o <i>Namespace</i> .

2.5.1.6. ElementOwnership

Esta meta-classe classifica a visibilidade de um elemento de modelação incluído num dado *Namespace*.

Associações	
<i>visibility</i>	Especifica se o elemento de modelação pode ser visto e referenciado por outros <i>ModelElements</i> de um modelo: • <i>public</i> – Qualquer elemento, mesmo exterior ao <i>Namespace</i>, pode visualizar o <i>ownedElement</i>. • <i>protected</i> – O próprio <i>ownedElement</i>, os elementos nele contidos e os descendentes do <i>ownedElement</i> podem visualizá-lo. • <i>private</i> – Apenas o próprio <i>ownedElement</i> ou elementos nele contidos podem visualizá-lo. Este é o valor por defeito. • <i>package</i> – O próprio <i>ownedElement</i>, os elementos nele contidos e os elementos contidos no <i>namespace</i> podem visualizá-lo.

2.5.1.7. Classifier

Um *Classifier* é um elemento que apresenta características de estrutura e de comportamento e que pode participar em associações, é um classificador de objectos. O meta-modelo define especializações concretas desta meta-classe, nomeadamente *Signal*, *Class*, *DataType's* e *Interface*. Uma vez que alguns *Classifier's* podem conter elementos de modelação, um *Classifier* é também um *Namespace*.

Qualquer classificador pode ser considerado uma estrutura caso se verifiquem as seguintes condições: (i) seja passivo, (ii) seja concreto, (iii) não apresente relações de associação na sua definição e (iv) qualquer das suas propriedades estruturais tenha, como tipo, também uma estrutura.

Etiquetas	
<i>persistence</i>	Esta qualificação especifica se o estado de instancias do classificador é mantido entre execuções (<i>persistent</i>) ou não (<i>transitory</i>).
Associações	
<i>feature</i>	O conjunto de características do <i>Classifier</i> .
<i>association</i>	O <i>associationEnd</i> que estabelece o papel do elemento numa dada <i>association</i> na qual participa.
<i>typedFeature</i>	O conjunto dos <i>StruturalFeature's</i> com o tipo do elemento.
<i>typedParameter</i>	O conjunto dos <i>Parameter's</i> com o tipo do elemento.

Características Herdadas	
<i>association</i>	Os descendentes de um <i>Classifier</i> herdam todas as associações dos seus ascendentes.
<i>feature</i>	Os atributos e portos de um ascendente são parte integrante do descendente e não podem ser declarados novamente nem sobrepostos. As operações de um ascendente são também parte integrante do descendente e podem ser redeclaradas eventualmente modificando a sua posição na hierarquia (<i>isRoot</i>, <i>isLeaf</i>, <i>isAbstract</i>), bem como a sua estrutura de parâmetros, isto é, pode ocorrer sobreposição e sobrecarga de operações. Uma operação sobreposta apresentará necessariamente o mesmo valor do atributo <i>isQuery</i> do ascendente.

2.5.1.8. Feature

Um *Feature* é uma característica ou propriedade de um *Classifier*. No meta-modelo é uma meta-classe abstracta que generaliza propriedades comportamentais ou estruturais.

Atributos	
<i>name</i>	(Herdado de <i>ModelElement</i>) O nome de um <i>Feature</i> deve ser único, juntamente com os <i>associationEnd</i> 's opostos de todas as associações em que o elemento participe, no universo do <i>Classifier</i> que caracterizam.
<i>ownerScope</i>	Especifica se o <i>feature</i> ocorre em cada uma das instâncias do <i>Classifier</i> (<i>instance</i>) ou se, pelo contrário, é <i>feature</i> do <i>Classifier</i> (<i>classifier</i>) e ocorre apenas uma vez no âmbito da cada concretização do modelo M1.
<i>visibility</i>	Especifica de que forma o <i>feature</i> pode ser usado por outros <i>Classifiers</i> de um modelo: • <i>public</i> – Qualquer <i>Classifier</i> com visibilidade para o classificador pode utilizar o <i>feature</i>. • <i>protected</i> – Qualquer descendente do classificador pode utilizar o <i>feature</i>. • <i>private</i> – Apenas o próprio <i>Classifier</i> pode utilizar o <i>feature</i>. Este é o valor por defeito do meta-atributo. • <i>package</i> – Qualquer <i>Classifier</i> pertencente ao mesmo <i>Package</i> (ou qualquer <i>Package</i> a ele pertencente) ao qual pertence o <i>owner</i> do <i>feature</i> pode utilizá-lo.
Associações	
<i>owner</i>	O <i>Classifier</i> que declara o <i>feature</i> e aquele do qual o <i>feature</i> é propriedade.

2.5.1.9. StructuralFeature

Um *StructuralFeature* é uma propriedade estrutural de um *Classifier*. No meta-modelo é uma meta-classe abstracta.

Atributos	
<i>access</i>	Estabelece o tipo de acesso: <i>write</i> – O <i>StructuralFeature</i> apenas pode ser escrito quando acedido do exterior. O acesso interno é de escrita ou leitura. <i>read</i> – O <i>StructuralFeature</i> apenas pode ser lido quando acedido do exterior. O acesso interno é de escrita ou leitura. <i>both</i> – Quer a escrita quer a leitura são permitidas. Este é o valor por defeito do meta-atributo. O acesso às propriedades estruturais do classificador do atributo é condicionado pela conjugação do tipo de acesso do classificador com o tipo de acesso da propriedade.
<i>changeability</i>	Estabelece se o atributo pode ser alterado após a criação da instância do <i>Classifier</i> que o possui (<i>changeable</i>) ou se é uma constante (<i>frozen</i>). Por defeito este meta-atributo tem o valor <i>changeable</i>.
<i>multiplicity</i>	Estabelece a multiplicidade do <i>StructuralFeature</i> no <i>Classifier</i> que o possui. A multiplicidade deve corresponder a um valor fixo não nulo. Por defeito este meta-atributo tem o valor 1.
Associações	
<i>type</i>	O tipo ou classificador do elemento, isto é, o <i>Classifier</i> que classifica as instâncias que são valores do <i>StructuralFeature</i>.

2.5.1.10. Attribute

A *Attribute* é uma meta-classe concreta do meta-modelo que concretiza *StructuralFeature* e cujas instâncias definem propriedades de um *Classifier*, ou seja, um atributo. O tipo de um atributo deve ser obrigatoriamente uma estrutura das meta-classes *Class* ou *DataType*.

Etiquetas	
<i>persistence</i>	Esta qualificação especifica se o estado dos atributos nas instâncias do <i>Classifier</i> que qualifica é mantido entre execuções (<i>persistent</i>) ou não (<i>transitory</i>). O estado dos atributos é persistente apenas se o classificador for também persistente. Por defeito este meta-atributo tem o valor <i>transitory</i>.

Associações	
<i>initialValue</i>	Um método de leitura com um único parâmetro de retorno que especifica o valor inicial por defeito do atributo, calculado em tempo de modelação. O valor inicial do atributo pode ser modificado em instâncias do classificador. O tipo do parâmetro do método é idêntico ao tipo do elemento.

2.5.1.11. BehaviouralFeature

Um *BehaviouralFeature* é uma propriedade comportamental de um *Classifier*. No meta-modelo é uma meta-classe abstracta.

Atributos	
<i>isQuery</i>	Se verdadeiro especifica que o <i>BehaviouralFeature</i> não altera o estado das instâncias do <i>Classifier</i> que o possui. Caso seja falso o comportamento do <i>BehaviouralFeature</i> não é, nesse aspecto, especificado. Por defeito este meta-atributo tem o valor falso.
Associações	
<i>parameter</i>	Uma lista ordenada de parâmetros.

2.5.1.12. Parameter

Um *Parameter* é um argumento, isto é, uma variável usada da definição de propriedades comportamentais de classificadores.

Atributos	
<i>name</i>	(Herdado de <i>ModelElement</i>) O nome de um <i>Parameter</i> deve ser único no universo da lista de parâmetros onde está contido.
<i>kind</i>	Especifica o tipo de parâmetro: • <i>in</i> – Um parâmetro de entrada, valor por defeito. • <i>out</i> – Um parâmetro de saída. • <i>inout</i> – Um parâmetro de entrada/saída. • <i>return</i> – Um parâmetro de saída considerado parâmetro de retorno.
Associações	
<i>type</i>	O <i>Classifier</i> que designa o tipo ao qual o parâmetro deve ser conforme. O tipo de um parâmetro é sempre uma estrutura.

2.5.1.13. Port

Um *Port* estabelece um ponto de acesso assíncrono a um classificador. A definição de um porto é realizada unicamente no classificador considerado servidor de um dado serviço, especificado por elementos comportamentais a ele associados. A lista de parâmetros de um porto corresponde ao conjunto de sinais que podem ser enviados (com o tipo *out*) ou recebidos (com o tipo *in*) pelo classificador servidor (parâmetros com os tipos *return* ou *inout* não têm significado semântico, pelo que este tipo não pode ser utilizado). O tipo dos parâmetros associados a um *Port* é sempre um *Signal*.

Atributos	
<i>isBroadcast</i>	Se o porto é um porto de <i>broadcast</i> então o tipo de todos os parâmetros é <i>out</i> e cada sinal enviado através do porto é entregue em todos os elementos que incluem eventos de recepção do sinal. Caso o porto não seja do tipo <i>broadcast</i>, então as interacções definidas através de troca de sinais entre instâncias correspondem a ligações ponto-a-ponto. Este tipo de portos pressupõe um modelo de serviço do tipo pergunta/resposta.

2.5.1.14. Operation

A *Operation* (operação) é uma meta-classe concreta que estabelece uma característica comportamental síncrona especificada num método. A lista de parâmetros de uma operação corresponde à definição de argumentos passados ou devolvidos na execução do método que implementa a operação. Apenas poderá existir um parâmetro de tipo *return* na lista de parâmetros. Apenas estruturas das meta-classes *Class* ou *DataType* podem ser tipos de parâmetros.

Atributos	
<i>isAbstract</i>	Se este meta-atributo apresentar o valor verdadeiro o <i>Operation</i> não apresenta implementação no <i>Classifier</i> que o possui e esta deve ser fornecida em descendentes do <i>Classifier</i>. Um <i>Classifier</i> que apresente este tipo de operações é obrigatoriamente abstracto. Se falso o <i>Operation</i> possui implementação no <i>Classifier</i>. Por defeito este meta-atributo tem o valor falso.
<i>isLeaf</i>	Caso este meta-atributo apresente o valor verdadeiro um <i>Classifier</i> descendente do <i>Classifier</i> que possui o <i>Operation</i> não pode declarar um <i>Operation</i> com o mesmo nome nem sobrepor uma implementação. Se falso a sobreposição em descendentes é permitida. Por defeito este meta-atributo tem o valor falso.
<i>isRoot</i>	Se este meta-atributo apresentar o valor verdadeiro um <i>Classifier</i> ascendente do <i>Classifier</i> que possui o <i>Operation</i> não pode declarar um <i>Operation</i> com o mesmo nome nem definir a sua implementação. Se falso o <i>Operation</i> pode ser declarado em ascendentes. Por defeito este meta-atributo tem o valor falso.
Associações	
<i>implementation</i>	O método que define o elemento, caso o elemento não seja abstracto. Caso o atributo <i>isQuery</i> seja verdadeiro este método deverá ser um método de leitura.

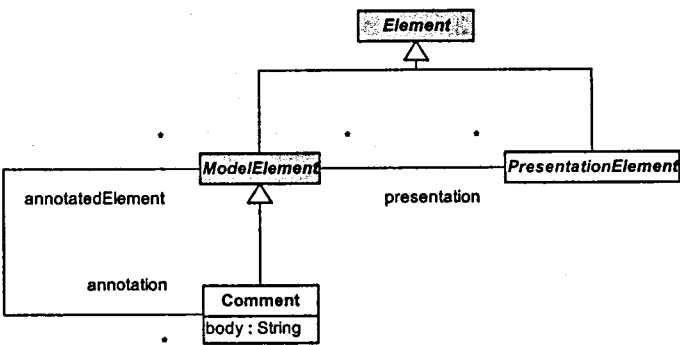


Figura 22. Elementos adicionais ao modelo base do pacote Core.

2.5.1.15. Comment

Uma anotação do modelo que pode ser associada a elementos de modelação e consiste num texto com significado semântico apenas para o modelador, ou seja, um comentário.

Atributos	
body	O comentário da anotação.
name	(Herdado de ModelElement) O nome do elemento pode ser vazio.
Associações	
annotatedElement	O conjunto de elementos aos quais se refere a anotação.

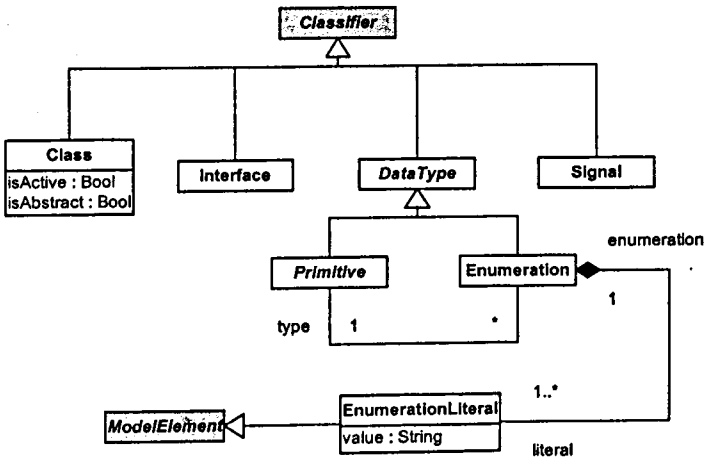


Figura 23. Concretizações de classificadores do pacote Core.

2.5.1.16. PresentationElement

Os *PresentationElement* correspondem aos elementos notacionais da linguagem.

2.5.1.17. Class

Um *Class* define um conjunto de objectos com as mesmas propriedades, ou seja, uma classe de objectos. Quanto a relações de herança uma classe pode realizar um ou mais interfaces e especializar uma outra classe, isto é, a herança múltipla está limitada à realização de interfaces. Uma classe pode participar em associações e incluir qualquer *Feature*. Uma classe define objectos em regra passivos, exceptuando se a ela for associado um elemento de modelação comportamental que estabeleça fluxos de controlo próprios, denominando-se activa. Uma classe, como *Namespace*, pode conter outros elementos, embora estejam limitados a elementos de modelação comportamental, de interacção ou outros *Classifiers*.

Atributos	
<i>isActive</i>	Uma classe activa, ou seja, uma classe cujas instâncias apresentam comportamento próprio. Por defeito este meta-atributo tem o valor falso.
<i>isAbstract</i>	Uma classe abstracta, ou seja, uma classe que contém elementos abstractos e não pode gerar instâncias.

2.5.1.18. Interface

Um *Interface* é uma definição abstracta de comportamento (interface) que pode ser concretizada numa classe, isto é, realizada num *Class*. A um *Interface* podem estar associadas apenas *BehavioralFeature's* abstractas e participar apenas em associações cujo extremo oposto não é navegável. Uma interface pode conter elementos de interacção.

2.5.1.19. DataType

É um tipo cujos valores são puros, isto é, não possuem identidade tal como os objectos. Um *DataType* não pode possuir qualquer *Feature*, participar em associações, conter outros elementos de modelação ou participar em relações de generalização.

2.5.1.20. Signal

Um *Signal* (sinal) é uma especificação de um estímulo assíncrono instantâneo comunicado entre instâncias, ou seja, uma mensagem assíncrona. Um sinal é composto por parâmetros que correspondem a *Attribute's* do *Classifier* e não

pode conter qualquer outro tipo de *Feature* ou elemento de modelação. Os parâmetros de um sinal são obrigatoriamente estruturas. A única relação que pode ser estabelecida entre sinais é a relação de generalização.

2.5.1.21. Primitive

Esta classe do meta-modelo generaliza os tipos básicos incluídos na linguagem, apresentados na figura 24 e tabela 9. Estes tipos primitivos foram escolhidos com base nos tipos vulgarmente disponíveis em sistemas de automação do sistema eléctrico.

Tabela 9. Tipos primitivos da linguagem.

Nome	Valores	Descrição	Valor Inicial
<i>Bool</i>	{verdadeiro, falso}	Trata-se de um valor booleano	FALSE
<i>Int8</i>	[-128, 127]	Correspondem a valores numéricos inteiros de 1, 2, 4 ou 16 octetos.	
<i>Int16</i>	[-32768, 32767]		
<i>Int32</i>	[-21474836848, 21474836847]		
<i>Int128</i>	$[-2^{127}, (2^{127})-1]$		
<i>UInt8</i>	[0, 255]	Correspondem a valores numéricos inteiros sem sinal de 1, 2 ou 4 octetos.	0
<i>UInt16</i>	[0, 65535]		
<i>UInt32</i>	[0, 4294967295]		
<i>Float32</i>	Especificado por IEEE 754	Valor numérico de vírgula flutuante IEEE 754	
<i>Float64</i>			
<i>String</i>	Qualquer conjunto de caracteres	Trata-se de um cadeia de caracteres com formato dependente da plataforma de <i>deployment</i> (considera-se Unicode como formato abrangente).	''
<i>Time</i>	Resolução aproximada de 60ns.	É um instante temporal absoluto definido por um inteiro de 4 octetos que representa o número de segundos desde 01/01/1970 00:00:00 e um inteiro de 3 octetos sem sinal que corresponde à fracção dentro do segundo.	T#1970-1-1 0:0:0.0.0.0
<i>TimePeriod</i>	Resolução aproximada de 60ns.	Trata-se de um período de tempo positivo ou negativo definido por um inteiro de 4 octetos que representa o número de segundos e um inteiro de 3 octetos sem sinal que corresponde à fracção dentro do segundo.	T#0s

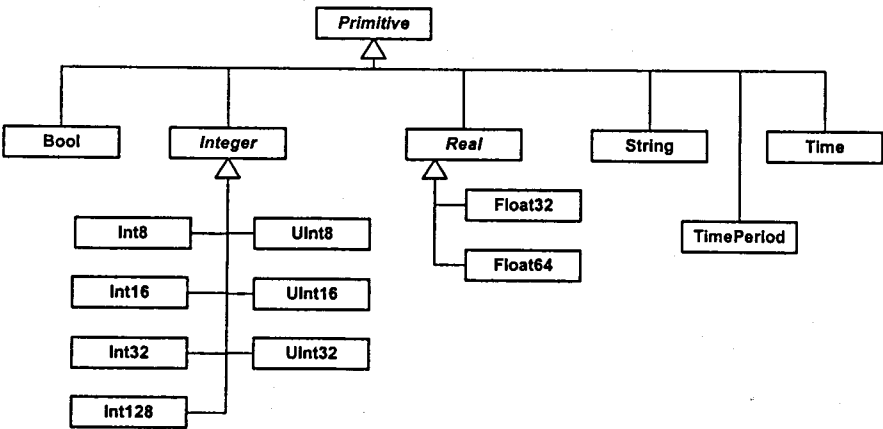


Figura 24. Tipos primitivos da linguagem.

2.5.1.22. Enumeration

Define um enumerado, isto é, um *DataType* cujos valores possíveis são definidos através de uma lista de *EnumerationLiteral*s. A um *Enumeration* está associado um tipo primitivo que define, conjuntamente com a lista de *EnumerationLiteral*s, os seus valores.

Associações	
<i>literal</i>	A lista ordenada e não vazia de valores possíveis para uma dada instância de <i>Enumeration</i> .
<i>type</i>	Estabelece o <i>DataType</i> primitivo do elemento.

2.5.1.23. EnumerationLiteral

Define um valor contido na lista de valores possíveis de um *Enumeration*. O nome, herdado de *ModelElement*, estabelece o identificador do valor. No meta-modelo da linguagem um *EnumerationLiteral* corresponde a um dos valores possíveis do tipo do *Enumeration* que o contém. O nome de cada *EnumerationLiteral* é único no contexto do *EnumerationLiteral* do qual é parte integrante, embora o seu valor possa apresentar duplicados.

Atributos	
<i>value</i>	O valor de um dos valores possíveis do tipo associado ao <i>Enumeration</i> do qual é parte integrante.
Associações	
<i>enumeration</i>	O <i>Enumeration</i> do qual é parte integrante.

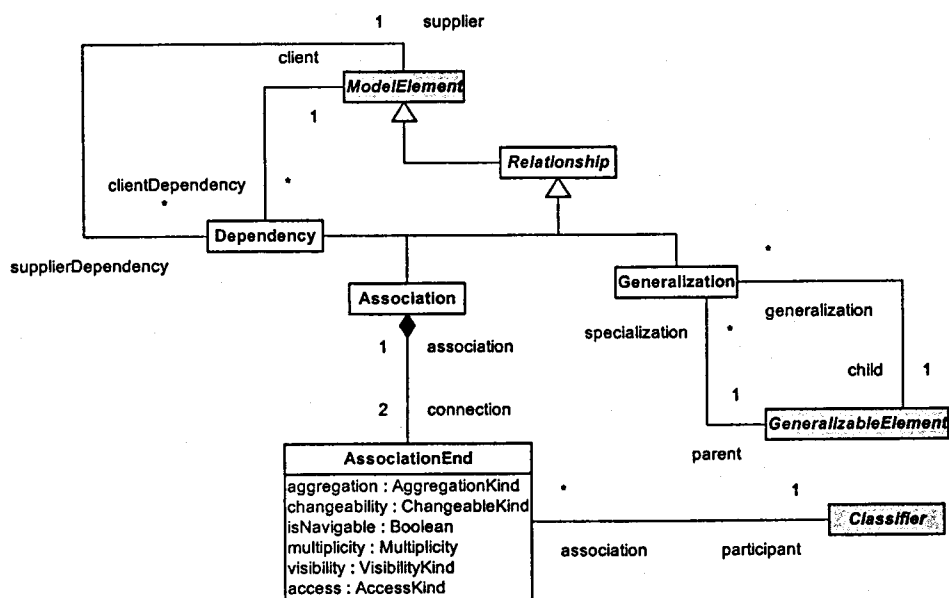


Figura 25. Relações do pacote *Core*.

2.5.1.24. Relationship

Um *Relationship* é uma relação genérica entre elementos de modelação. É uma meta-classe abstracta do meta-modelo da linguagem. As relações são binárias, ou seja, estabelecem uma relação entre dois elementos de modelação. Os nomes de relações podem ser vazios. Em regra uma relação pode apenas ser estabelecida entre elementos de modelação da mesma meta-classe.

Atributos	
<i>name</i>	(Herdado de <i>ModelElement</i>) Dois ou mais <i>Relationship</i> 's podem ter nomes idênticos.

2.5.1.25. Dependency

Uma dependência é uma relação orientada e estabelece que uma alteração ao elemento de modelação alvo poderá ter impacto no elemento fonte. Esta relação pode ser estabelecida entre instâncias de meta-classes distintas. As relações de dependência num dado modelo M1 podem ser explícitas, caso estejam identificadas por um *Dependency*, ou implícitas, caso existam mas não ocorra qualquer *Dependency* que as identifique.

Estereótipos	
«friend»	A dependência modela uma relação de permissão de acesso pelo elemento fonte ao elemento alvo. Relações deste tipo sobrepõem-se aos meta-atributos de visibilidade do elemento fonte.
Associações	
client	O elemento afectado pela dependência, denominado fonte.
supplier	O elemento alvo, inverso de <i>client</i> .

2.5.1.26. Generalization

Uma relação entre um elemento com características mais genéricas (ascendente) e um outro de características mais específicas (descendente) é modelada com recurso a um *Generalization*. O elemento especializado é consistente com o elemento genérico, pode substituí-lo sintacticamente podendo demonstrar no entanto características adicionais. Esta *Relationship* define relações de herança, isto é, especialização e generalização. Uma relação de generalização apenas pode ser aplicada a elementos generalizáveis da mesma meta-classe concreta (à excepção de relações entre interfaces e classes). À especialização de interfaces ou classes abstractas por classes concretas denomina-se realização, na medida em que a classe, para ser concreta, deve fornecer implementações dos elementos comportamentais abstractos definidos no ascendente.

Associações	
child	O elemento descendente na relação de herança.
parent	O elemento ascendente na relação de herança.

2.5.1.27. Association

Um *Association* estabelece uma relação semântica entre dois *Classifier's* definida pelos *AssociationEnd* da associação. Esta relação define ligações entre instâncias dos *Classifier* envolvidos.

Associações	
connection	Os <i>AssociationEnd</i> da relação de associação.

2.5.1.28. AssociationEnd

Um *AssociationEnd*, parte integrante de um *Association*, possui um nome e estabelece as características de um ponto de ligação dessa associação.

Atributos	
<i>aggregation</i>	Quando atribuído a um ponto de ligação (alvo) de uma associação estabelece o tipo de associação entre o classificador alvo e o classificador ligado no extremo oposto (fonte): <i>none</i> – O classificador alvo não é uma composição de classificadores fonte. A associação pode ser vista como uma referência. <i>composite</i> – O classificador alvo é uma composição de classificadores fonte pelo que o tipo de agregação fonte deve ser <i>none</i>. Instâncias do classificador fonte são parte integrante de instâncias do alvo.
<i>isNavigable</i>	Quando colocado num extremo alvo estabelece se é ou não possível navegar de instâncias do classificador fonte para o classificador alvo. Por defeito este meta-atributo tem o valor falso.
<i>multiplicity</i>	Quando colocado no extremo alvo estabelece a multiplicidade de instâncias da ligação do extremo alvo. Por defeito este meta-atributo tem o valor 1.
<i>visibility</i>	Especifica a visibilidade do alvo face do lado da fonte, de forma semanticamente idêntica à visibilidade de propriedades de classificadores: <i>public</i> – Qualquer <i>Classifier</i> com visibilidade para o elemento fonte pode utilizar o alvo (de forma idêntica a um atributo público). <i>protected</i> – Qualquer descendente do elemento fonte pode utilizar o alvo (de forma idêntica a um atributo protegido). <i>private</i> – Apenas o próprio <i>Classifier</i> fonte pode utilizar o alvo (de forma idêntica a um atributo privado). Este é o valor por defeito do meta-atributo. <i>package</i> – Qualquer <i>Classifier</i> pertencente ao mesmo <i>Package</i> (ou qualquer <i>Package</i> a ele pertencente) ao qual pertence a fonte pode utilizar o alvo (de forma idêntica a um atributo com visibilidade de pacote).
<i>access</i>	Estabelece o tipo de acesso ao classificador alvo através do classificador fonte desde o exterior deste. O acesso desde o interior do classificador fonte é sempre de escrita e leitura. O tipo de acesso pode ser um de: <i>write</i> – Apenas pode ser escrito. <i>read</i> – Apenas pode ser lido. <i>both</i> – Quer a escrita quer a leitura são permitidas. Caso o tipo de acesso seja <i>read</i> as propriedades comportamentais com o atributo <i>isQuery</i> falso do classificador alvo não podem ser referidas. Nos restantes casos qualquer propriedade comportamental poderá ser referidas através da associação. O acesso às propriedades estruturais do classificador alvo é condicionado pela conjugação do tipo de acesso do <i>associationEnd</i> com o tipo de acesso da propriedade.

Atributos (continuação)	
<i>name</i>	(Herdado de <i>ModelElement</i>) Estabelece o nome do papel das instâncias associadas ao extremo da ligação. Pode ser usado em expressões como se de um nome de atributo se tratasse. O nome do papel pode ser omitido apenas se a ligação não for navegável através desse extremo, caso contrário deve ser único no universo do <i>Classifier</i> do extremo oposto.
Associações	
<i>participant</i>	O classificador associado ao extremo da associação.
<i>association</i>	A associação da qual o extremo é parte integrante.

2.5.2. Pacote DataTypes

Este pacote define tipos de dados utilizados na definição do meta-modelo da linguagem (figura 26). Estes tipos de dados compreendem tipos elementares conhecidos e tipos enumerados cuja semântica é detalhada ao longo deste capítulo. Neste ponto descrevem-se apenas os elementos não completamente detalhados noutros pontos do capítulo.

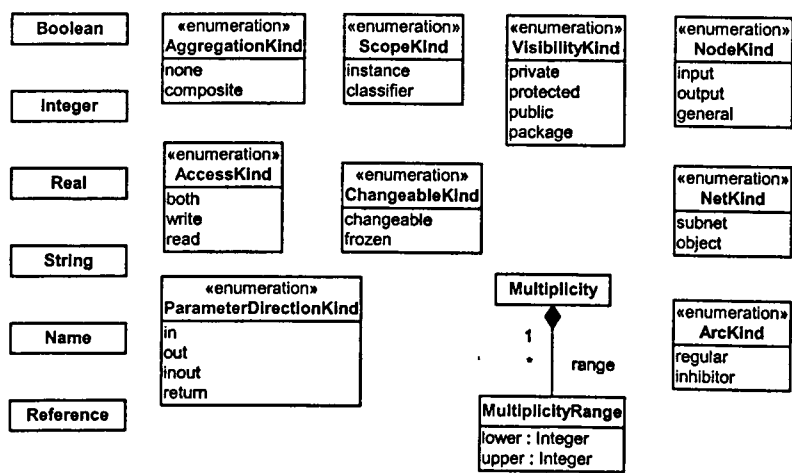


Figura 26. Pacote *DataTypes*.

2.5.2.1. Multiplicity

Um *Multiplicity* é utilizado na definição de multiplicidades. A multiplicidade de um dado elemento é inteira, não negativa e pode ser fixa ou variável, caso em que pode compreender um ou mais intervalos de valores.

Associações	
range	O conjunto de intervalos de valores que define a multiplicidade.

2.5.2.2. MultiplicityRange

Um *MultiplicityRange* define um intervalo de valores.

Atributos	
lower	O limite inferior não negativo do intervalo (inclusivo). Inferior ou igual a <i>upper</i> .
upper	O limite superior não negativo do intervalo (inclusivo). O limite superior pode ser omitido, caso em que qualquer valor inteiro pode ser limite superior.

2.5.3. Pacote Extension Mechanisms

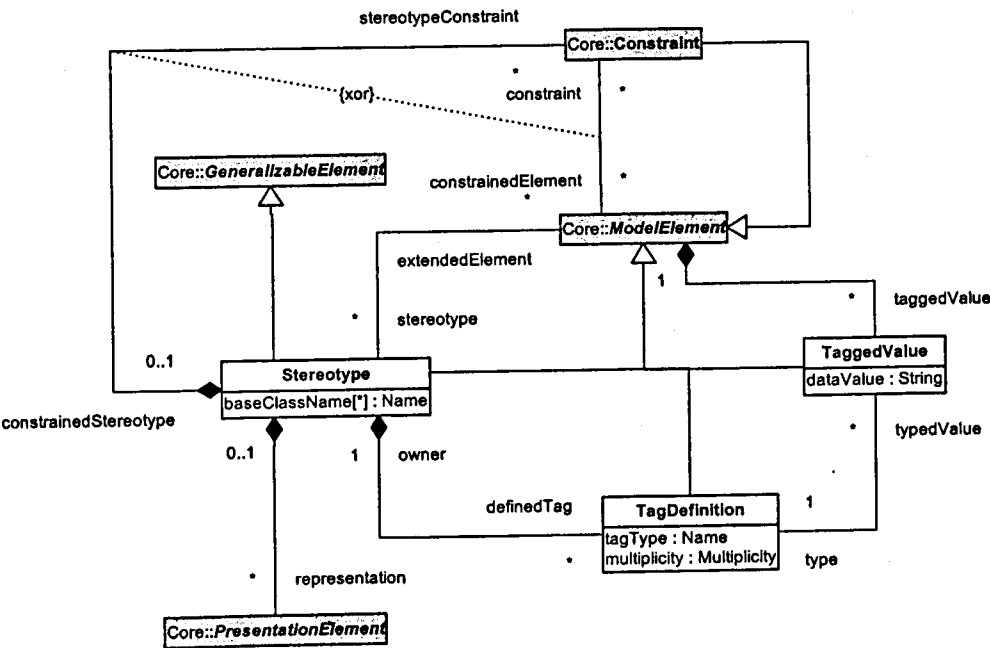


Figura 27. Pacote Extension Mechanisms.

Este pacote define elementos que permitem ampliar, de forma ligeira, quer a semântica quer a notação visual da linguagem. A inclusão destes meta-elementos pretende dotar a ASML da possibilidade de expansão sem haver necessidade de a redefinir. Isto permite aumentar a estabilidade dos componentes nucleares de uma ferramenta baseada na linguagem. Alguns elementos aqui apresentados são na realidade elementos de nível M2, embora sejam manipulados pelo modelador.

2.5.3.1. Stereotype

Um *Stereotype*, estereótipo, permite marcar ou classificar elementos de modelação de forma a alterar ou precisar a sua semântica. Um estereótipo define restrições e novos atributos aplicáveis a uma dada meta-classe do meta-modelo da linguagem. Os estereótipos não podem ser aplicados a outros estereótipos.

Atributos	
<i>baseClass</i>	O conjunto de nomes de meta-classes da linguagem às quais se pode aplicar o estereótipo.
Associações	
<i>extendedElement</i>	O conjunto de elementos de modelação de um dado modelo que estão marcados pelo estereótipo. Cada elemento deverá ser uma instância de uma das meta-classes identificadas em <i>baseClass</i> .
<i>definedTag</i>	Especifica o conjunto de <i>TagDefinition's</i> associadas ao estereótipo. Um elemento marcado pelo estereótipo (pertencente a <i>extendedElement</i>) pode definir <i>TaggedValue's</i> dos tipos de elementos de <i>definedTag</i> .
<i>stereotypeConstraint</i>	Designa o conjunto de <i>Constraint's</i> que afectam qualquer elemento de modelação marcado pelo estereótipo.
<i>representation</i>	Representações alternativas do elemento de modelação marcado pelo estereótipo. A definição de representações alternativas com recurso a elementos de interacção permite estabelecer novas notações para elementos marcados pelo estereótipo.
Características Herdadas	
<i>definedTag</i>	Todos os <i>TagDefinition's</i> do estereótipo estão também associados aos seus descendentes.
<i>stereotypeConstraint</i>	As restrições associadas ao ascendente estão também associadas ao descendente.
<i>representation</i>	As representações são características herdadas.

2.5.3.2. Constraint

Um *Constraint* é a condição semântica já definida em 2.5.1.3 e aqui melhor detalhada.

Associações	
body	Uma descrição textual da restrição.
Associações	
condition	O predicado que se deve verificar para que o modelo definido esteja bem formado. Este predicado pode apenas existir caso o elemento afectado pela restrição seja uma instância ou um classificador (caso em que a restrição é aplicada a todas as instâncias do classificador). Este predicado é passível de ser utilizado em validações automáticas do modelo. O meta-modelo da linguagem não define uma linguagem para definição de condições de nível M2 como a OCL.
constrainedElement	O conjunto de elementos afectados pelo <i>Constraint</i> . Não podem ser elementos dos tipos <i>Stereotype</i> , <i>Constraint</i> ou <i>TaggedValue</i> . Caso a restrição seja aplicada a um <i>TagDefinition</i> , então ela corresponde a uma restrição que deve ser verificada por todos os <i>TagValue's</i> existentes por ele definidos.
constrainedStereotype	O <i>Stereotype</i> ao qual pertence o <i>Constraint</i> . Caso esta associação exista numa dada instância, então a <i>constrainedElement</i> deve ser vazia e a condição é imposta a todos os elementos marcados pelo <i>Stereotype</i> .

2.5.3.3. TagDefinition

Um *TagDefinition* especifica o tipo de *TaggedValue's* que podem ser atribuídos a elementos de modelação. Um *TagDefinition* é sempre parte integrante de um estereótipo.

Atributos	
multiplicity	O número de <i>TaggedValue's</i> definidos pelo <i>TagDefinition</i> que deverão ocorrer num dado elemento de modelação marcado pelo <i>owner</i> .
tagType	O <i>DataType</i> que estabelece os valores possíveis dos <i>TaggedValue's</i> classificados pelo <i>TagDefinition</i> .
Associações	
typedValue	O conjunto de todos os <i>TaggedValue's</i> que são instâncias do <i>TagDefinition</i> .
owner	O estereótipo ao qual pertence o <i>TagDefinition</i> .

2.5.3.4. TaggedValue

Um *TaggedValue*, etiqueta, é um atributo semântico associado a um elemento de modelação marcado pelo *Stereotype* que o define. É uma instância de um *TagDefinition*.

Atributos	
<i>name</i>	(Herdado de <i>ModelElement</i>) Ao name de um <i>TaggedValue</i> é automaticamente atribuído o name do <i>TagDefinition</i> que o define.
<i>dataValue</i>	O valor do <i>TaggedValue</i> do tipo especificado em <i>type</i> .
Associações	
<i>type</i>	A definição do <i>TaggedValue</i> .

2.6. Gestão de Modelos

O pacote *Model Management* define elementos que permitem gerir e organizar elementos de modelação. A linguagem estabelece elementos básicos para organização de elementos de modelação, identificação de versões e interligação entre modelos.

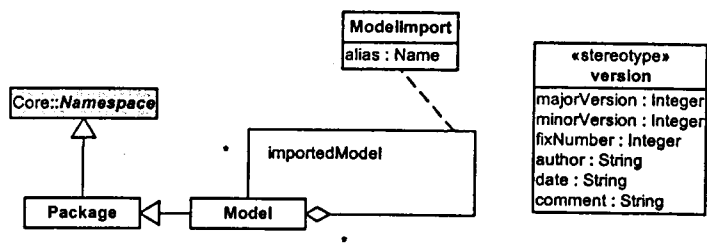


Figura 28. Gestão de modelos - *Model Management*.

2.6.1.1. Package

Um *Package* (pacote) é um agrupamento de quaisquer elementos de modelação. Esta meta-classe que realiza *Namespace* permite organizar elementos de modelação numa estrutura hierárquica. A visibilidade de elementos de um dado *Namespace* é imediata em qualquer elemento independentemente das suas posições relativas na hierarquia, isto é, a utilização de elementos de diferentes *Namespace*, não condicionada pela utilização de elementos de modelação próprios, mas apenas pela visibilidade dos elementos no seu *Namespace*.

2.6.1.2. Model

Um *Model*, modelo, é um pacote de topo na hierarquia de pacotes que define todo um modelo. A utilização de um *Model* é semelhante a uma biblioteca de elementos de modelação que pode ser reutilizada em vários projectos. A qualquer projecto corresponde a construção de um *Model* que poderá recorrer a outros modelos previamente elaborados.

Estereótipos	
«version»	Informação de versão do modelo. Para um modelo a existência de informação de versão é obrigatória.
Associações	
importedModel	O conjunto dos modelos importados e utilizados no modelo.

2.6.1.3. Version

A identificação de versões é conseguida através do estereótipo *Version*, que pode ser aplicado a qualquer elemento de modelação. Uma versão é identificada por um número com três componentes na forma “majorVersion.minorVersion.fixNumber”. Note-se que o meta-modelo define apenas regras para identificação de versões, a gestão de versões ou a identificação de compatibilidade entre modelos não é âmbito desta especificação.

Atributos (Definição de Etiquetas)	
majorVersion	Número da versão do elemento. Diferentes números de versão correspondem a versões do elemento com características significativamente distintas.
minorVersion	Número secundário da versão do elemento. O número secundário de versão permite distinguir pequenas alterações a uma dada versão do elemento.
fixNumber	Número de correcção da versão secundária. Permite identificar correcções a erros encontrados no elemento.
author	Nome do autor da revisão que originou a versão.
date	Data da revisão que originou a versão.
comment	Comentário da versão.

2.6.1.4. ModelImport

Um *ModelImport* define o nome alternativo do modelo importado no contexto do modelo importador. Apenas os elementos com visibilidade pública no modelo importado são visíveis no elemento importador.

Atributos	
alias	O nome alternativo local do modelo importado (pode ser omitido, caso em que o nome original é utilizado). Este meta-atributo permite ultrapassar conflitos de nomes.

2.7. Elementos de Comportamento

O pacote *Behavioural Elements* compreende os elementos associados à modelação comportamental, mas também os elementos de modelação estrutural estática para instâncias. O pacote *Behavioural Elements* é decomposto no pacote *Petri Nets*, que corresponde à definição de elementos de modelação associados a redes de Petri, no pacote *Methods*, que inclui a definição de uma linguagem procedimental para modelação de acções e condições, e pelos meta-elementos genéricos do pacote *Common Behaviour*.

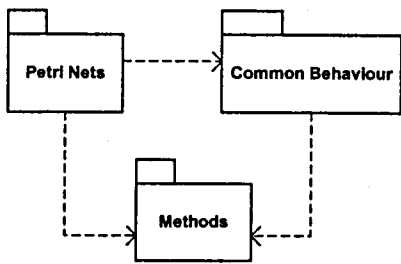


Figura 29. Elementos de Comportamento – *Behavioural Elements*.

2.7.1. Pacote Common Behaviour

A modelação de comportamento discreto implica a modelação de estruturas de definição de controlo de fluxo e a modelação de interações entre essas estruturas, o tempo e restantes elementos do modelo. Estas interações são modeladas com recurso a eventos e acções. Os eventos especificam acontecimentos com potencial para influenciarem a evolução do estado de instâncias e as acções, no sentido inverso, modelam modificações ao estado geradas pelo modelo de comportamento. Neste pacote incluem-se ainda os elementos de modelação associados à representação de instâncias de classificadores e associações e um elemento de modelação de comportamento assíncrono elementar.

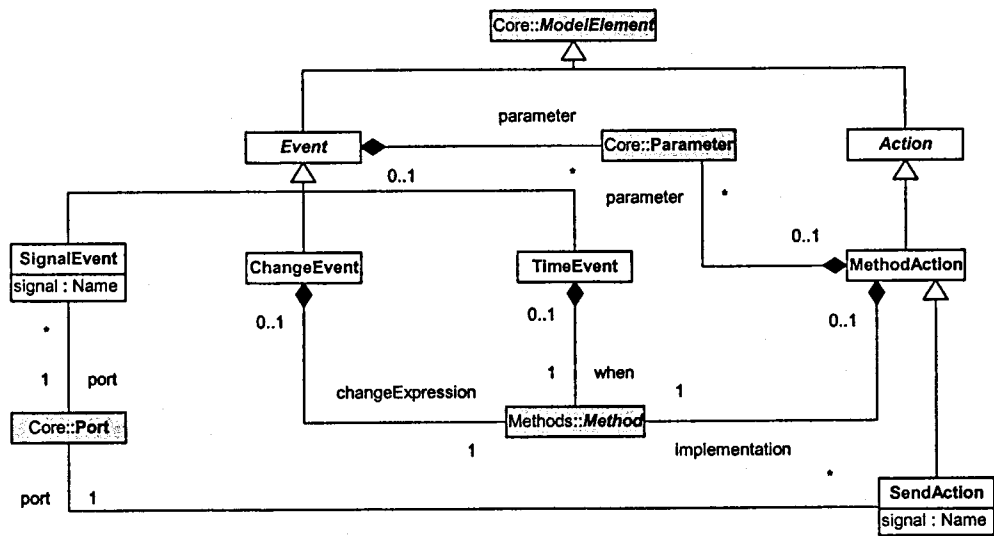


Figura 30. Acções e eventos do pacote *Common Behaviour*.

2.7.1.1. Action

Um *Action* generaliza acções executáveis sobre o estado de instâncias do modelo Uma acção é parte integrante de uma classe e a sua visibilidade é sempre protegida ou privada.

2.7.1.2. MethodAction

Um *MethodAction* corresponde a uma acção definida por um método.

Associações	
<i>implementation</i>	O método que implementa a acção.
<i>parameter</i>	Uma lista ordenada de parâmetros da acção.

2.7.1.3. SendAction

Um *SendAction* é uma acção cujo efeito é enviar o sinal associado a um dado porto. A lista de parâmetros de uma acção deste tipo deverá incluir pelo menos o conjunto de parâmetros do sinal a enviar. Estes parâmetros, que no contexto da acção correspondem a parâmetros de saída, determinam os valores dos parâmetros do sinal a enviar.

Atributos	
<i>signal</i>	O nome de um dos parâmetros associados ao porto que deverá ser obrigatoriamente do tipo <i>out</i> e que corresponde ao sinal a enviar.
Associações	
<i>port</i>	Uma referência para um porto da classe à qual se associa a acção.

2.7.1.4. Event

A meta-classe abstracta *Event* generaliza as meta-classes concretas de eventos. Um *Event* corresponde a um acontecimento instantâneo que pode ser utilizado na modelação de comportamento discreto. Um evento é parte integrante de uma classe e a sua visibilidade é sempre protegida ou privada.

Associações	
<i>parameter</i>	Uma lista ordenada de parâmetros do evento. Os parâmetros de um evento são sempre do tipo <i>out</i> e determinados pelo evento concreto.

2.7.1.5. SignalEvent

Um *SignalEvent* é uma classe de eventos cujas instâncias são geradas no instante da geração de um dado sinal. Os parâmetros deste evento correspondem aos parâmetros do sinal ao qual está associado.

Atributos	
<i>signal</i>	O nome de parâmetro, de um dos sinais associados a <i>port</i> , que estabelece o sinal que dispara o evento. Apenas parâmetros de <i>port</i> com o tipo <i>in</i> são admitidos.
Associações	
<i>port</i>	A referência para o porto onde os sinais que disparam o evento são gerados.

2.7.1.6. TimeEvent

Um *TimeEvent* é uma classe de eventos temporizada, isto é, as instâncias de eventos são geradas após um período de tempo determinado. Este evento não tem parâmetros.

Associações	
when	Um método de leitura que devolve um intervalo de tempo após o qual são geradas instâncias do evento. O tipo do parâmetro de retorno, único na interface do método, é do tipo <i>TimePeriod</i> .

2.7.1.7. ChangeEvent

Um *ChangeEvent* é um evento que ocorre quando o valor de uma dada expressão booleana se torna verdadeira por alteração do valor de algum dos seus componentes, ou seja, a cada transição de valor de falso para verdadeiro. Este evento não tem parâmetros.

Associações	
changeExpression	O método de leitura que especifica o evento, com um único parâmetro, de retorno, do tipo <i>Bool</i> .

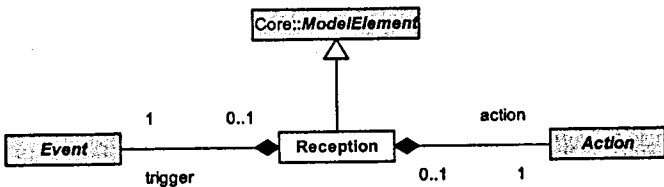


Figura 31. Elemento de modelação de comportamento assíncrono elementar.

2.7.1.8. Reception

Um *Reception* corresponde a um elemento de modelação comportamental elementar cujo significado semântico é a execução de uma acção na ocorrência de um dado evento. Note-se que um *Reception* define o comportamento de uma classe, apresentando-se como uma característica protegida, herdada e substituível. Um *Reception* de um ascendente é também parte integrante do descendente e pode ser redeclarado, isto é, pode ocorrer sobreposição de *Reception*'s. Uma classe que inclui um ou mais *Reception*'s é necessariamente activa. Na realidade um *Reception* é um caso particular de uma rede de Petri com um só elemento do tipo transição. No entanto, dado que muitos comportamentos simples podem ser descritos com recurso a um elemento deste tipo, este elemento, apesar de redundante, é incluído no meta-modelo.

Associações	
<i>action</i>	A acção que modela o comportamento na ocorrência do evento.
<i>trigger</i>	Evento de disparo da acção. Os eventos da classe <i>TimeEvent</i> , quando associados a <i>Reception</i> 's, correspondem a uma execução cíclica da acção com um período entre execuções determinado pelo evento.

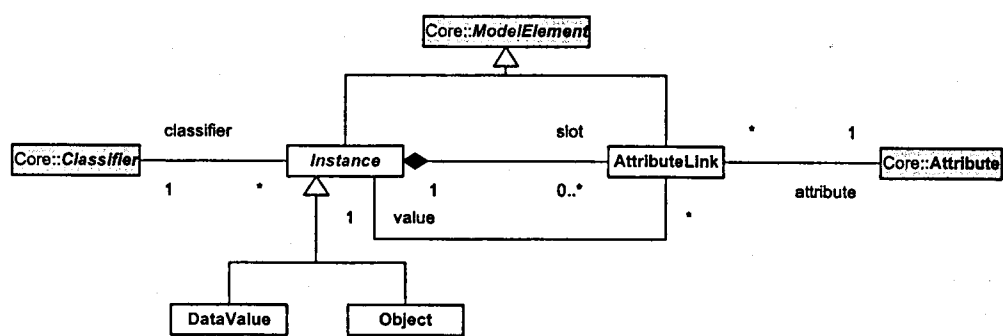


Figura 32. Instâncias do pacote *Common Behaviour*.

2.7.1.9. Instance

Um *Instance* (instância) é uma entidade que define instâncias de um *Classifier*.

Associações	
<i>classifier</i>	O <i>Classifier</i> que define a estrutura e o comportamento da instância. É sempre um classificador concreto.
<i>slot</i>	O conjunto de <i>AttributeLink</i> 's que contêm os valores dos atributos definidos em <i>classifier</i> .
<i>linkEnd</i>	Os extremos de ligação ligados à instância.

2.7.1.10. Object

Uma instância de uma classe é modelada por um *Object*, objecto. Entre as instâncias, apenas os objectos podem ser directamente incluídos num pacote. Um objecto incluído directamente num pacote não pode ser dinamicamente instanciado num dado modelo M0, o que implica que todos os objectos deste tipo são definidos em tempo de modelação. O conjunto de todos os objectos de um dado modelo e as ligações entre estes definem concretamente um dado sistema.

2.7.1.11. DataValue

Uma instância sem identidade, usada para instâncias de *Datatype's*.

2.7.1.12. AttributeLink

Um *AttributeLink* é um contentor para instâncias de atributos de uma dada classe.

Associações	
<i>value</i>	A instância correspondente ao <i>AttributeLink</i> .
<i>attribute</i>	O atributo que define o <i>AttributeLink</i> .

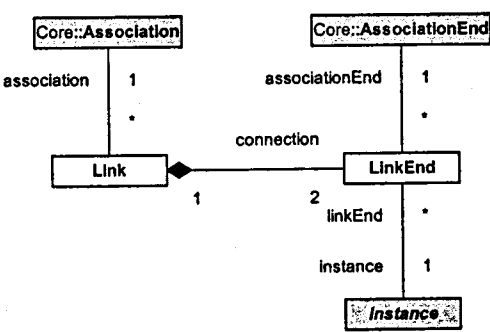


Figura 33. Ligações do pacote *Common Behaviour*.

2.7.1.13. Link

Um *Link* (ligação) é uma entidade que define instâncias de associações. Uma ligação não é dinâmica pelo que não pode ser alterada após a criação da instância num dado modelo M0, o que implica que todas as ligações são estabelecidas em tempo de modelação. Uma associação com extremos cuja multiplicidade possa ser superior a um é instanciada em múltiplas ligações.

Associações	
<i>association</i>	A associação que declara a ligação.
<i>connection</i>	Os dois extremos da ligação.

2.7.1.14. LinkEnd

Um *LinkEnd* (extremo de ligação) é uma entidade que define um ponto final de uma ligação.

Associações	
<i>associationEnd</i>	O extremo de associação que declara o ponto de ligação.
<i>instance</i>	A instância ao qual o ponto de ligação está ligado. O classificador da instância é condicionado pelo classificador ao qual está associado <i>associationEnd</i> . O primeiro deverá ser sempre o mesmo que o segundo, ou então, um seu descendente.

2.7.2. Pacote Petri Nets

Este pacote compreende as redes de Petri, o modelo principal para definição comportamental.

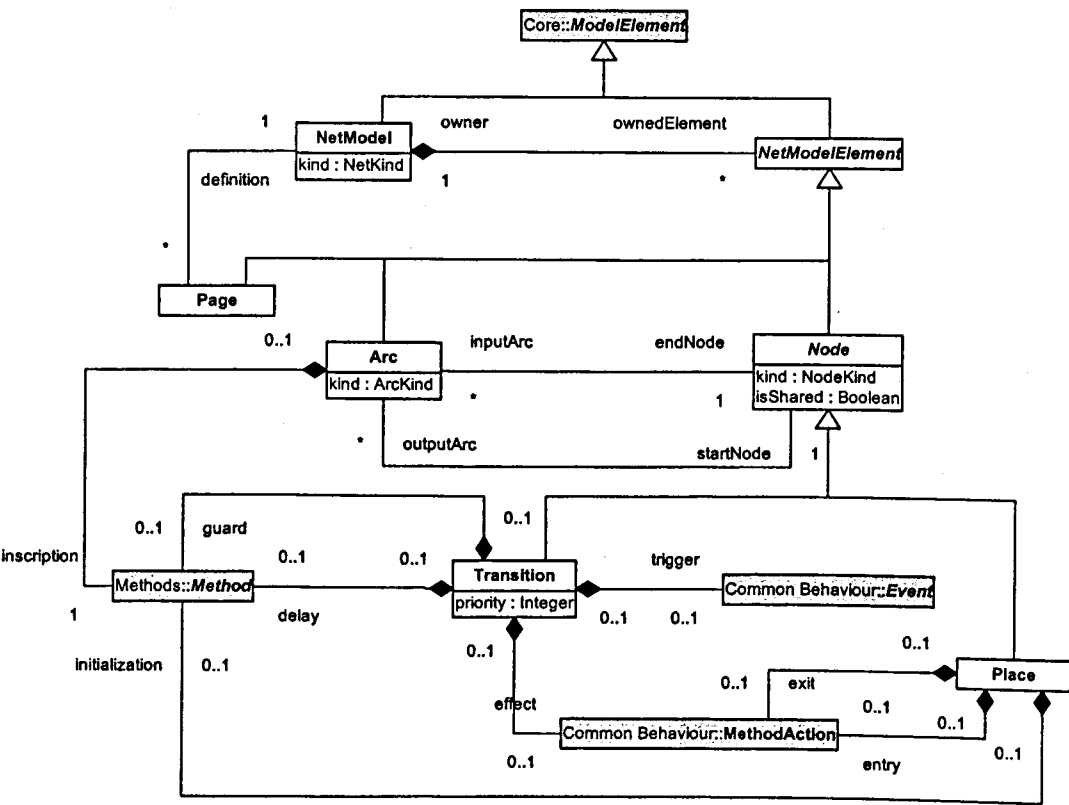


Figura 34. Modelo do pacote *Petri Nets*.

2.7.2.1. NetModel

A *NetModel* é uma meta-classe concreta cujas instâncias são definições de redes de Petri, ou seja, modelos de rede. Um *NetModel* está associado ao elemento de modelação cujo comportamento define, restrito a classes, e apresenta um de dois tipos: rede de objecto e subrede.

Um *NetModel* do tipo *subnet* (subrede) corresponde a uma rede que é utilizada na definição de outras redes de qualquer que seja o seu tipo. Uma subrede proporciona ao meta-modelo características de estruturação. A instanciação de um *NetModel* deste tipo ocorre na instanciação de cada página por ele definida.

Um *NetModel* do tipo *object* (rede de objecto) corresponde a uma rede que estabelece o comportamento de instâncias da classe à qual está associada, conferindo-lhe a qualificação de activa. A instanciação deste tipo de redes ocorre em cada instanciação do classificador cujo comportamento especifica.

Note-se que um *NetModel* define o comportamento de uma classe, um elemento generalizável, e proporciona características de herança de comportamento. A refinação de redes de Petri ocorre numa perspectiva de herança estrita por substituição de *NetModel*s com nome idênticos do elemento especializado no elemento generalizado.

Atributos	
<i>kind</i>	Especifica o tipo de rede: <i>subnet</i> – subrede. <i>object</i> – rede de objecto (valor por defeito).
Associações	
<i>ownedElement</i>	Os elementos constituintes da rede e aqueles que definem a sua estrutura.

2.7.2.2. NetModelElement

A *NetModelElement* é uma meta-classe abstracta que generaliza todos os elementos de modelação que são constituintes estruturais de um modelo de rede de Petri.

Associações	
<i>owner</i>	O modelo de rede no qual se inclui o elemento.

2.7.2.3. Node

Node é uma meta-classe abstracta que generaliza os nós de um modelo de rede de Petri, nomeadamente os lugares e as transições.

Atributos	
<i>name</i>	(Herdado de <i>ModelElement</i>) O nome de um nó deve ser único no contexto do modelo da rede de Petri.
<i>isShared</i>	Caso este meta-atributo apresente o valor verdadeiro o nó é partilhado por todas as instâncias do modelo de rede que o inclui.
<i>kind</i>	Especifica o tipo de nó na rede de Petri: <i>in</i> – nó de entrada. <i>out</i> – nó de saída. <i>general</i> – nó genérico (valor por defeito). Os nós de entrada e de saída correspondem a pontos de acesso a instâncias do modelo de rede desde redes de Petri que as incluam. Um nó de entrada não pode conter arcos incidentes e um nó de saída apenas pode conter arcos incidentes. Nós destes dois tipos correspondem a pontos de comunicação entre páginas e nós do modelo de rede, sendo, entres estes partilhados de forma síncrona. Os nós genéricos são nós constituintes da estrutura da rede e podem estar associados a outros nós por qualquer tipo de arcos.
Associações	
<i>outputArc</i>	O conjunto de arcos provenientes do nó. Os nós sobre os quais incidem os <i>outputArc</i> denominam-se nós de saída do nó.
<i>inputArc</i>	O conjunto de arcos incidentes no nó. Os nós sobre os quais incidem os <i>inputArc</i> denominam-se nós de entrada do nó.

2.7.2.4. Arc

Um *Arc* (arco) é um elemento de modelação que estabelece ligações orientadas entre nós.

Atributos	
<i>name</i>	(Herdado de <i>ModelElement</i>) O nome de um arco deve ser único no contexto da rede de Petri mas pode ser vazio.
<i>kind</i>	O tipo de arco: <i>regular</i> – Um arco de uma rede ordinária (valor por defeito). <i>inhibitor</i> – Um arco inibidor, aplicável apenas a arcos incidentes em transições.

Associações	
<i>inscription</i>	A inscrição de um arco é uma expressão que determina o número de marcas a extrair de um lugar de origem e também o necessário para sensibilizar a transição em que incide ou o número de marcas a acrescentar ao lugar em que incide dado o disparo de transição de origem. Este valor é calculado em tempo de execução. A inscrição é definida num método de leitura com um único parâmetro de retorno de um qualquer tipo primitivo inteiro. O valor devolvido por este método será sempre um inteiro positivo. No caso de arcos inibidores a inscrição corresponde a um número de marcas que deverá ser superior ao número de marcas do lugar de origem para que a transição esteja sensibilizada.
<i>startNode</i>	O nó de onde é proveniente o arco, que deverá ser de tipo distinto de <i>endNode</i> e incluído no mesmo modelo de rede que o arco.
<i>endNode</i>	O nó onde incide o arco, que deverá ser de tipo distinto de <i>startNode</i> e incluído no mesmo modelo de rede que o arco.

2.7.2.5. Page

Um *Page* (página) corresponde à inclusão de um modelo de rede externo conferindo propriedades hierárquicas ao meta-modelo de redes de Petri. Os nós do tipo *in* ou *out* do modelo de rede associado à página são considerados também nós do próprio modelo de rede e apresentam a restrição inversa para ligações de arcos daquela apresentada em 2.7.2.3.

Associações	
<i>definition</i>	Um modelo de rede do tipo <i>subnet</i> associado à classe (ou seus atributos) onde o modelo de rede que inclui a página se insere. Este modelo de rede é usado para especificar e instanciar a página.

2.7.2.6. Place

Um *Place* corresponde a um lugar, ou posição, de uma rede de Petri.

Associações	
<i>initialization</i>	Um método de leitura que devolve um inteiro positivo não negativo cujo valor determina o número de marcas depositadas no lugar no instante que precede o início da execução da rede, calculado em tempo de execução.
<i>entry</i>	Ação executada após o acréscimo de marcas no lugar, sem quaisquer parâmetros.
<i>exit</i>	Ação executada antes da remoção de marcas do lugar, sem quaisquer parâmetros.

2.7.2.7. Transition

Um *Transition* corresponde a uma transição de uma rede de Petri. Uma transição é disparada sempre que se verifique em simultâneo a sua sensibilização e, caso exista, a sua a condição. O disparo ocorre no instante de ocorrência do evento de disparo, caso este exista, ou, caso contrário, no instante de verificação das condições de disparo. O disparo da transição implica: a remoção de marcas dos lugares de origem, a execução, da acção, caso esta esteja definida, e a deposição de marcas nos lugares de saída.

Atributos	
<i>priority</i>	Estabelece a prioridade da transição. Em caso de conflito de disparo entre duas ou mais transições a transição disparada é aquela com prioridade mais elevada. Caso a prioridade das transições em conflito seja igual a escolha da transição a disparar não é determinística. A prioridade corresponde um inteiro não negativo, decrescendo a prioridade com o crescimento do seu valor.
Associações	
<i>trigger</i>	Evento de disparo da transição. Caso o evento não seja definido a transição, caso esteja sensibilizada, é avaliada para disparo continuamente. Os eventos da classe <i>TimeEvent</i>, quando associados a transições de redes de Petri, correspondem a metas temporais relativas ao instante de deposição de marcas nos seus lugares de origem. A meta temporal é calculada no instante de execução de qualquer passo da rede em que as transições nele disparadas depositem marcas nos lugares de origem da transição. Caso a transição esteja, nesse instante inicial, sensibilizada e o predicado <i>guard</i> seja verdadeiro uma instância do evento será gerada no final da meta temporal.
<i>guard</i>	Condição especificada por um método de leitura com valor de retorno booleano e avaliado sempre que ocorre o evento de disparo e a transição está sensibilizada. O disparo da transição ocorre apenas se o predicado for verdadeiro. Caso o evento de disparo da transição exista então a lista de parâmetros do método inclui todos os parâmetros do evento. Em todos os restantes casos a lista de parâmetros inclui apenas o valor de retorno. A condição é avaliada em tempo de execução.
<i>delay</i>	Estipula o tempo de espera entre a remoção das marcas dos lugares de entrada e a deposição das marcas nos lugares de saída no disparo da transição. Corresponde a um método de leitura cujo valor de retorno é do tipo <i>TimePeriod</i> . O período de espera é calculado em tempo de execução a cada disparo da transição.
<i>effect</i>	Acção executada sempre que é disparada a transição. Os parâmetros da acção, caso esta seja do tipo <i>MethodAction</i>, são pré-determinados pelo tipo do evento associado à transição. Caso o evento de disparo exista então a lista de parâmetros da acção é idêntica aos parâmetros do evento.

2.7.3. Pacote Methods

O pacote *Methods* estabelece uma linguagem procedimental para definição de métodos utilizados para modelar a implementação de alguns elementos de modelação comportamental ou o cálculo de determinadas propriedades. O modelo do pacote *Methods* é apresentado nos diagramas das figuras 35 a 39.

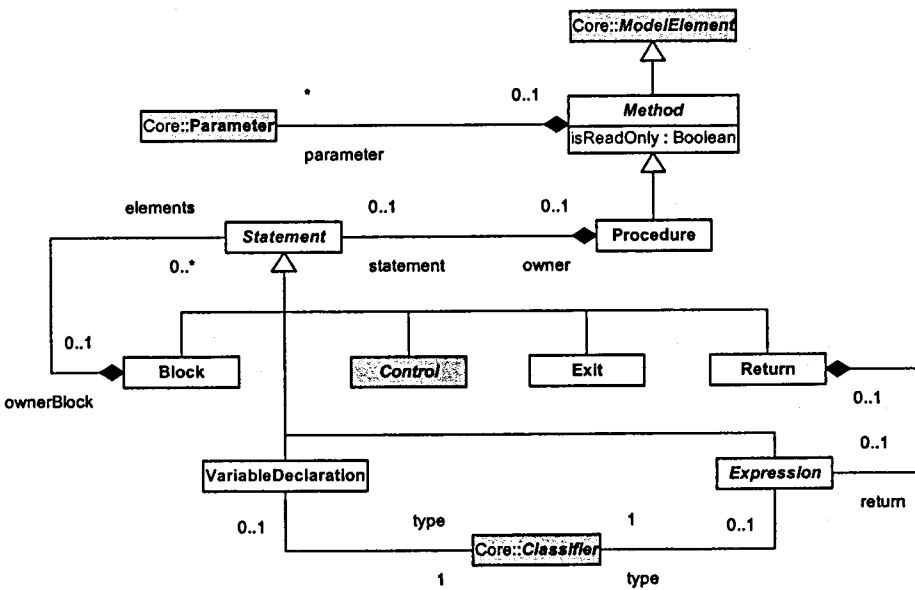


Figura 35. Modelo base do pacote *Methods*.

2.7.3.1. Method

Um *Method* (método) modela a implementação de elementos como operações ou acções. Um método existe apenas no meta-modelo da linguagem como elemento integrante da definição de um outro elemento de modelação.

Atributos	
<i>name</i>	(Herdado de <i>ModelElement</i>) O nome de um método é, em regra, sempre vazio e não têm significado semântico.
<i>isReadOnly</i>	Caso este meta-atributo apresente o valor verdadeiro, então a método é um método de leitura. Um método de leitura não pode realizar alterações ao estado de instâncias.
Associações	
<i>parameter</i>	Uma lista ordenada de parâmetros que corresponde à interface do método. Todos os parâmetros de um método são passados ou devolvidos por valor e o seu tipo deverá corresponder a uma estrutura. Apenas poderá existir um parâmetro de tipo <i>return</i> na lista de parâmetros.

2.7.3.2. Procedure

Um *Procedure* (procedimento) é um método que estabelece uma sequência de instruções invocável que se considera serem executadas instantaneamente.

Associações	
<i>statement</i>	A instrução de processamento a executar a cada invocação do procedimento.

2.7.3.3. Statement

Um *Statement* corresponde a uma instrução. Na definição de algumas meta-classes concretas de *Statement* são incluídos meta-atributos do tipo *Reference* que correspondem a referências para elementos de modelação. Os elementos que podem ser por referenciados por um dado *Reference* são determinados pelo elemento de modelação que integra a instrução. A tabela 10 mostra que elementos podem ser referenciados.

Atributos	
<i>name</i>	(Herdado de <i>ModelElement</i>) O nome de um <i>Statement</i> é, em regra, sempre vazio e não têm significado semântico.
Associações	
<i>owner</i>	O procedimento ao qual a instrução está associada, caso exista.
<i>ownerBlock</i>	O bloco no qual se inclui a instrução de processamento, caso exista.

2.7.3.4. Block

Um *Block* corresponde a uma sequência de instruções a serem executadas numa dada ordem.

Associações	
<i>element</i>	A lista ordenada de instruções de processamento.

2.7.3.5. Exit

Uma instrução que permite terminar a execução de um ciclo (*While*, *Repeat*, *For*) quando incluída numa das suas instruções constituintes. O ciclo terminado por um *Exit* é o primeiro ciclo encontrado ao navegar a hierarquia de instruções no sentido contrário ao da execução.

Tabela 10. Elementos referenciáveis em instruções.

Elemento ao qual se associa a instrução	Descrição	
Procedure	Qualquer instrução incluída num procedimento pode referir: (i) qualquer parâmetro da interface do procedimento, (ii) qualquer variável local declarada e (iii) os parâmetros de saída de operações invocadas no procedimento. Operações cujo atributo <i>isQuery</i> apresenta o valor falso não são referenciáveis em procedimentos de leitura.	
Rede de Petri, atributos, eventos, acções e elementos de interacção	Os elementos visíveis a partir do classificador que contém o elemento de modelação.	Inclui todos os atributos, operações e portos do classificador bem como os elementos aos quais este está associado por relações de associação navegáveis e visíveis.
Constraint	Os elementos visíveis a partir do classificador afectado pela restrição.	
Operation	Caso a operação seja de instância podem ser referidos os elementos visíveis a partir do classificador que contém a operação. Caso não seja uma operação de instância apenas os <i>Feature</i> de classe podem ser referenciados	

2.7.3.6. VariableDeclaration

A *VariableDeclaration* representa uma declaração de variável local a dado procedimento. O contexto de uma variável declarada num dado bloco estende-se a todos os blocos nele contidos

Atributos	
name	⟨Herdado de <i>ModelElement</i> ⟩ O nome de um <i>VariableDeclaration</i> corresponde ao nome da variável local por ele declarada.
Associações	
type	O <i>Classifier</i> que designa o tipo da variável. Apenas estruturas podem ser tipos de variáveis.

2.7.3.7. Return

Um *Return* é uma instrução que delimita a execução do procedimento e representa o fim de execução da sua sequência de instruções.

Associações	
<i>return</i>	Uma expressão que determina o valor de retorno do método, caso este apresente um parâmetro desse tipo.

2.7.3.8. Assignment

Um *Assignment* é uma instrução cujo efeito é atribuir um dado valor a uma instância. Caso o procedimento seja de leitura as instruções do tipo *Assignment* não podem estar presentes na sua definição, excepto nos casos em que *target* é uma variável ou um parâmetro de saída do procedimento.

Atributos	
<i>target</i>	Uma referência para uma instância à qual o valor se atribui. Caso <i>target</i> seja um atributo, este não deve ser constante.
Associações	
<i>value</i>	Uma expressão de tipo idêntico àquele do elemento definido por <i>target</i> . Ao resultado da avaliação desta expressão corresponde o valor a atribuir a <i>target</i> .

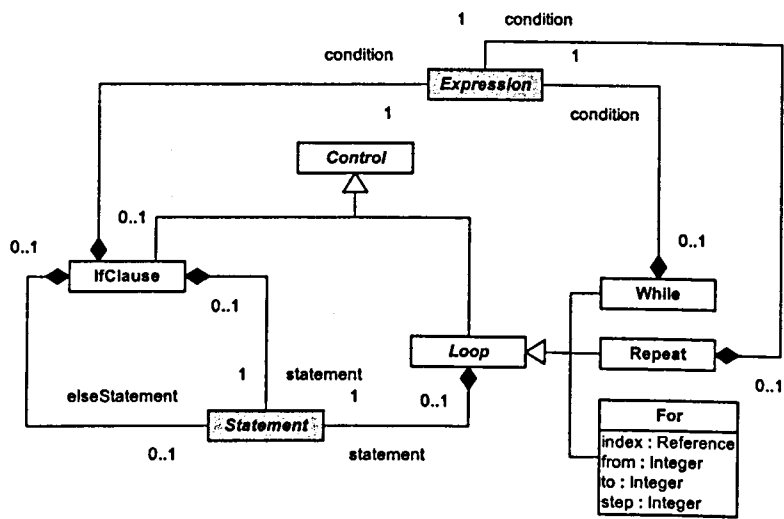


Figura 36. Instruções de controlo do pacote *Methods*.

2.7.3.9. Expression

A *Expression* é uma meta-classe que generaliza a implementação das expressões, ou seja instruções com um valor associado.

Associações	
<i>type</i>	O <i>Classifier</i> que designa o tipo da expressão. Apenas estruturas podem ser tipos de expressões.

2.7.3.10. Control

A *Control* é uma meta-classe abstracta que generaliza todas as instruções de controlo. Qualquer instrução de controlo compreende a execução condicionada de uma ou mais instruções.

2.7.3.11. IfClause

Uma instrução de controlo condicional. As instruções a executar caso a avaliação da condição devolva o valor verdadeiro.

Associações	
<i>condition</i>	Uma expressão do tipo <i>Bool</i> que define a implementação da condição.
<i>statement</i>	O <i>Statement</i> condicionado.
<i>elseStatement</i>	O <i>Statement</i> a executar caso a avaliação da condição devolva o valor falso.

2.7.3.12. Loop

Um *Loop* (ciclo) corresponde a uma generalização das instruções de controlo que permitem a execução múltipla de uma ou mais instruções.

Associações	
<i>statement</i>	O <i>Statement</i> condicionado que implementa as instruções a executar em cada ciclo.

2.7.3.13. While

Um ciclo cuja instrução é executada repetidamente sempre que a avaliação da condição devolve o valor verdadeiro. A avaliação da condição é executada antes de cada execução da instrução e, caso devolva o valor falso, o ciclo é terminado.

Associações	
<i>condition</i>	Uma expressão do tipo <i>Bool</i> que define a condição de terminação da execução do ciclo.

2.7.3.14. Repeat

Um ciclo cuja instrução é executada repetidamente sempre que a avaliação da condição devolve o valor verdadeiro. A avaliação da condição é executada após cada execução da instrução, pelo que esta é executada pelo menos uma vez. Se a avaliação da condição devolver o valor falso, a instrução cíclica é terminada.

Associações	
<i>condition</i>	Uma expressão do tipo <i>Bool</i> que define a condição de terminação da execução do ciclo.

2.7.3.15. For

Um ciclo cuja instrução é executada um número determinado de vezes.

Atributos	
<i>index</i>	A referência de uma variável local de tipo inteiro que é utilizada como índice de iteração do ciclo.
<i>from</i>	O valor inicial de <i>index</i> .
<i>to</i>	O valor máximo de <i>index</i> . Caso <i>index</i> tome um valor superior a <i>to</i> , antes de cada ciclo, a instrução é terminada. Deve ser igual ou superior a <i>from</i> .
<i>step</i>	O valor do incremento a aplicar a <i>index</i> após a execução de cada ciclo. Deve ser um inteiro positivo.

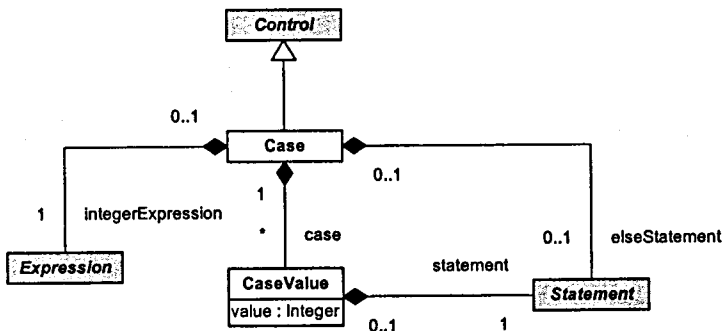


Figura 37. Instrução de controlo Case.

2.7.3.16. Case

Uma instrução de controlo que permite a execução de uma ou várias instruções, determinada pelo resultado da avaliação de um expressão de tipo inteiro.

Associações	
<i>integerExpression</i>	Uma expressão de tipo inteiro que estabelece o valor a comparar para determinar a instrução que deverá ser executada.
<i>case</i>	O conjunto de tuplos valor/instrução cujo valor é comparado com o resultado da avaliação de <i>integerExpression</i> para determinar qual deverá ser executada.
<i>elseStatement</i>	O <i>Statement</i> a executar caso nenhum dos tuplos de <i>case</i> apresente o valor da avaliação de <i>integerExpression</i> .

2.7.3.17. CaseValue

Um tuplo valor/instrução utilizado na definição de uma instrução Case.

Atributos	
<i>value</i>	O valor inteiro para comparação.
Associações	
<i>statement</i>	O <i>Statement</i> a executar.

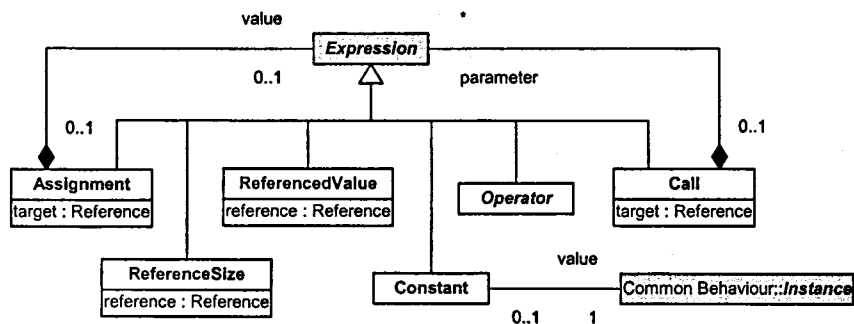


Figura 38. Expressões do pacote *Methods*.

2.7.3.18. Constant

Uma expressão cujo valor é constante em qualquer execução.

Atributos	
value	O valor constante com o tipo da expressão.

2.7.3.19. Call

Uma instrução de invocação de uma operação. O tipo de uma expressão *Call* é idêntico ao tipo do parâmetro de retorno da operação. Se a operação não apresenta parâmetro de retorno a invocação deve ser considerada um *Statement* e não uma expressão. Caso a invocação esteja contida num procedimento de leitura a invocação de operações está restrita àquelas cujo atributo *isQuery* apresente o valor verdadeiro.

Atributos	
target	A referência para uma operação.
Associações	
parameter	Uma lista ordenada de expressões que compreende uma expressão para cada parâmetro de entrada da operação a invocar. Os tipos de cada expressão correspondem àqueles da interface da operação.

2.7.3.20. ReferencedValue

Uma expressão cujo valor corresponde ao valor do elemento referido no instante da execução. O tipo desta expressão é idêntico àquele do elemento referido.

Atributos	
reference	Uma referência para um elemento do qual se obtém o valor. Neste caso particular a referência pode ser também um <i>EnumeratioLiteral</i> cujo valor constante se pretende obter.

2.7.3.21. ReferenceSize

Uma expressão cujo valor corresponde ao número de elementos referidos na instância de uma associação.

Atributos	
reference	Uma referência para uma associação.

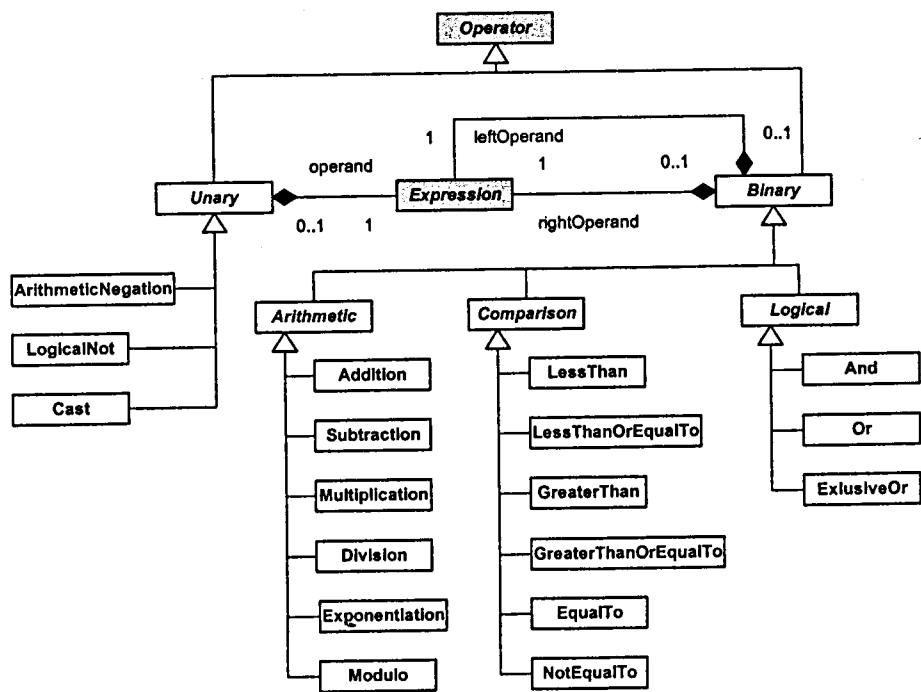


Figura 39. Operadores do pacote *Methods*.

2.7.3.22. Operator

Esta meta-classe abstracta generaliza todos os operadores. Um operador é uma expressão composta com características semânticas precisas.

2.7.3.23. Unary

Um operador com um só operando. A semântica para todos os operadores deste tipo, exceptuado o operador *Cast*, está descrita na tabela 11.

Associações	
<i>operand</i>	O operando do operador. O tipo do operando depende do operador concreto.

Tabela 11. Operadores unários.

Operador	Tipo	Tipos do Operando	Descrição
<i>ArithmeticNegation</i>	Idêntico ao do operando	Qualquer primitivo numérico com sinal	Inverte o sinal aritmético do operando.
<i>LogicalNot</i>		<i>Bool</i>	Negação lógica do operando.

2.7.3.24. Cast

Um *Cast* é operador unário que permite alterar o tipo do operando. Uma conversão de tipo permite converter valores entre tipos primitivos, entre estruturas com relações de herança e entre enumerados e o seu tipo associado. As conversões disponíveis entre tipos primitivos são apresentadas na tabela 12. A conversão entre um tipo enumerado e o tipo primitivo associado corresponde à cópia do valor, uma vez que, na realidade, representam um valor do mesmo tipo. A conversão entre estruturas com relações de herança apenas existe quando o tipo do operando é descendente do tipo para o qual se pretende converter.

Tabela 12. Operadores de conversão de tipos primitivos.

Tipo do Operando	Tipo da Expressão	Descrição
<i>Bool</i>	Numérico	Se o valor booleano for verdadeiro o tipo numérico terá o valor um e zero caso contrário.
Numérico	<i>Bool</i>	O valor booleano será verdadeiro caso o valor numérico seja um e falso caso contrário.
Inteiro com sinal	Inteiro com sinal de tamanho superior	Não ocorre alteração de valor.
	Inteiro com sinal de tamanho inferior	São truncados os <i>bits</i> mais significativos e mantido o sinal.
	Inteiro sem sinal.	Ocorre conversão de tamanho para inteiro com sinal de tamanho idêntico àquele do tipo da expressão. Após esta conversão não ocorre alteração de valor caso o inteiro original seja positivo. Caso o inteiro original seja negativo o <i>bit</i> de sinal é convertido em valor após conversão de tamanho.
Inteiro sem sinal	Inteiro sem sinal de tamanho superior	Não ocorre alteração de valor.
	Inteiro sem sinal de tamanho inferior	São truncados os octetos mais significativos.
	Inteiro com sinal.	Ocorre conversão de tamanho para inteiro com sinal de tamanho idêntico àquele do tipo da expressão. O <i>bit</i> mais significativo passa a ser considerado sinal.
Inteiro	Vírgula flutuante	Não ocorre alteração de valor a menos da eventual perda de precisão.
Vírgula flutuante	Inteiro	O valor é truncado no ponto decimal e convertido para inteiro. Caso a parte inteira seja superior ao máximo inteiro positivo representável ou inferior ao mínimo, o valor do inteiro será esse limite.
Vírgula flutuante	Vírgula flutuante	Não ocorre alteração de valor a menos da eventual perda de precisão. Caso o valor origem seja superior ao máximo representável ou inferior ao mínimo, o valor será esse limite.
Numérico	<i>TimePeriod</i> ~	O valor inteiro corresponde ao número de milissegundos representados pelo período.
<i>TimePeriod</i>	Numérico	
Qualquer tipo primitivo excepto <i>String</i>	<i>String</i>	Uma representação do valor em texto, dependente da plataforma.

2.7.3.25. Binary

Um operador com dois operandos. Os operadores deste tipo estão descritos na tabela 12.

Associações	
<i>leftOperand</i>	O operando esquerdo do operador. O tipo do operando depende do operador concreto.
<i>rightOperand</i>	O operando direito do operador. O tipo do operando depende do operador concreto.

Tabela 13. Operadores binários.

Operador	Tipo da Expressão	Tipo dos Operandos ³	Descrição
<i>Arithmetic</i>	Idêntico ao dos operandos	Qualquer primitivo numérico, <i>TimePeriod</i>	A operação aritmética respectiva: adição, subtracção, multiplicação, divisão, exponenciação ou resto da divisão inteira.
<i>Addition</i>	<i>String</i>	<i>String</i>	Concatenação.
<i>Subtraction</i>	<i>TimePeriod</i>	<i>Time</i>	Determinação do intervalo entre dois instantes temporais.
<i>EqualTo</i> , <i>NotEqualTo</i>	<i>Bool</i>	Qualquer primitivo	A operação de comparação respectiva: igualdade ou diferença.
<i>LessThan</i> , <i>LessThanOrEqualTo</i> , <i>GreaterThan</i> , <i>GreaterThanOrEqualTo</i>		Primitivos numéricos, <i>Time</i> , <i>TimePeriod</i>	A operação de comparação respectiva: menor, menor ou igual, maior e maior ou igual.
<i>Logical</i>	<i>Bool</i>		A operação de lógica respectiva: e, ou-inclusivo, ou-exclusivo.

³ Os operandos apresentam sempre o mesmo tipo.

2.8. Elementos de Interação

O modelo de elementos de interação é um modelo generalizável de nível M1 e uma instância da ASML (figura 40) com uma semântica própria. O facto destes elementos não integrarem o meta-modelo da linguagem apresenta vantagens interessantes nomeadamente na redução da dimensão e complexidade do meta-modelo. Outra vantagem é o facto permitir a integração directa da modelação estática e dinâmica na definição de elementos de interação.

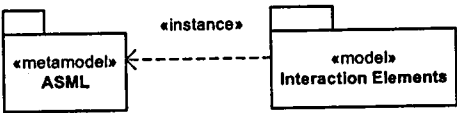


Figura 40. Modelo de elementos de interação da ASML.

O modelo de elementos de interação compreende os pacotes (i) *Foundation*, onde são definidos os elementos fundamentais, incluindo a modelação de acções do utilizador, (ii) *Geometric*, compreendendo a definição de formas geométricas, (iii) *Animation*, para modelação de animações de formas geométricas, e (iv) *Widgets*, que modelam elementos comuns de aplicações de interface ao utilizador. A figura 41 apresenta uma visão geral deste modelo.

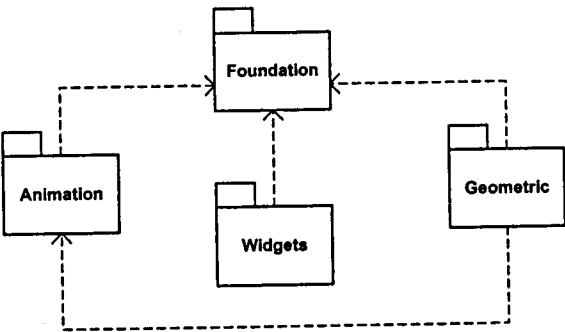


Figura 41. Modelo do pacote *Interaction Elements*.

2.8.1. Pacote Foundation

O modelo do pacote *Foundation* é apresentado nos diagramas ASML das figuras 42, 43 e 44.



Figura 42. Modelo do pacote *Foundation*.

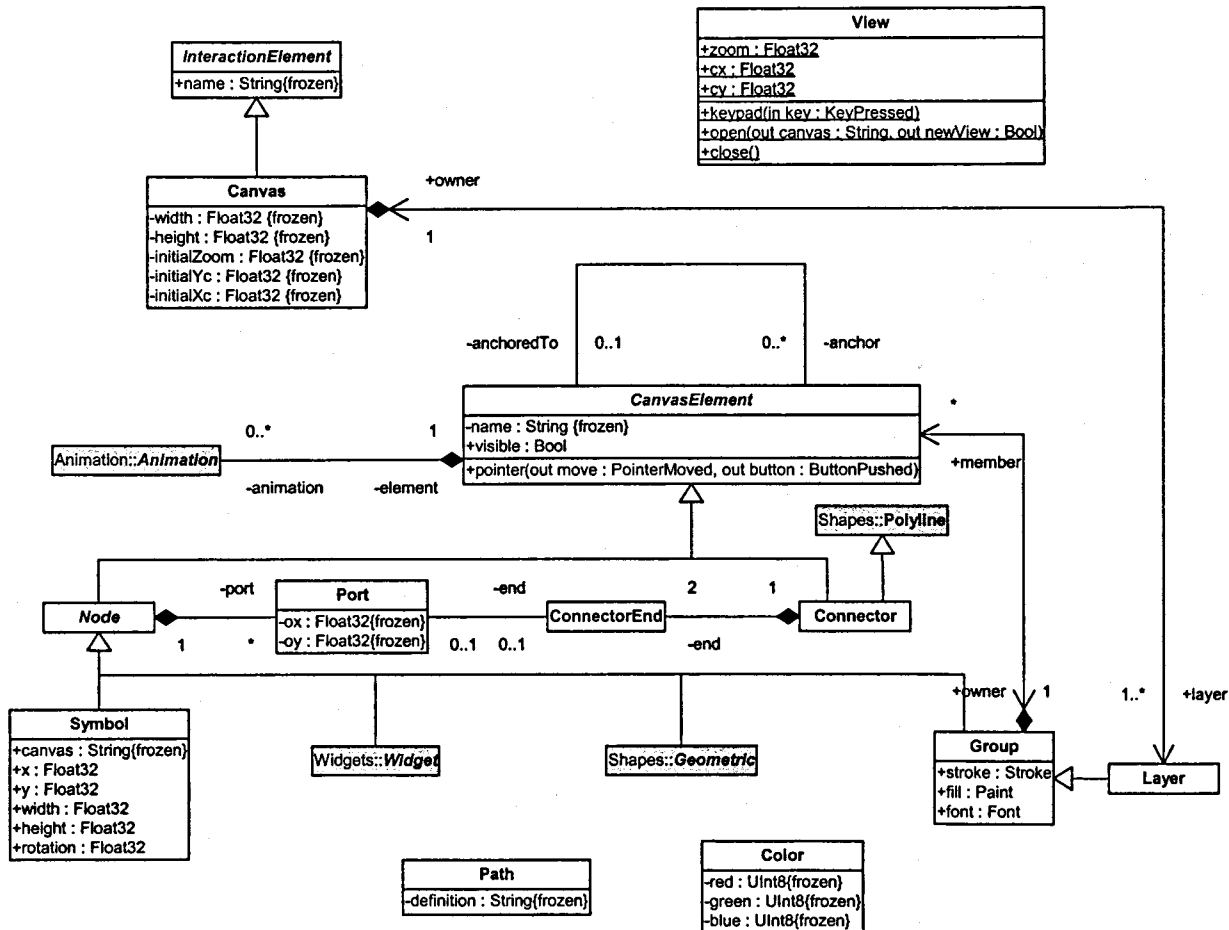


Figura 43. Modelo do pacote Core.

2.8.1.1. InteractionElement

A classe *InteractionElement* generaliza elementos de interacção. Um elemento de interacção pode estar associado a um classificador ou a um estereótipo, definindo, respectivamente, representações de instâncias do classificador ou simbologia alternativa para a notação da linguagem. No contexto de um classificador ou estereótipo, um elemento de interacção é uma característica herdada e que pode ser especializada por substituição de elementos de interacção com nomes idênticos no elemento generalizável especializado.

Atributos

<i>name</i>	Nome do elemento de interacção. Este nome deve apresentar o mesmo valor do meta-atributo <i>name</i> da meta-classe <i>ModelElement</i> da instância ou propriedade classificada como <i>InteractionElement</i> . Este nome pode ser utilizado em referências.
-------------	--

2.8.1.2. View

A *View* é uma classe com métodos, portos e atributos de classe que representa o estado da vista actual do *Canvas* em tempo de execução. Esta classe permite o acesso para leitura e modificação de propriedades da vista em animações. O modelo subjacente a uma vista inclui a possibilidade de ampliação (*zoom*), movimentação do *Canvas* (*pan*) e interacção através de um teclado e de um dispositivo apontador como um rato ou ecrã táctil.

Atributos	
<i>zoom</i>	Valor de ampliação da vista (modificável em animações ou por acção do utilizador).
<i>cx, cy</i>	Valor do ponto central de visualização da vista (modificável em animações ou por acção do utilizador).
Portos	
<i>keypad</i>	Este porto define um sinal de saída que é enviado sempre que uma tecla é premida. Este sinal pode ser utilizado em elementos comportamentais de qualquer <i>Canvas</i> .
Operações	
<i>open</i>	A invocação desta operação permite abrir uma nova vista ou substituir a vista actual com um novo <i>Canvas</i> .
<i>close</i>	Permite fechar a vista actual.

2.8.1.3. Canvas

Um *Canvas* é um elemento de interacção, visualizável numa ou mais vistas, que consiste num painel onde são desenhados elementos gráficos interactivos. A dimensão e localização de elementos gráficos pressupõe um sistema de unidades e de coordenadas. Neste modelo a unidade dimensional utilizada é o *pixel*, dependente do dispositivo de visualização. O sistema de coordenadas apresenta origem no canto superior esquerdo da vista, eixo de ordenadas crescente do topo para o fundo e eixo de abcissas crescente da esquerda para a direita.

Atributos	
<i>width, height</i>	Dimensão do <i>Canvas</i> .
<i>initialZoom</i>	Factor de ampliação de uma vista inicial do <i>Canvas</i> , em valor percentual.
<i>initialXc, initialYc</i>	Valores do ponto central de visualização do <i>Canvas</i> numa vista inicial.
Associações	
<i>layer</i>	Lista ordenada de camadas de objectos gráficos que compõem o <i>Canvas</i> . A ordem de desenho de cada camada é a ordem da lista.

2.8.1.4. CanvasElement

Esta classe generaliza todos os elementos bidimensionais que podem ser colocados num *Canvas*, numa dada posição.

Atributos	
<i>name</i>	Nome do elemento. Este nome pode ser utilizado em referências como se tratasse de um nome de um elemento de modelação de nível M2.
<i>visible</i>	Atributo que especifica a visibilidade do elemento na vista (modificável em animações).
Portos	
<i>pointer</i>	Especifica sinais de saída, utilizáveis em elementos comportamentais assíncronos do <i>Canvas</i> onde se inclui o elemento e que correspondem a acções do utilizador sobre o elemento através de um dispositivo apontador.
Associações	
<i>animation</i>	Um conjunto de animações associadas ao elemento.
<i>anchor</i>	O conjunto de elementos ancorados. Um elemento pode ser ancorado a outro elemento, sendo a sua posição relativa à posição do elemento <i>anchoredTo</i> . Um elemento ancorado move-se automaticamente ao ser deslocado o elemento ao qual está ancorado.
<i>anchoredTo</i>	O elemento ao qual está ancorado o <i>CanvasElement</i> .

2.8.1.5. Node e Connector

Qualquer elemento de um *Canvas* é um nó (*Node*) ou conector (*Connector*). Um nó corresponde a um elemento de qualquer complexidade gráfica, vulgarmente um forma geométrica complexa. Um conector é um elemento que permite unir nós e é definido por uma linha definida pelo seu atributo *path* (herdado de *Polyline*). O caminho da linha de um conector não pode ser fechado. A posição de cada um dos seus extremos (*ConnectorEnd*) depende do ponto de um nó ao qual o extremo está associado, caso essa ligação seja realizada. O ponto de ligação a um nó é representado por instâncias da classe *Port*, com a sua posição definida de forma relativa ao centro do objecto.

2.8.1.6. Symbol

Um símbolo é um nó cuja representação gráfica e interactividade é definida num *Canvas*.

Atributos	
<i>canvas</i>	O nome composto de um <i>Canvas</i> . O canvas deverá ser um <i>Canvas</i> isolado ou um <i>canvas</i> que seja parte da definição do classificador onde se inclui o <i>Canvas</i> que contém o símbolo ou de um dos seus atributos ou extremos de associação navegáveis e visíveis.
<i>x, y, width, height</i>	A área ocupada pelo símbolo (modificável em tempo de execução).
<i>rotation</i>	Um valor em graus que representa a rotação sobre o centro geométrico da área para definir a forma final do nó (modificável em tempo de execução).

2.8.1.7. Group

Um grupo é um nó cuja definição compreende uma lista ordenada de elementos. A ordem de cada elemento na lista define a ordem de desenho numa vista.

Atributos	
<i>stroke, fill, font</i>	Estilos de desenho, modificáveis em tempo de execução, aplicáveis não ao grupo mas sim aos elementos do grupo que definem os seus estilos como herdados. A herança de estilos ocorre na estrutura hierárquica de grupos de um <i>Canvas</i> .

2.8.1.8. Layer

Uma camada é um grupo de elementos particular que faz parte da definição de um *Canvas*.

2.8.1.9. Color

Esta classe define um tipo de dados cujas instâncias representam cores no espaço RGB com oito *bits* de profundidade para cada componente.

2.8.1.10. Path

A classe *Path* define um caminho, utilizável, por exemplo, na definição de linhas ou formas geométricas complexas. Um caminho singular compreende um conjunto de segmentos interligados definidos por uma recta, uma curva cúbica ou uma curva quadrática. Uma forma geométrica pode ser definida à custa de um caminho singular fechado. Um *Path* compreende um conjunto de caminhos singulares.

Atributos	
definition	Um caminho é descrito por uma linguagem imperativa baseada em instruções de movimentação de um ponto em analogia com o movimento de uma caneta sobre o papel. As instruções disponíveis são: • <i>moveTo</i> – mover o ponto, sem desenhar. • <i>lineTo</i> – mover o ponto, desenhando uma linha recta. • <i>curveTo</i> – mover o ponto, desenhando uma linha Bézier cúbica ou quadrática. • <i>closePath</i> – fechar a linha actual. A gramática para especificação de caminhos, bem como a semântica, é baseada na SVG e apresentada no anexo VI.

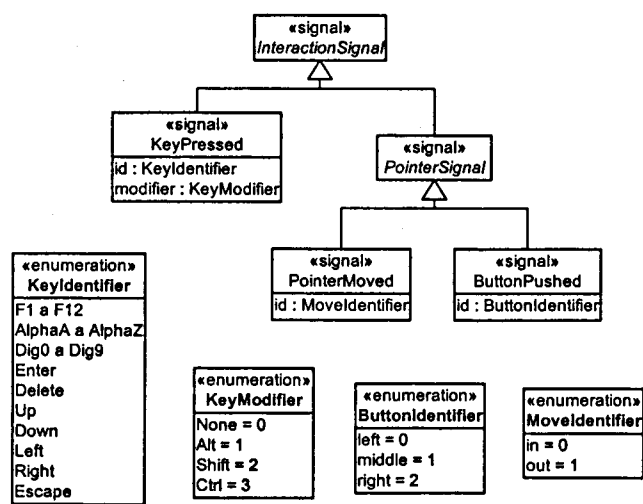


Figura 44. Modelo do pacote *Signals*.

2.8.1.11. InteractionSignal

Um sinal de interacção generaliza os sinais associados aos dispositivos de interface de IEDs, enviados quando o utilizador interage com uma vista. O significado semântico destes sinais é imediato da análise do diagrama da figura 44. Os atributos de cada sinal correspondem a uma representação virtual que será mapeada de formas distintas em dispositivos concretos.

2.8.2. Pacote Geometric

O modelo do pacote *Geometric* é apresentado nas figuras 45, 46 e 47. Este pacote compreende a definição de nós que representam formas geométricas, linhas, mapas de *pixels* e caixas de texto, bem como dos estilos de desenho

aplicáveis. A maior parte dos atributos de cada nó é modificável em tempo de execução e acessível em animações. A acessibilidade dos atributos é modelada pela sua visibilidade e tipo de acesso.

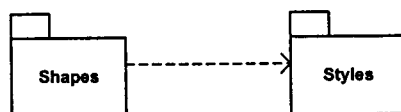


Figura 45. Modelo do pacote *Geometric*.

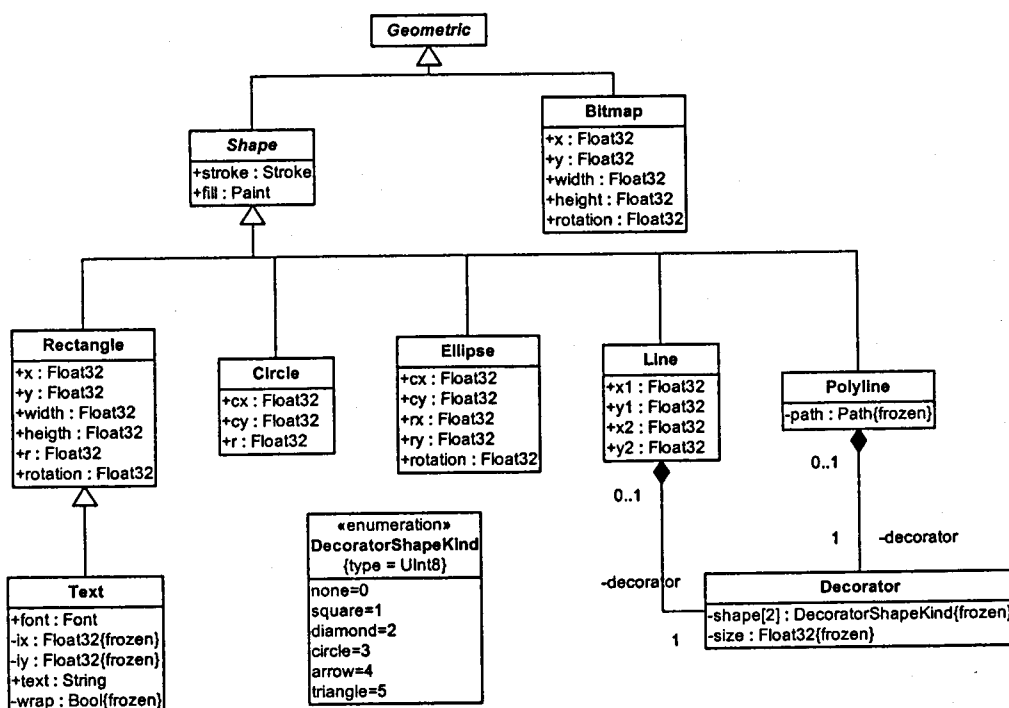


Figura 46. Modelo do pacote *Shapes*.

2.8.2.1. Bitmap

Um nó definido por uma matriz que contém um mapa de *pixels*.

Atributos	
<i>x, y, width, height</i>	A área rectangular da matriz do mapa de <i>pixels</i> .
<i>rotation</i>	Um valor em graus que representa a rotação sobre o centro geométrico da matriz para definir a forma final do nó.

2.8.2.2. Shape

Um nó definido por uma forma geométrica fechada ou aberta.

Atributos	
<i>stroke, fill</i>	Estilos de desenho para a linha e o preenchimento da forma geométrica.

2.8.2.3. Rectangle

Trata-se de um rectângulo.

Atributos	
<i>x, y, width, height</i>	A área do rectângulo.
<i>r</i>	Raio para desenho de cantos arredondados.
<i>rotation</i>	Um valor em graus que representa a rotação sobre o centro geométrico do rectângulo para definir a forma final do nó.

2.8.2.4. Circle

Trata-se de um círculo.

Atributos	
<i>font</i>	O estilo para desenho do texto.
<i>ix, iy</i>	O espaço entre os limites do rectângulo que define a área da caixa de texto e a área reservada para desenho do texto.
<i>text</i>	O texto a desenhar.
<i>wrap</i>	Atributo booleano que determina a forma com é desenhado o texto. Caso este atributo seja verdadeiro o texto pode ser partido em múltiplas linhas de forma a se adequar à forma do rectângulo.

2.8.2.5. Ellipse

Trata-se de uma elipse.

Atributos	
<i>cx, cy</i>	O centro da elipse.
<i>rx, ry</i>	Os raios da elipse na horizontal e na vertical.
<i>rotation</i>	Um valor em graus que representa a rotação sobre o centro geométrico da elipse para definir a forma final do nó.

2.8.2.6. Text

Trata-se de uma caixa de texto.

Atributos	
<i>x, y, width, height</i>	A área do rectângulo.
<i>r</i>	Raio para desenho de cantos arredondados.
<i>rotation</i>	Um valor em graus que representa a rotação sobre o centro geométrico do rectângulo para definir a forma final do nó.

2.8.2.7. Line

Trata-se de um linha definida por um segmento de recta.

Atributos	
<i>x1, y1</i>	O ponto inicial do segmento de recta.
<i>x2, y2</i>	O ponto final do segmento de recta.
Associações	
<i>decorator</i>	Duas formas geométricas que podem figurar nos extremos da linha.

2.8.2.8. Polyline

Trata-se de um linha, ou uma forma geométrica fechada, definida por um caminho.

Atributos	
<i>path</i>	O caminho que define a forma do nó.
Associações	
<i>decorator</i>	Duas formas geométricas que podem figurar nos extremos da linha, caso esta não seja fechada.

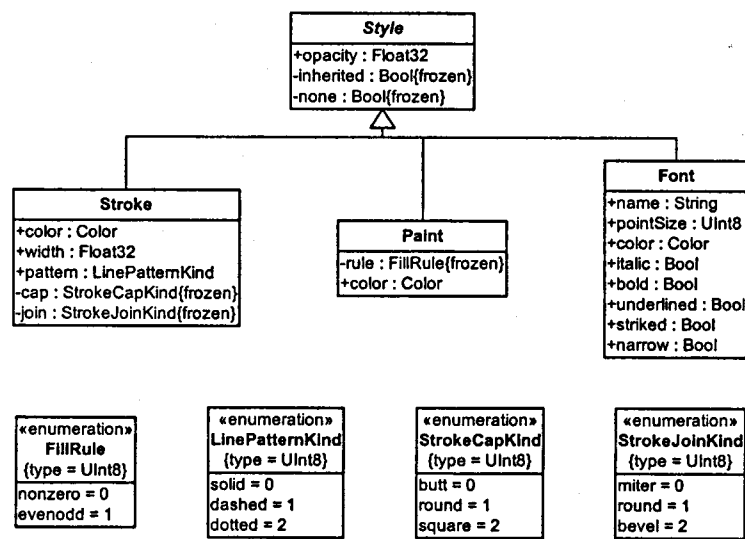


Figura 47. Modelo do pacote *Styles*.

2.8.2.9. Style

A classe *Style* generaliza todos os estilos de desenho.

Atributos	
<i>opacity</i>	Um nível percentual de opacidade para desenho do elemento ao qual se aplica o estilo.
<i>inherited</i>	Um atributo booleano que determina a herança de estilos. Caso o estilo apresente um valor verdadeiro neste atributo então o estilo da mesma classe concreta a aplicar ao elemento é o estilo imediatamente acima na hierarquia de grupos e não o definido na instância da classe.
<i>none</i>	Caso este atributo apresente o valor booleano verdadeiro a componente do elemento ao qual se aplica o estilo não é desenhada.

2.8.2.10. Stroke

Um estilo para definição de desenho de linhas.

Atributos	
<i>color</i>	Cor para desenho de linha.
<i>width</i>	Largura da linha.
<i>pattern</i>	Padrão sólido, tracejado ou ponteadado.
Atributos (continuação)	
<i>cap</i>	Estilo para terminação de linhas: <i>butt</i> –  <i>round</i> –  <i>square</i> – 
<i>join</i>	Estilo para vértices de linhas: <i>miter</i> –  <i>round</i> –  <i>bevel</i> – 

2.8.2.11. Paint

Um estilo para definição de preenchimento de áreas.

Atributos	
<i>color</i>	Cor de preenchimento.
<i>rule</i>	Regra para preenchimento de formas complexas, aplicável apenas a <i>Polyline</i> 's. Admitindo um raio infinito com origem no ponto, qualquer ponto de uma forma geométrica é considerado fundo a preencher, se <i>nonzero</i> – o número de segmentos do caminho que cruzam o raio da esquerda para a direita subtraído ao número de segmentos do caminho que cruzam o raio da direita para a esquerda for não nulo. <i>evenodd</i> – o número de todos os segmentos do caminho cruzados pelo raio for ímpar.

2.8.2.12. Font

Um estilo para definição de desenho de texto.

Atributos	
name	Nome da fonte a utilizar para desenhar o texto.
pointSize	Tamanho dos caracteres definido em pontos de fonte.
color	Cor para desenho do texto.
italic, bold, underlined, striked, narrow	Atributos booleanos que definem propriedades comuns para desenho de texto.

2.8.3. Pacote Animation

O modelo do pacote *Animation* é apresentado na figura 48. Este modelo permite a definição de animações cíclicas e esporádicas de elementos constituintes de um *Canvas*.

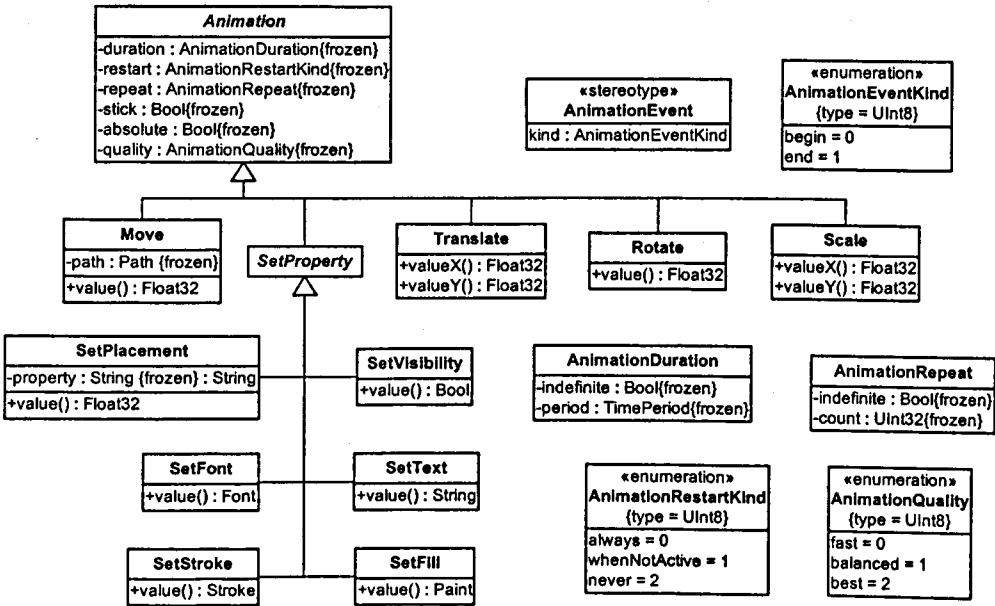


Figura 48. Modelo do pacote *Animation*.

2.8.3.1. Animation

Esta classe generaliza animações. Uma animação corresponde a uma acção a executar sobre o modelo gráfico que é despoletada por um evento. Assim, um evento da meta-classe *Event* deverá estar sempre associado a uma animação para definir os instantes do seu início. Dependendo do tipo de evento associado uma animação, esta pode ser despoletada de forma cíclica, em resposta ao envio de um sinal ou em resposta a alterações do modelo de dados. Caso não exista evento a animação é despoletada no início da visualização de cada vista. Um segundo evento pode também estar disponível para terminação da animação, caso esta esteja em execução no instante da sua ocorrência.

Atributos	
<i>duration</i>	Duração temporal da animação.
<i>restart</i>	Determina os instantes em que a animação pode ser reiniciada: <i>always</i> – A animação é reiniciada a cada ocorrência do evento, mesmo que esteja já a correr. <i>whenNotActive</i> – a animação apenas é reiniciada caso ocorra o evento e não exista qualquer sua instância em execução. <i>never</i> – a animação apenas é executada um vez durante um período de visualização de uma vista.
<i>repeat</i>	Número de repetições automáticas da animação após ter sido despoletada.
<i>stick</i>	Caso este atributo apresente o valor falso as características do objecto animado são repostas após a finalização da animação. Caso contrário o objecto permanece com as características que apresenta no instante final da animação.
<i>absolute</i>	Determina se a animação provoca efeitos absolutos ou relativos na posição ou rotação do objecto animado.
<i>quality</i>	Um qualificador da qualidade da animação que será dependente do dispositivo de visualização.
<i>element</i>	O elemento animado.

2.8.3.2. AnimationEvent

Este estereótipo deve ser associado a qualquer evento de uma animação.

Definição de Etiquetas	
<i>kind</i>	Tipo de evento: evento de início de animação ou evento de fim de animação.

2.8.3.3. Move

Uma animação que compreende a movimentação do objecto animado ao longo de um caminho. Esta animação permite apenas alterar a posição do objecto. Para animações com velocidade variável deverão ser utilizados eventos periódicos e animações relativas.

Atributos	
<i>path</i>	Caminho singular para a animação.
Operações	
<i>value</i>	O valor percentual (face ao comprimento da linha) para colocação do centro geométrico do objecto animado sobre o caminho.

2.8.3.4. Translate

Uma animação que compreende a movimentação do centro do objecto animado. Esta animação permite apenas alterar a posição do objecto. Para animações com velocidade variável deverão ser utilizados eventos periódicos e animações relativas.

Operações	
<i>valueX</i>	O valor para colocação da abcissa do centro geométrico do objecto animado.
<i>valueY</i>	O valor para colocação da ordenada do centro geométrico do objecto animado.

2.8.3.5. Scale

Uma animação que compreende a alteração da área do objecto animado. O ponto fixo é o canto superior esquerdo do objecto. Esta animação permite apenas alterar a dimensão do objecto. Para animações com velocidade variável deverão ser utilizados eventos periódicos e animações relativas.

Operações	
<i>valueX</i>	O valor para a nova dimensão horizontal do objecto animado.
<i>valueY</i>	O valor para a nova dimensão vertical do objecto animado.

2.8.3.6. Rotate

Uma animação que compreende a rotação do centro do objecto. Esta animação permite apenas alterar a posição do objecto. Para animações com velocidade variável deverão ser utilizados eventos periódicos e animações relativas.

Operações	
value	O valor, em graus, para rotação sobre o centro geométrico do objecto animado.

2.8.3.7. SetProperty

Uma animação genérica que compreende a alteração de um atributo de posição ou dimensão (*SetPlacement*), da visibilidade (*SetVisibility*), estilo (*SetFont*, *SetFill*, *SetStroke*) ou texto (*SetText*) do objecto animado.

Atributos	
property	O nome do atributo a alterar, apenas disponível na animação <i>SetPlacement</i> . O atributo deve ser visível.
Operações	
value	Uma operação que determina o novo valor da propriedade visual associada à animação. O tipo do parâmetro de retorno da operação (único na sua interface) é igual ao tipo da propriedade a modificar.

2.8.4. Pacote Widgets

O modelo do pacote *Widgets*, apresentado na figura 49, incorpora elementos básicos de interface ao utilizador cuja visualização é dependente da plataforma de implementação e que podem estar associados a um *Canvas*.

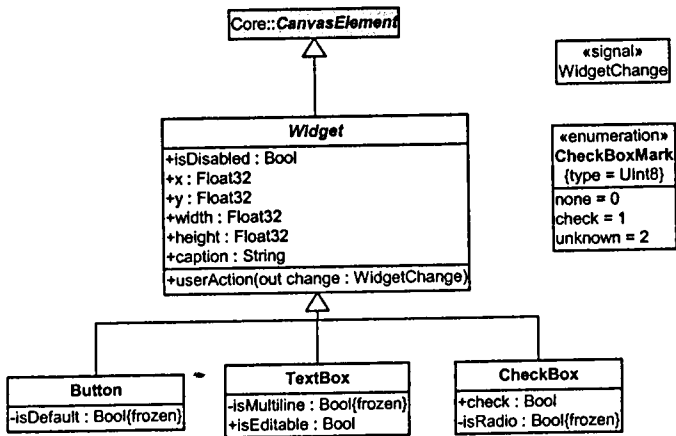


Figura 49. Modelo do pacote *Widgets*.



2.8.4.1. Widget

Um elemento comum de aplicações de interface ao utilizador.

Atributos	
<i>isDisabled</i>	Atributo booleano que determina se o utilizador pode interagir com o <i>Widget</i> (falso) ou não (verdadeiro).
<i>x, y, width, height</i>	A área ocupada pelo elemento.
<i>caption</i>	O texto visível no elemento, no caso de botões ou o texto editável em caixas de texto.
Portos	
<i>userAction</i>	Especifica um único sinal de saída, sem parâmetros que é enviado sempre que ocorre uma interacção com o <i>Widget</i> que implica uma mudança do seu estado, ou, no caso de botões simples, sempre que o botão é premido.

2.8.4.2. Button

Trata-se de um botão simples.

Atributos	
<i>isDefault</i>	Caso este atributo apresente o valor verdadeiro o botão é o botão apresentado ao utilizador como a escolha por defeito.

2.8.4.3. TextBox

Trata-se de uma caixa de texto.

Atributos	
<i>isMultiline</i>	Atributo booleano que determina se o texto apresentado pode conter mais do que uma linha.
<i>isEditable</i>	Atributo booleano que determina se o utilizador pode modificar o conteúdo da caixa de texto.

2.8.4.4. CheckBox

Trata-se de um botão que permite realizar escolhas.

Atributos	
<i>check</i>	A posição actual do botão: • <i>none</i> – opção não escolhida. • <i>check</i> – opção escolhida. • <i>unknown</i> – indeterminado.
<i>isRadio</i>	Atributo que determina se o botão pertence a um grupo de escolhas exclusivas.

2.9. Biblioteca de Operações Comuns

Dada a necessidade frequente da inclusão de operações comuns na modelação de comportamento introduz-se um modelo de nível M1 (instância da ASML) de uma biblioteca com funções matemáticas, de manipulação de cadeias de caracteres e de conversão. Este modelo, apresentado nas figuras 50 e 51, é parte integrante da ASML.

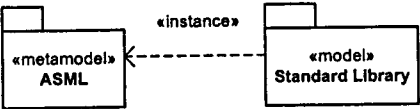


Figura 50. Pacote *Standard Library*.

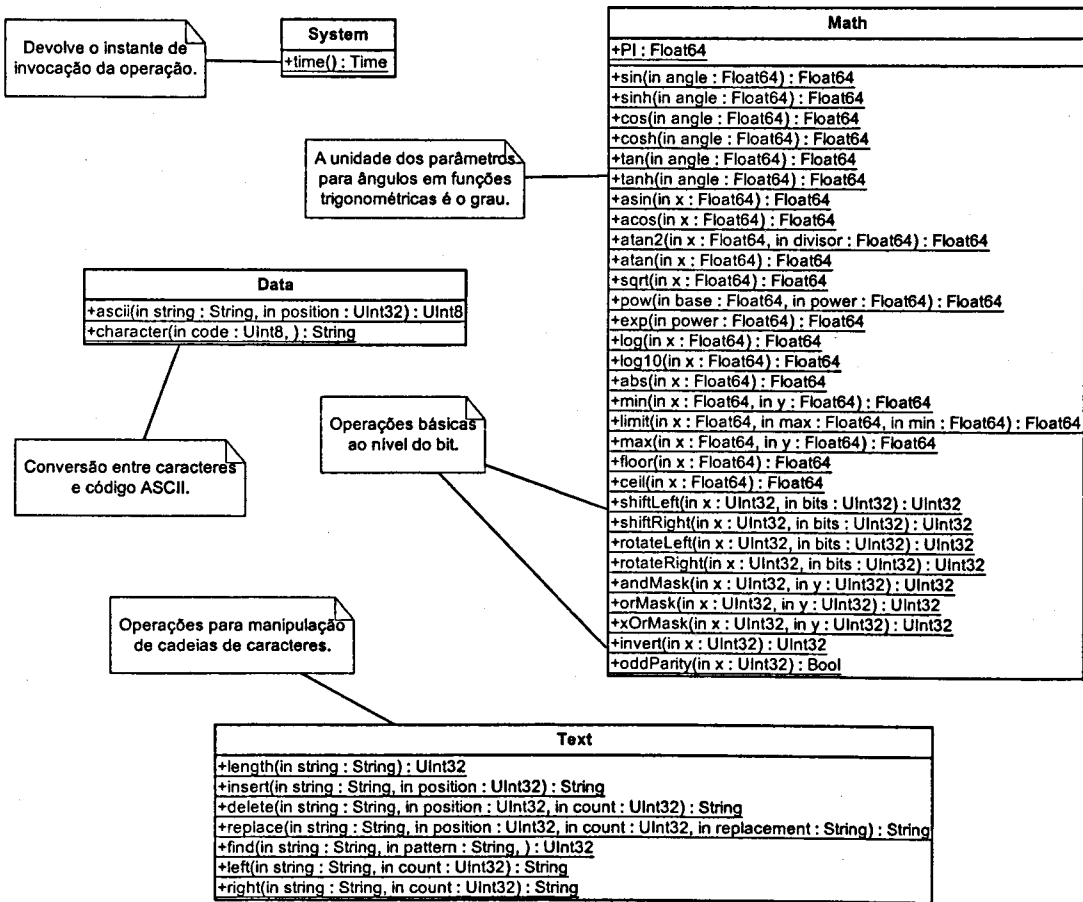


Figura 51. Modelo de operações da *Standard Library*.

Capítulo 3

Notação

Neste capítulo apresenta-se a notação da ASML numa definição ligeira dos elementos generalizados por *PresentationElement*. Esta notação corresponde a um modelo léxico e a um modelo sintáctico concreto, instância da sintaxe definida no capítulo anterior. O modelo definido neste capítulo é útil para a definição dos componentes de interface ao utilizador de uma ferramenta de engenharia.

3.1. Visão Geral

Na figura 52 apresenta-se o modelo notacional que especializa os elementos de interacção da ASML (figura 53).

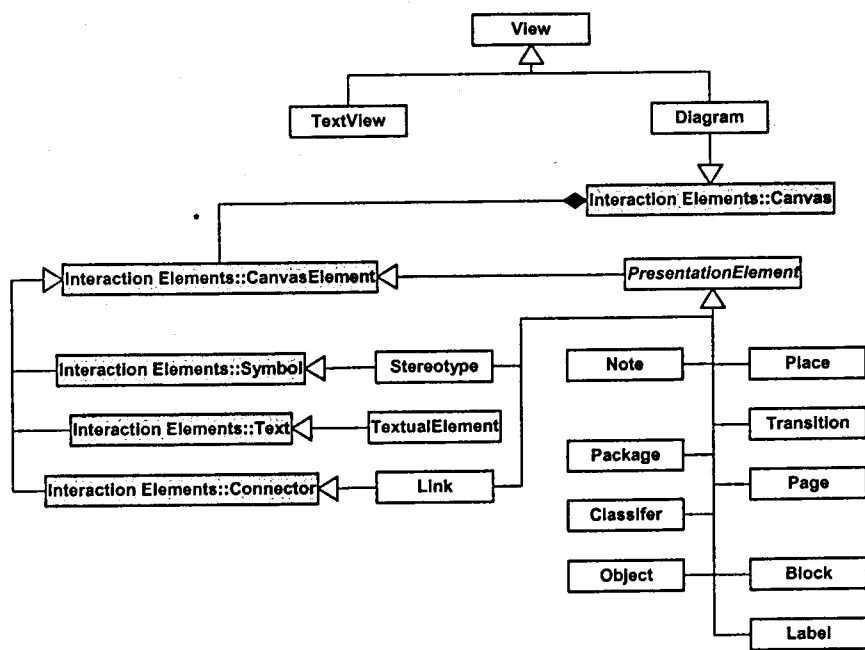


Figura 52. Modelo da notação da linguagem – pacote *Notation*.

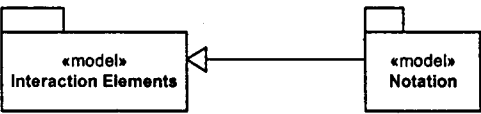


Figura 53. Pacote Notation.

A notação apresentada neste capítulo não pretende ser exaustiva, mas sim ilustrar os vários elementos notacionais e estabelecer a sua relação com os elementos da sintaxe abstracta, sendo o modelo léxico e gramatical completo remetido para anexo.

A notação da linguagem é essencialmente composta por elementos gráficos que podem figurar em diagramas e por uma notação textual alternativa para expressões e procedimentos. Esta notação reflecte a notação UML, a notação IEC 61131-3, bem como a representação comum das redes de Petri. Os elementos gráficos, apresentados na figura 52 e detalhados no anexo VI, são organizados em diagramas e representam elementos de modelação concretos do meta-modelo sintáctico abstracto: classificadores, relações, objectos, ligações, redes de Petri, procedimentos e expressões. Os elementos de interacção são também definíveis de forma gráfica, estando a sua notação implícita na definição semântica, pelo que esta não é abordada neste capítulo.

3.2. Elementos Gráficos Comuns

A definição visual dos vários elementos notacionais gráficos compreende várias figuras geométricas que podem incluir elementos de texto (*TextualElement*). A estes elementos visuais de texto aplica-se a sintaxe apresentada no anexo V.

Um dos elementos notacionais gráficos que pode estar presente em qualquer diagrama é a anotação (*Note*). Uma anotação pode ser utilizada na representação de comentários, restrições, etiquetas ou blocos de texto associados a procedimentos ou expressões (figura 54).

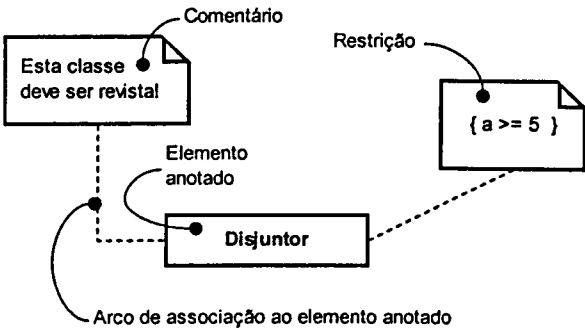


Figura 54. Anotações.

Um pacote é representado pelo símbolo apresentado na figura 55 (*Package*). Caso ao pacote esteja atribuído um estereótipo ou etiqueta, o seu nome pode ser colocado junto do nome do pacote, com a notação para elementos textuais. Como qualquer outro elemento de modelação, um pacote com um estereótipo associado pode ser representado pelo seu símbolo notacional ou completamente substituído pelo elemento de interacção associado ao estereótipo (*Stereotype*). No caso do pacote representar um modelo, a palavra-chave «*model*» pode também ser incluída.

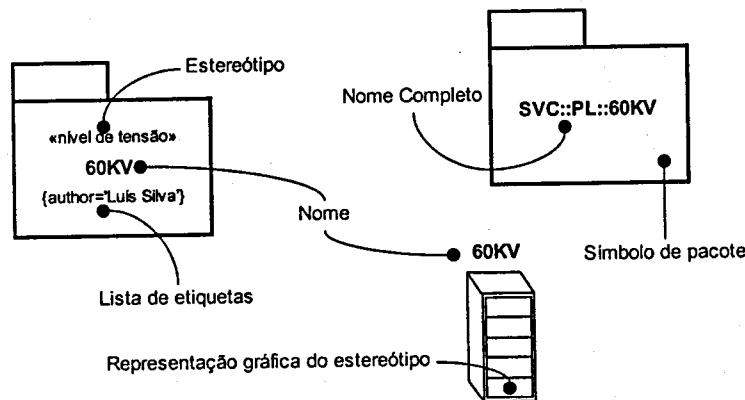


Figura 55. Três representações alternativas de um mesmo pacote.

Em qualquer diagrama podem ainda figurar quaisquer elementos generalizados por *CanvasElement*.

3.3. Classificadores

Os classificadores e estereótipos são apresentados genericamente pelo seu símbolo representativo (*Classifier*), ilustrado na figura 56.

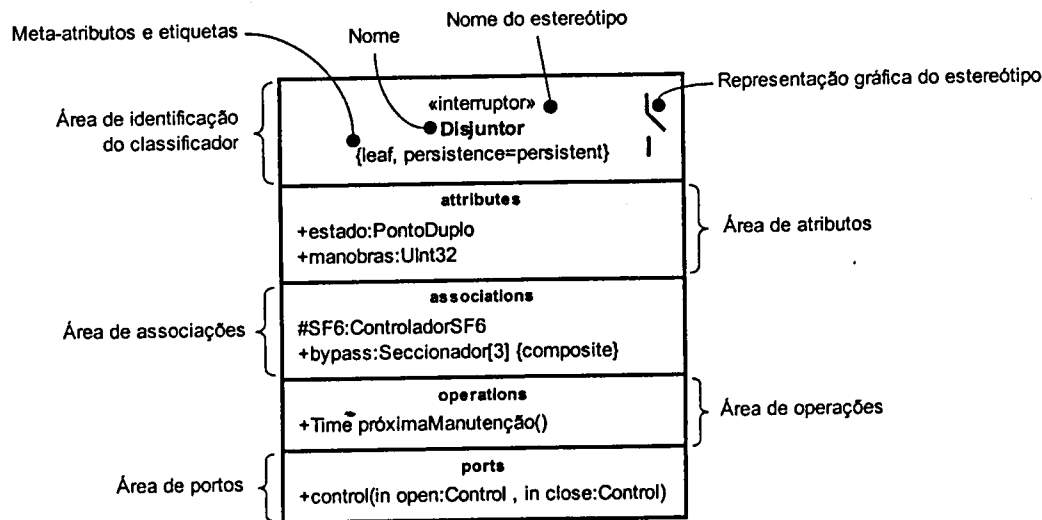


Figura 56. Classificador.

Esta representação, compreende um rectângulo composto por vários compartimentos rectangulares justapostos sequencialmente na vertical. O compartimento de identificação do classificador deve estar sempre presente no cimo de todos os compartimentos. Os compartimentos restantes representam os vários *Feature's* do classificador e a sua figuração é opcional. A figura 57 apresenta exemplos de símbolos para os vários classificadores. Um classificador activo é representado com uma espessura superior na linha do símbolo.

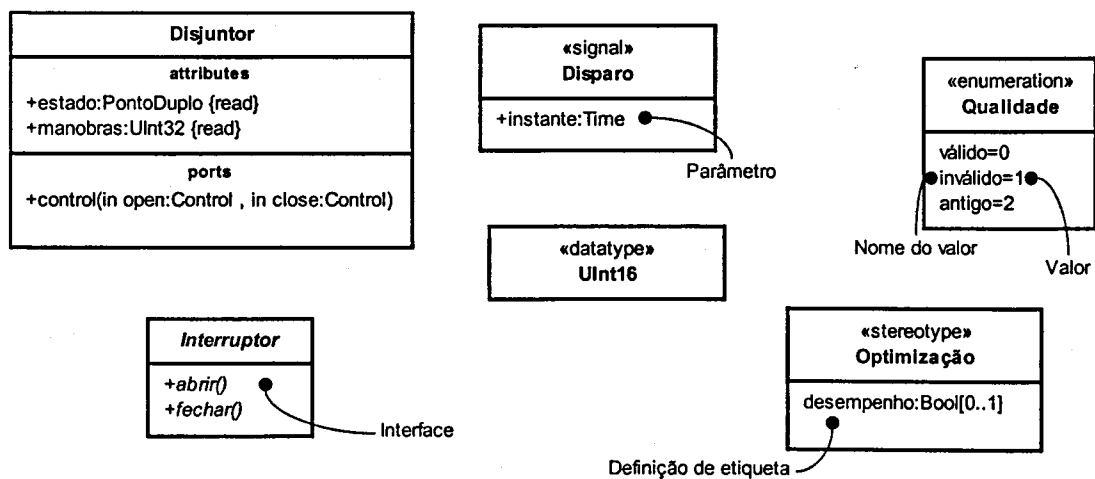


Figura 57. Classificadores.

As relações são representadas por arcos de ligação (*Link*) interligando os símbolos dos elementos relacionados. Nos extremos dos arcos podem figurar símbolos decorativos que representam direccionalidade ou o papel dos elementos relacionados Os símbolos para relações são generalizados na tabela 14 e apresentados na figura 58.

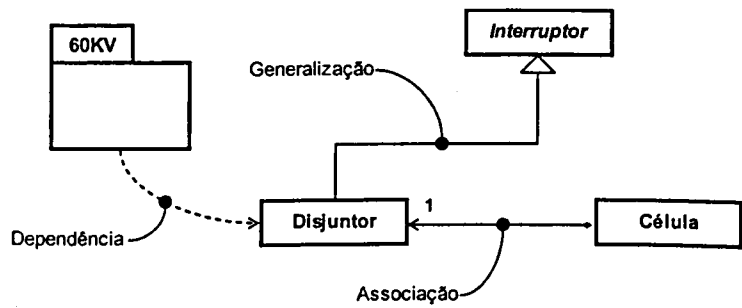


Figura 58. Relações.

Tabela 14. Simbologia para relações.

Relação	Simbologia	Descrição
Qualquer	«stereotype» name	O nome da relação e os seus estereótipos podem ser apresentados no centro do arco da relação.
Dependência	----->	O decorador é colocado no extremo ligado ao elemento alvo da dependência.
Generalização	----->	O decorador é colocado no extremo ligado ao elemento especializado.
Associação	-----◆	O decorador indica que o extremo da associação é uma composição de elementos do extremo oposto.
	←----->◆	A navegabilidade de um extremo pode ser representada por uma seta.
	----- 0..1 name	O nome e multiplicidade do extremo podem figurar no símbolo da associação. Neste exemplo a associação é simples, ou seja não é uma composição.

3.4. Instâncias

Os objectos podem ser apresentados pelo símbolo de objecto (*Object*) ou por qualquer elemento de interacção (*Symbol*) associado ao seu classificador. As figuras 59 e 60 apresentam exemplos de instâncias.

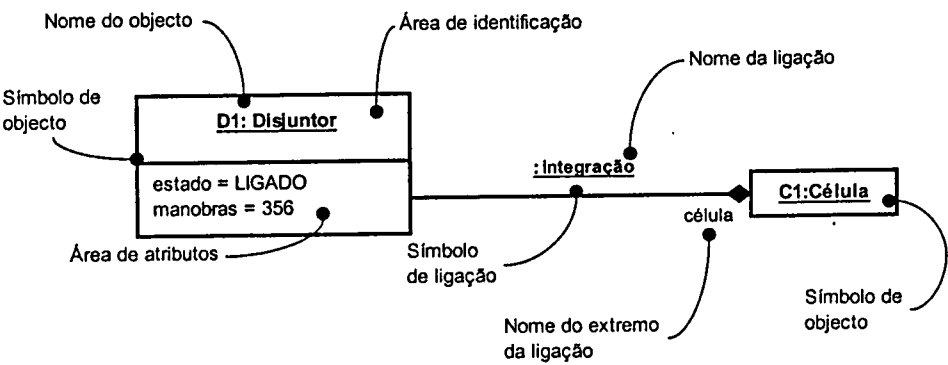


Figura 59. Notação base para instâncias.

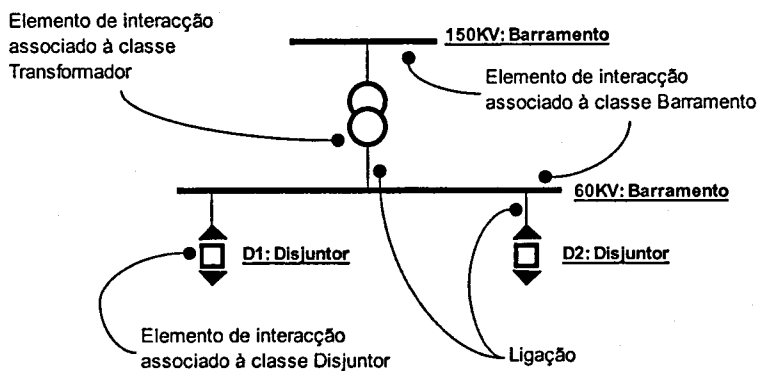


Figura 60. Uma notação alternativa para instâncias.

O símbolo de objecto pode ser utilizado para representar instâncias de classificadores com existência individual no modelo ou associados a atributos de objectos. As ligações são representadas por arcos de ligação (*Link*) estabelecendo interligações entre objectos. O nome da ligação pode ser apresentado no centro do arco da ligação e os extremos da ligação podem apresentar o nome do extremo de associação que instanciam bem como decoradores de agregação da simbologia de associações.

3.5. Redes de Petri

Para redes de Petri adopta-se uma notação comum, apresentada na figura 61.

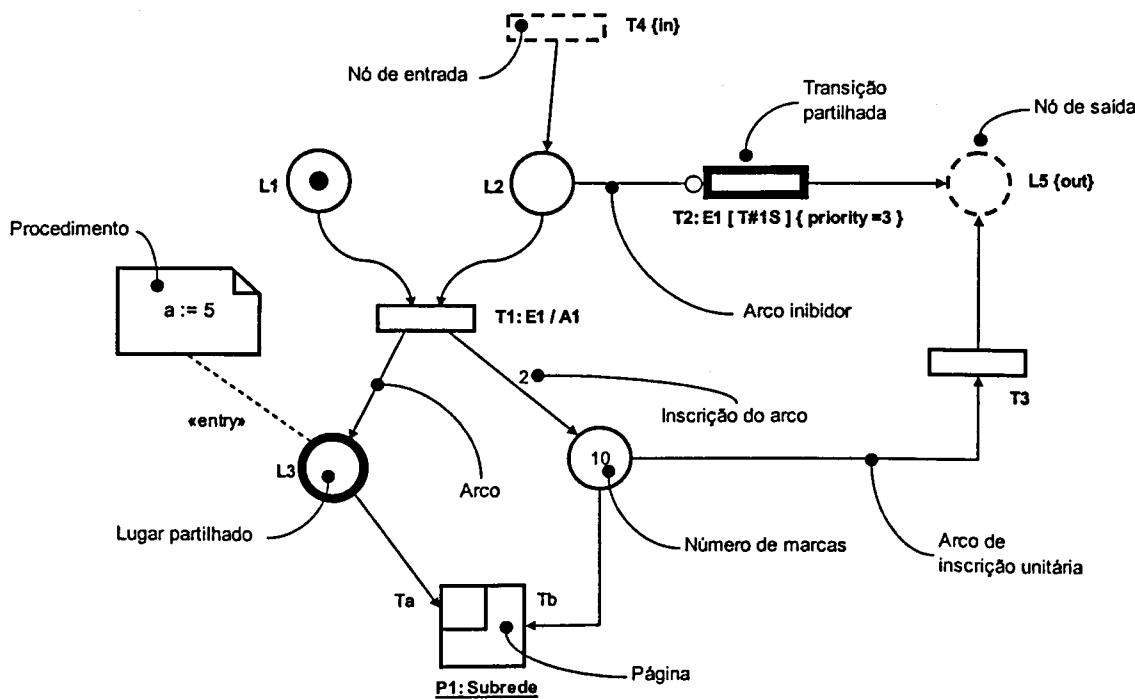


Figura 61. Redes de Petri.

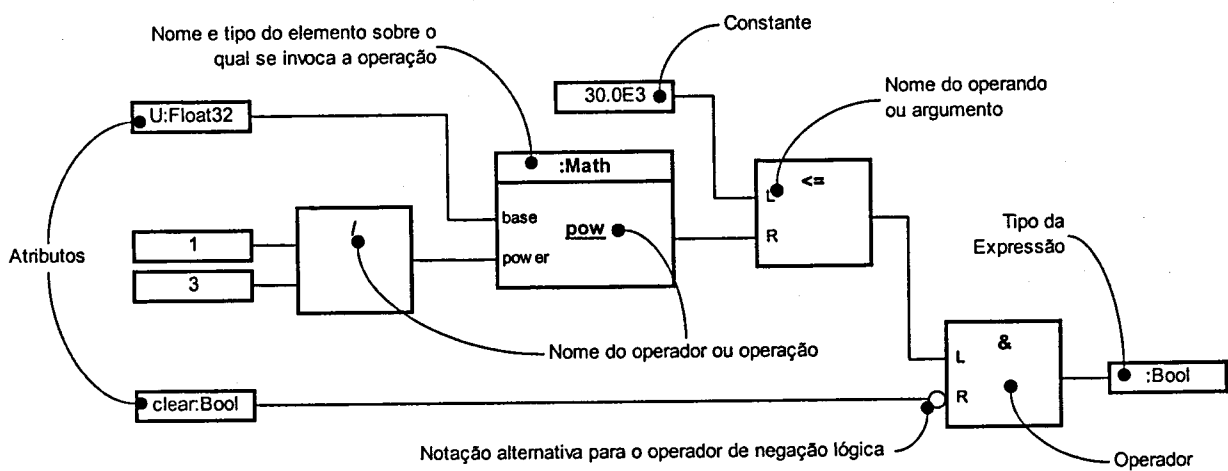


Figura 62. Expressão.

A representação gráfica da implementação de procedimentos é semelhante, podendo compreender múltiplas expressões que se ordenam na vertical sendo a expressão de topo a primeira a ser executada. Para a definição de procedimentos estão disponíveis etiquetas adicionais, com notações próprias, que permitem representar instruções de controlo e o retorno do procedimento. A figura 63 apresenta a definição da implementação de um procedimento que pode ser traduzido, em linguagem textual, por:

```
aux : Float32 := 30.0E3;
IF GetMode() = Mode::Auto THEN aux := 35.0E3;
level := aux;
RETURN value > level;
```

Os lugares são representados por elipses (*Place*) contendo, no seu interior, o número de marcas presentes no lugar representado por círculos preenchidos ou por um número inteiro. O nome do lugar é inscrito junto da elipse. As transições são representadas por rectângulos (*Transition*) contendo, eventualmente, o evento de disparo, a condição e a acção de efeito. A notação para nós partilhados é idêntica à apresentada mas com uma espessura de linha superior. Nós de entrada ou de saída apresentam os seus limites a tracejado. Um arco da rede é representado por uma ligação (*Link*) entre nós sendo a sua orientação ilustrada pelo decorador no extremo ligado ao nó onde é incidente. Este decorador pode apresentar uma de duas formas: a seta, para arcos genéricos, ou a circunferência, para arcos inibidores. A inscrição pode ser representada junto do centro do arco, ou omitida, caso o seu valor seja constante e unitário. Páginas incluídas na rede são representadas pelo duplo rectângulo (*Page*). Os arcos incidentes em nós de entrada ou com origem em nós de saída da subrede podem ser anotados com o nome do nó ao qual se associam.

3.6. Expressões e Procedimentos

Os procedimentos e as expressões podem ser definidos em diagramas de blocos ou em módulos de texto. A representação textual segue em linhas gerais a linguagem ST da IEC 61131-3 e está completamente definida pelo léxico e gramática apresentados no anexo IV. A representação gráfica, semelhante à linguagem FBD da IEC 61131-3, é introduzida neste ponto do documento.

Os símbolos gráficos utilizados para representar instruções de processamento são o bloco (*Block*), a ligação (*Link*) e a etiqueta (*Label*). Os blocos representam operadores da linguagem ou a invocação de operações, as etiquetas representam referências para variáveis, atributos ou extremos de associação e as ligações estabelecem a hierarquia de expressões compostas de uma expressão. Nos blocos figuram pontos de ligação aos quais se podem associar os extremos de arcos de ligação e que representam os parâmetros da operação ou operandos do operador. Os parâmetros de entrada são apresentados no lado esquerdo do bloco e os parâmetros de saída no seu lado direito. Um extremo de uma ligação, se associado ao lado esquerdo de uma etiqueta ou bloco, representa a atribuição de valor ao elemento referido. Caso esteja associado ao lado direito da etiqueta, o extremo representa a leitura do valor. Os extremos de uma ligação devem obrigatoriamente estar associados a elementos com o mesmo tipo. A figura 62 apresenta a definição da implementação de uma expressão booleana que pode ser traduzida, em linguagem textual, por:

```
30.0E3 <= Math::pow(U, 1/3) AND NOT clear
```

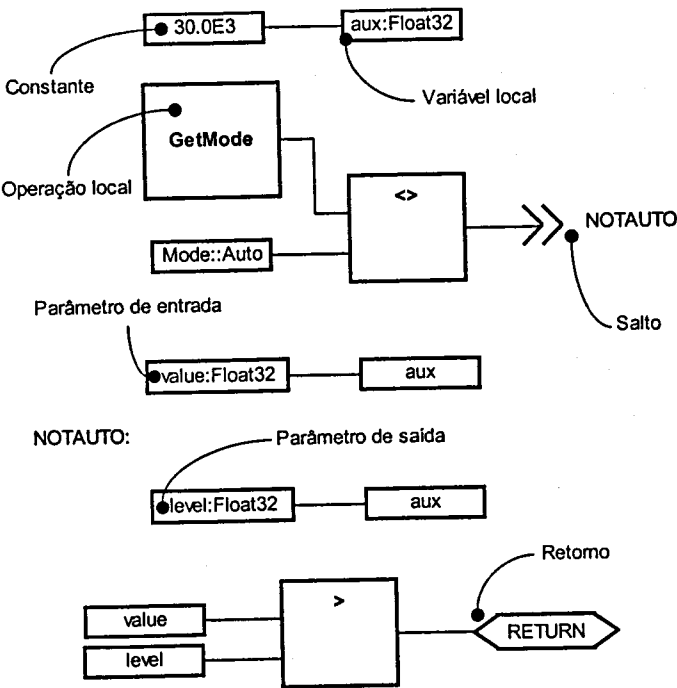


Figura 63. Procedimento.

Capítulo 4

Semântica de Domínio para Sistemas de Automação

Neste capítulo apresenta-se uma instância da ASML que compreende um modelo de classes, uma definição semântica e regras sintácticas para automação de instalações do sistema eléctrico. A definição desta semântica de domínio é necessária uma vez que a linguagem ASML definida não apresenta características que a tornem directamente aplicável no domínio da engenharia de automação de instalações do sistema eléctrico. O modelo definido neste capítulo é baseado nos modelos definidos na arquitectura IEC 61850 e serve de base à definição dos componentes de uma ferramenta de engenharia directamente envolvidos no suporte à actividade de engenharia de automação.

4.1. Visão Geral e Relação com o Sistema de Automação e a Ferramenta

A semântica para automação de sistemas de energia compreende um modelo ASML constituído por dois pacotes (figura 64): *LogicalSystem* e *PhysicalSystem*. Esta organização é intuitiva e apresenta uma relação directa com os componentes essenciais de um sistema de automação, segundo o modelo definido na primeira parte da dissertação.

O pacote *LogicalSystem* compreende as regras semânticas para definição de modelos de dispositivos e nós lógicos, capazes de modelar de forma independente o sistema lógico de automação. O pacote *PhysicalSystem* inclui regras para modelação da configuração do sistema primário, permitindo a definição de classes de equipamentos e a definição gráfica, através de diagramas unifilares, de instalações concretas. Este pacote inclui também a arquitectura para modelação da configuração e parametrização do sistema secundário, que engloba a definição de IEDs, incluindo a sua configuração e parametrização internas, as associações comunicativas entre IEDs, a atribuição de funções a dispositivos e a relação dos IEDs terminais com o sistema primário.

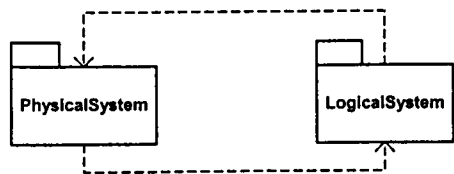


Figura 64. Visão geral da semântica para automação de sistemas de energia.

Pressupõe-se, na definição deste modelo de domínio, a arquitectura para a ferramenta de engenharia representada pelo diagrama da figura 65.

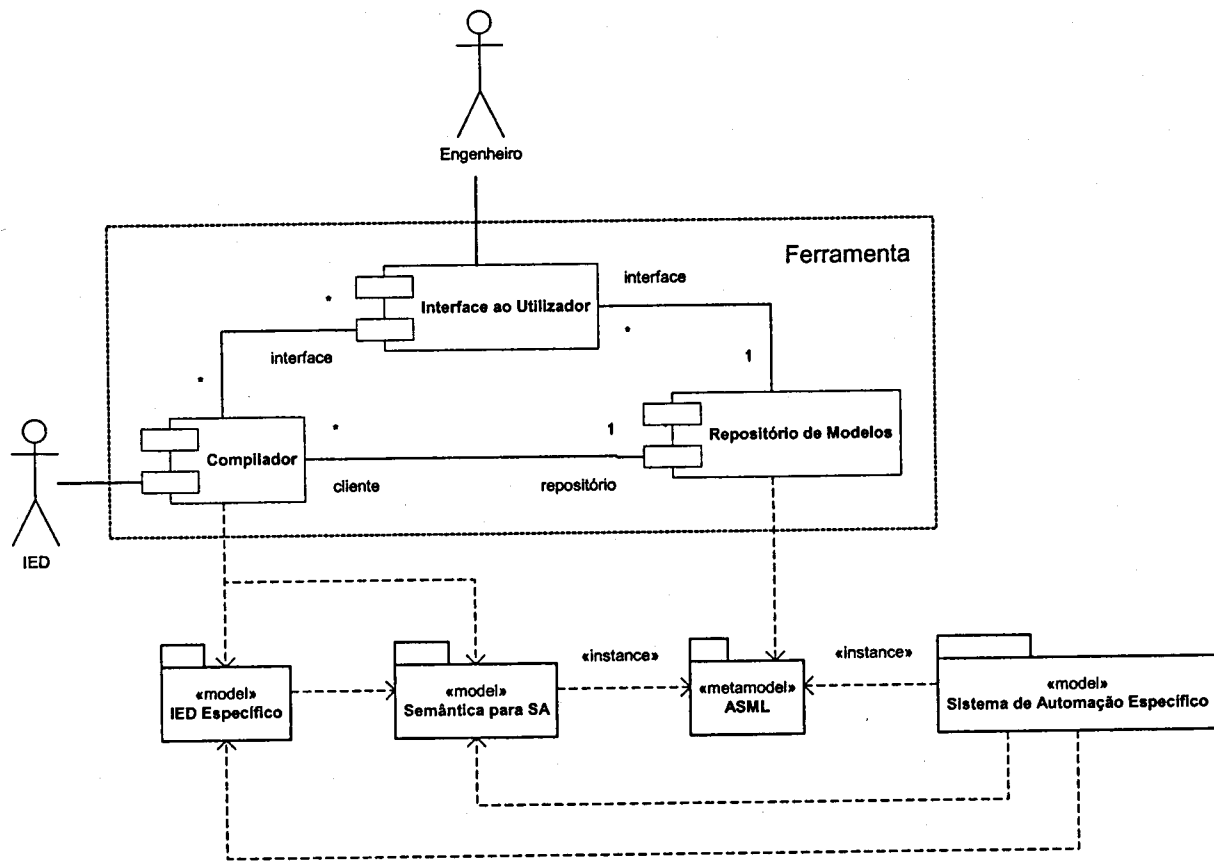


Figura 65. Arquitectura da ferramenta.

O núcleo central da ferramenta compreende componentes essencialmente dependentes do meta-modelo ASML, nomeadamente o repositório, para armazenamento de modelos, e os componentes de interface ao utilizador, para manipulação de modelos segundo a notação ASML, ou componentes de simulação. A ferramenta deverá integrar também componentes que dependem não só do meta-modelo ASML, mas também da semântica para sistemas de automação e de modelos de IEDs específicos. As funções típicas destes componentes compreendem a transformação de modelos ASML para modelos de parametrização desses dispositivos.

4.2. Sistema Primário

Os elementos do pacote *PhysicalSystem* vocacionados para a modelação do sistema primário compreendem os estereótipos da figura 66. Qualquer dispositivo primário deverá ser modelado numa classe ASML estereotipada com um de dois estereótipos: *PrimaryApparatus* ou *PrimaryConnection*. A todas as classes referentes a equipamentos activos

(disjuntores, seccionadores, bancos de condensadores, geradores ou outros) deverá ser aplicado o estereótipo *PrimaryApparatus*. Dispositivos que estabelecem essencialmente ligação eléctrica (barramentos ou linhas) são modelados por classes com o estereótipo *PrimaryConnection*. A identificação do tipo de dispositivo (disjuntor, seccionador, transformador, etc.) é realizada com recurso à etiqueta *primaryDeviceType*, cuja presença é obrigatória, e que denota uma classificação do dispositivo segundo a IEC 61850-6 [13]. Um dispositivo físico pode também ser visto como uma composição de outros dispositivos físicos, pelo que pode ser modelado por uma classe isolada ou por uma classe com associações de composição com outras classes de dispositivos. As ligações eléctricas entre dispositivos primários que não são âmbito de controlo ou protecção, mas que determinam a topologia de uma instalação pela interligação física de dispositivos, são modeladas com recurso a associações simples entre classes de dispositivos.

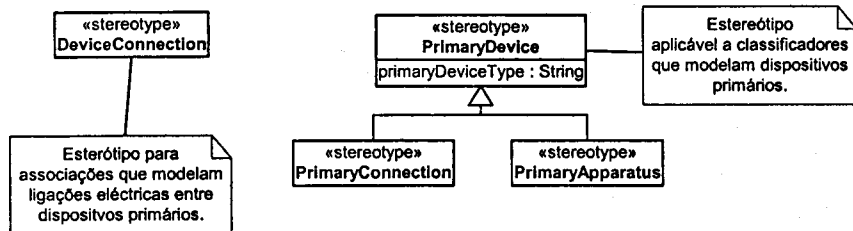


Figura 66. Modelo ASML para modelação do sistema primário.

A sintaxe para classes estereotipadas com *PrimaryDevice* e seus descendentes é ilustrada na figura 67.

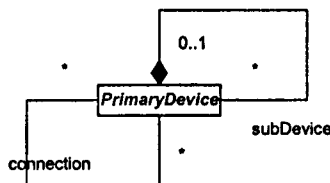


Figura 67. Regras sintáticas para modelação do sistema primário.

Uma classe de dispositivo primário pode conter quaisquer elementos de modelação associados a classes ASML que se considerarem adequados para modelar o dispositivo concreto, nomeadamente integrar uma dada estrutura de herança de classes. Todas as relações de composição entre classes de dispositivos primários, tal como atributos com tipos associados a classes de dispositivos, são consideradas relações entre dispositivo e sub-dispositivo. Todas as associações simples entre classes de dispositivos estereotipadas com *DeviceConnection* representam ligações eléctricas entre dispositivos primários. Qualquer classe de dispositivo primário deverá conter pelo menos um *Canvas* representativo de forma a permitir a criação de modelos de instalações recorrendo, em alternativa à notação ASML, a diagramas do tipo

unifilar⁴. A simbologia para dispositivos deve seguir, sempre que possível, normas internacionais aplicáveis, embora notações alternativas possam ser definidas. A figura 68 ilustra um exemplo simples de modelação.

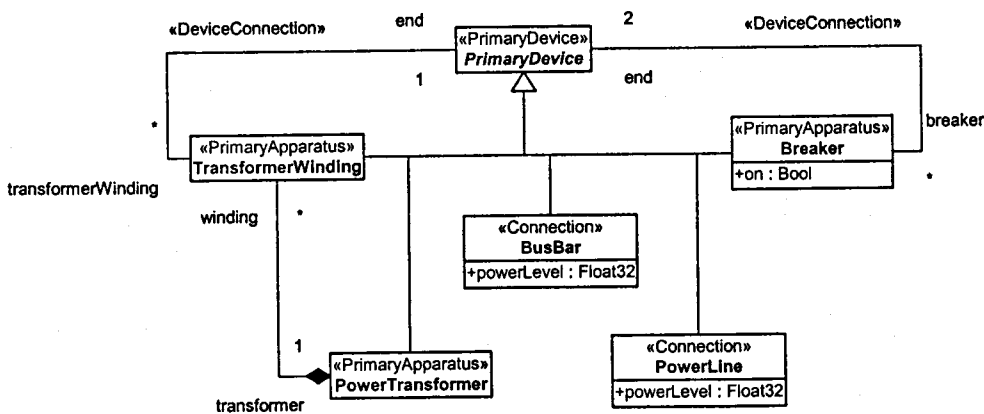


Figura 68. Um exemplo simples de modelação de dispositivos primários.

4.3. Sistema Lógico

Os elementos fundamentais do pacote *LogicalSystem* compreendem os estereótipos da figura 69. Estes estereótipos são aplicáveis a classes que permitem modelar as funções do sistema lógico de automação, bem como as relações entre elas.

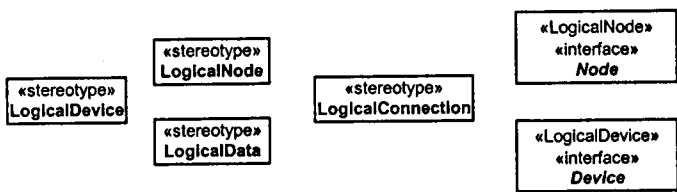


Figura 69. Estereótipos e interfaces para modelação do sistema lógico.

Uma classe ASML estereotipada com *LogicalDevice* apresenta uma semântica simples de composição de nós lógicos. Este estereótipo permite qualificar uma classe como elemento agregador de um conjunto de nós lógicos com relações funcionais fortes, vulgarmente implementado num único dispositivo.

⁴ A figura 60 apresenta um exemplo de um diagrama unifilar construído segundo este modelo.

Um nó lógico é uma classe ASML activa estereotipada com *LogicalNode*. Os nós lógicos compreendem a definição das três componentes de uma função que pode abranger um ou mais nós: (i) a componente dinâmica de uma função é modelada com recurso aos elementos comportamentais ASML associados ao conjunto de nós lógicos que a definem, (ii) a componente de interface é modelada por elementos de interacção ASML e (iii) a modelação da componente estática é realizada com recurso aos atributos e composições de nós lógicos.

O estereótipo *LogicalData* identifica classes de dados que modelam pontos de acesso a funções. Uma classe de dados pode ser simples ou compreender uma composição de outras classes de dados. Das classes de dados simples destacam-se as classes de dados primitivas que permitem modelar as bases de dados de tempo real de IEDs. Não se pretende, ao contrário de todas as restantes, que o utilizador defina classes de dados primitivas. É definido no VIII um conjunto de classes de dados primitivas, consideradas classes de dados comuns, baseado na IEC 61850^[44].

Qualquer nó lógico compreende um conjunto de atributos ou relações de composição de tipos estereotipados com *LogicalData*, os únicos constituintes de um nó lógico que podem apresentar visibilidade pública. Estes atributos estabelecem a interface do nó lógico para cooperação com os restantes nós de um modelo. As ligações lógicas entre nós são modeladas com recurso a associações simples estereotipadas com *LogicalConnection*.

As regras sintácticas para classes estereotipadas no âmbito do sistema lógico são ilustradas na figura 70.

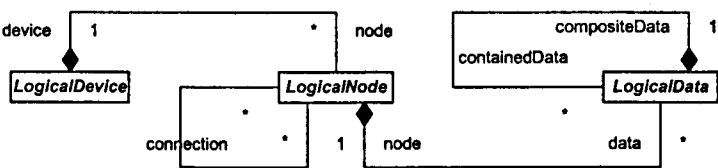


Figura 70. Regras sintácticas para modelação do sistema lógico.

4.4. Sistema Secundário

O sistema secundário inclui os estereótipos da figura 71, aos quais se associa a sintaxe da figura 72 e a semântica descrita na tabela 15. A partir de modelos com a semântica descrita é possível extrair a configuração de sistemas de automação (a arquitectura de comunicações, o número, tipo e função dos dispositivos presentes, e as associações comunicativas entre dispositivos), bem como a sua parametrização interna.

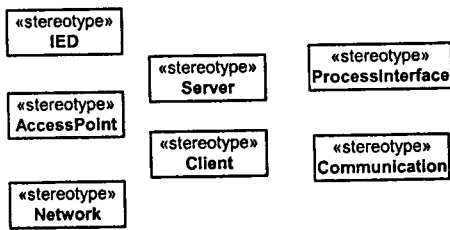


Figura 71. Estereótipos para modelação do sistema secundário.

Tabela 15. Semântica para o modelo do sistema secundário.

Estereótipo	Semântica
<i>IED</i>	Uma classe estereotipada com <i>IED</i> modela dispositivos electrónicos inteligentes. Atributos destas classes podem corresponder a parâmetros específicos do dispositivo, com uma semântica própria e dependente do dispositivo. Um IED pode ser constituído por um ou mais servidores e um ou mais clientes que representam o sistema de comunicações e a sua associação a dispositivos e ao modelo lógico. Dispositivos com interface ao processo podem apresentar relações de associação simples com os dispositivos primários aos quais estão electricamente ligados. Estas associações são estereotipadas por <i>ProcessInterface</i>. A definição de um IED deve ser sempre acompanhada de elementos de interacção de forma a permitir a definição de diagramas de interfaces comunicativas e a modelação gráfica da configuração de dispositivos.
<i>Server</i>	Classes estereotipadas por <i>Server</i> correspondem a entidades de IEDs que instanciam a interface de nós lógicos. Cada servidor é composto por um conjunto de dispositivos lógicos que estabelecem os nós lógicos atribuídos ao IED.
<i>AccessPoint</i>	Cada servidor inclui um conjunto de pontos de acesso que modelam o lado servidor das ligações comunicativas lógicas entre IEDs, efectivamente existentes no sistema de automação.
<i>Client</i>	Classes estereotipadas por <i>Client</i> compreendem entidades de IEDs que instanciam a interface remota de nós lógicos de outros IEDs. A cada cliente é associada uma ligação comunicativa com outro IED através de um dado ponto de acesso. As associações que modelam ligações comunicativas são estereotipadas com <i>Communication</i>.
<i>Network</i>	Uma rede compreende a definição de uma infra-estrutura comunicativa física que suporta as ligações comunicativas da cada ponto de acesso. As associações ente pontos de acesso e redes são estereotipadas com <i>Communication</i>.

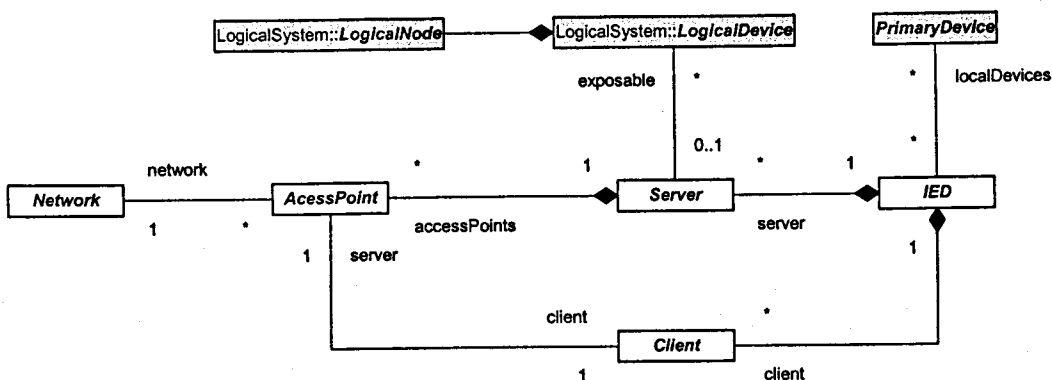


Figura 72. Sintaxe para modelação do sistema secundário.

A definição de modelos de protocolos de comunicações deve ser suportada por um conjunto de classes estereotipadas por *Server*, *AccessPoint*, *Client* e *Network*. Cada IED com capacidade comunicativa segundo um dado protocolo deve apresentar associações a este tipo de classes. O anexo IX apresenta um modelo de dois protocolos de comunicações vulgarmente utilizados na automação de sistemas de energia.

4.5. Sobre a Semântica para Automação de Sistemas de Energia

A semântica para automação de sistemas de energia compreende apenas a definição de regras gerais, não sendo âmbito deste trabalho a definição exaustiva de uma biblioteca de elementos para automação de instalações. O objectivo da introdução deste meta-modelo é desenvolver uma solução para suportar componentes de uma ferramenta associados a este domínio de aplicação, deixando a definição de modelos concretos para trabalho futuro. Na IEC 61850 [42] [43] [44] [45], bem como na IEC 61400-25 [7], são apresentados modelos de elementos considerados referência para os sistemas actuais, que se podem enquadrar na ASML. A configuração e parametrização de um sistema de automação específico correspondem à instanciação, definida por modelos de objectos, de classes ASML estereotipadas segundo o modelo semântico apresentado.

Numa perspectiva de síntese é relevante sistematizar a forma como a ASML e a semântica para automação de sistemas de energia suportam o desenho de sistemas de automação. Uma síntese da aplicação da linguagem é apresentada na tabela 16.

Tabela 16. Rastreio de requisitos da ASML para o desenho de sistemas de automação.

Elemento do Modelo de Sistemas de Automação	Suporte
Sistema lógico	Estes modelos são suportados pela ASML e a semântica para automação do sistema eléctrico.
Sistema Físico Primário	
Sistema Físico Secundário	
Relações entre o sistema primário, secundário e lógico	As relações entre os vários modelos são modeladas com recurso a associações simples.
Parametrização	A parametrização estática do sistema de automação corresponde aos modelos definidos segundo a semântica para automação de sistemas de energia. O conjunto de parâmetros estáticos de cada IED é obtido de forma directa de cada instância de dispositivo físico do modelo, incluindo eventualmente também alguns elementos modelados sobre atributos de nós lógicos. A modelação da parametrização dinâmica, ou seja, dos parâmetros alteráveis em tempo de execução do sistema, é abordada em atributos específicos de nós lógicos. Os parâmetros de sistema e processo correspondem essencialmente à parametrização estática, ao invés dos parâmetros funcionais que podem ser estáticos ou dinâmicos. Todos os parâmetros funcionais correspondem a atributos de nós lógicos. A organização em níveis hierárquicos dos parâmetros do sistema, bem como de outros elementos seus constituintes, é mapeada directamente da hierarquia de pacotes ASML dos modelos definidos pelo engenheiro de automação.

Capítulo 5

Conclusões

Neste último capítulo da segunda parte da dissertação é apresentada uma avaliação do potencial da ASML para aplicação no âmbito da automação de sistemas de energia, bem como orientações para evolução futura dos modelos propostos.

5.1. A ASML e os Sistemas de Automação

O desenvolvimento de ferramentas de engenharia segundo a abordagem da ASML apresenta outras vantagens, para além da promoção da eficiência da actividade de engenharia de automação.

A ASML é uma linguagem genericamente aplicável a sistemas de eventos discretos, não compreendendo a sua definição qualquer produção directamente aplicável a sistemas de automação. Este facto constitui uma vantagem na perspectiva das ferramentas de engenharia, uma vez que uma ferramenta capaz de manipular modelos definidos segundo a ASML pode ser aplicada a vários sistemas discretos, dos quais os sistemas de automação de instalações do sistema eléctrico são um caso de aplicação particular. A introdução de um modelo especialmente vocacionado para o desenho destes sistemas vem colmatar a generalidade da linguagem, tomando-a directamente aplicável ao desenho conduzido por modelos no domínio de aplicação deste trabalho.

Tendo como base o meta-modelo orientado ao objecto da ASML, o núcleo central da ferramenta de engenharia pode manter-se actualizado face a alterações dos modelos do sistema de automação de instalações, bem como ser aplicável a outras áreas da automação do sistema eléctrico não tratadas directamente neste trabalho, como a protecção de grande área, ou mesmo a outros domínios da engenharia. Esta orientação permite também que a ferramenta seja aplicada a uma grande variedade de IEDs, suplantando os seus ciclos de vida. Estes factores são elementos de justificação económica da abordagem proposta.

5.2. Rastreio de Requisitos Funcionais

A tabela 17 explicita o suporte fornecido pela ASML a uma ferramenta de engenharia para o cumprimento dos requisitos funcionais estabelecidos na primeira parte da dissertação.

Tabela 17. Rastreio de requisitos da ASML.

Requisito		Suporte
Modelação	Criação e edição de modelos	A ASML é, no essencial, um meta-modelo cujas instâncias são modelos. Este requisito é abrangido pela linguagem de forma directa. Os modelos criados suportam o desenho das várias vertentes do sistema de automação. O desenho é realizado de forma gráfica, recorrendo à notação base da linguagem ou a notações alternativas definidas no próprio modelo, como diagramas unifilares, de ligações comunicativas ou de configuração de IEDs.
	Gestão de versões de modelos	Os elementos de gestão de modelos permitem a sua estruturação interna, a incorporação e reutilização de múltiplos modelos na definição de um sistema, bem como elementos para identificação e versionamento de modelos ou de partes de modelos.
	Exportação e importação de modelos	A definição não ambígua da sintaxe abstracta permite definir, no âmbito do desenvolvimento de uma ferramenta, uma API de acesso a modelos que suporta a existência de componentes de importação/exportação. A título de exemplo, a conversão de modelos para a linguagem de configuração de subestações (SCL) proposta na IEC 61850 pode ser concretizada de forma facilitada.
Validação	Estática	Uma ferramenta de manipulação da linguagem, pode incluir componentes de validação estática, nomeadamente suportando a validação de um modelo segundo as regras semânticas definidas no segundo e quarto capítulos desta parte da dissertação. A validação pode incluir a determinação de erros semânticos ao nível da ASML, como sejam a validação de classes ou de relações entre classes, ou ao nível das características semânticas introduzidas pela arquitectura para sistemas de automação. A validação sintáctica, por ser essencial à interface entre o engenheiro de automação e a ferramenta, será necessariamente parte integrante da ferramenta.
	De comportamento	Dado que a definição de comportamento é, na ASML, conseguida através de um meta-modelo concreto não ambíguo, a simulação pode ser realizada com recurso a componentes específicos da ferramenta. A interface para visualização do estado do sistema durante uma simulação pode recorrer directamente aos diagramas animados incluídos no modelo.
	Monitorização	Este requisito não é directamente suportado pela linguagem uma vez que não é âmbito da sua definição. No entanto a interface ao utilizador de um componente da ferramenta para ensaio de IEDs pode ser coincidente com os modelos de interacção criados pelo engenheiro de automação.

Tabela 17. Rastreio de requisitos da ASML (continuação).

Requisito		Suporte
Parametrização	Transformação de modelos	A definição de sintaxe abstracta sem ambiguidade permite definir, no âmbito do desenvolvimento de uma ferramenta, uma API de acesso a modelos que suporta a existência de componentes de transformação de modelos ASML para modelos específicos de cada IED e vice-versa.
	Gestão de versões de parametrização	Este requisito não é directamente suportado pela linguagem, exceptuando a sua relação com os mecanismos de identificação de versões de modelos.
	Transferência de parametrização	A recuperação de parametrização ou a alteração de parâmetros em IEDs não é âmbito da definição da linguagem.
Documentação	Geração de elementos documentais	A produção de esquemas gráficos pode ser conseguida através da exportação de diagramas, dado que a linguagem apresenta uma notação gráfica, bem como a possibilidade de definição de sinópticos e notações gráficas alternativas. Outros elementos documentais como listas podem ser gerados a partir de um modelo, com recurso a componentes específicos da ferramenta.

5.3. Evolução Futura da ASML

Numa análise da ASML pode constatar-se que esta apresenta, por um lado, elementos passíveis de melhoria e que, por outro, alguns elementos não foram equacionados no seu desenvolvimento. Estes elementos, enumerados na tabela 18, poderão ser alvo de trabalho futuro.

Tabela 18. Elementos da ASML passíveis de trabalho futuro.

Elemento	Descrição
Linguagem para restrições semânticas de nível M2.	A ASML não inclui uma linguagem de nível M2, como a OCL, para definição de restrições semânticas. A introdução de uma linguagem deste tipo permitiria a definição de semânticas de domínio concretas, passíveis de validação automática pela ferramenta de engenharia. A título de exemplo, a arquitectura para sistemas de automação poderia ser definida recorrendo a uma linguagem deste tipo, o que eliminaria por completo a necessidade de desenvolvimento e manutenção de componentes de validação de modelos desse domínio. Outras características interessantes poderiam também surgir da inclusão de uma linguagem deste tipo, nomeadamente a possibilidade de criação de notações alternativas animadas e interactivas em tempo de execução da ferramenta.
Transformação de modelos	A definição de mecanismos de transformação de modelos ASML para e a partir de outro tipo de modelos não foi abordada neste trabalho. A introdução de uma forma genérica de realização deste tipo de operações seria interessante para o desenvolvimento rápido de componentes de importação, exportação para outras ferramentas ou para modelos dependentes de dispositivos.

Tabela 18. Elementos da ASML passíveis de trabalho futuro (continuação).

Elemento	Descrição
Outros elementos do sistema de automação	Alguns elementos presentes no sistema de automação não foram alvo de modelação detalhada, nomeadamente: gestão de utilizadores e gestão de acesso, gestão de alarmes e arquivos, visualização em gráficos de tendência, transferência de ficheiros ou manipulação de perturbografia.
Relação com outras vertentes do projecto de sistemas de automação	Algumas vertentes do projecto de sistemas de automação não foram abordadas no trabalho de análise e caracterização, nomeadamente a gestão de projectos ou o desenho de esquemas para electrificação de armários e sistemas. A integração deste tipo de funcionalidades ou a relação de uma ferramenta de engenharia com ferramentas de apoio a estas actividades podem também ser alvo de estudo.
Biblioteca de elementos para sistemas de automação	Uma vez que a semântica para automação de sistemas de energia engloba apenas princípios elementares, a afinação e validação da ASML sobre um sistema real, bem como a criação de uma biblioteca de elementos para sistemas de automação, é uma orientação possível para trabalho futuro.

5.4. Síntese

A ASML pode ser aplicada no projecto de sistemas de automação de dimensões variadas, na definição de modelos de domínio, dos quais se pode extrair quer a configuração quer a parametrização de sistemas e dispositivos, e no apoio à geração de documentação. Embora a definição da ASML e da semântica para sistemas de automação não inclua todas as características necessárias para apoio completo ao projecto de sistemas de automação de terceira geração, as características essenciais dos sistemas de automação são endereçadas e os elementos fundamentais da linguagem permitem estender o modelo proposto de forma a abranger as características ausentes. Conclui-se assim que a ASML deve ser o núcleo da ferramenta de engenharia, uma vez que permite dotá-la de características que suportam um investimento de longo prazo para utilização alargada.

Parte III

Estratégias de Implementação

Sobre a Implementação da Ferramenta

Proposta de arquitectura e de orientações de implementação

IDE, repositório e editor gráfico

Experiências de implementação

Capítulo 1

Sobre a Implementação da Ferramenta

A Linguagem de Modelação para Sistemas de Automação (ASML) permite descrever modelos para automação de instalações do sistema eléctrico. É proposta neste capítulo uma solução arquitectural com vista à implementação de uma ferramenta que permita a construção, armazenamento e transformação de modelos ASML. São descritas também implementações experimentais de componentes da ferramenta. A proposta apresentada neste capítulo não aborda a totalidade dos requisitos apontados para uma ferramenta de engenharia, não sendo equacionada, por exemplo, a implementação de componentes de validação estática ou dinâmica.

As orientações de implementação aqui propostas resultam de (i) uma análise dos paradigmas de interface ao utilizador de ferramentas de engenharia actuais para suporte ao desenvolvimento de sistemas de informação, (ii) de experiências realizadas de utilização da plataforma NetBeans ^[49] e (iii) de uma análise das propostas OMG deste domínio, nomeadamente as UML ^[29], MOF ^[50] e XMI ^[51].

1.1. Princípios Arquitecturais

O desenvolvimento da ferramenta de engenharia deve seguir o princípio arquitectural de três camadas ilustrado na figura 73. A camada inferior oferece exclusivamente suporte para dados, sendo o repositório de modelos um exemplo de um componente associado essencialmente a esta camada. A camada intermédia oferece serviços de manipulação e processamento de dados. Um componente de validação estática de modelos é um exemplo de um componente essencialmente associado à camada intermédia. A camada superior fornece a lógica de interface ao utilizador. Todas as funcionalidades da ferramenta devem ser desenvolvidas tendo em consideração este princípio arquitectural, embora casos particulares possam omitir, por exemplo, a camada intermédia. A aplicação deste princípio permite garantir uma estruturação de nível geral.

Dada a dimensão de uma ferramenta de engenharia, a sua implementação deverá ter como base uma arquitectura modular. A arquitectura de implementação modular proposta, baseada na arquitectura do NetBeans ^[49] apresentada na figura 74, posiciona-se de forma ortogonal face ao princípio de três camadas. Esta arquitectura de implementação sugere que cada macro-funcionalidade seja implementada por um conjunto de módulos cooperantes. Um módulo cooperante consiste num macro-componente com funcionalidades e interface pública bem delimitados. Um

componente particular desta arquitectura consiste no componente de gestão de módulos, denominado núcleo, cujo propósito inclui a gestão de versões e dependências de módulos, o arranque e a terminação da ferramenta.

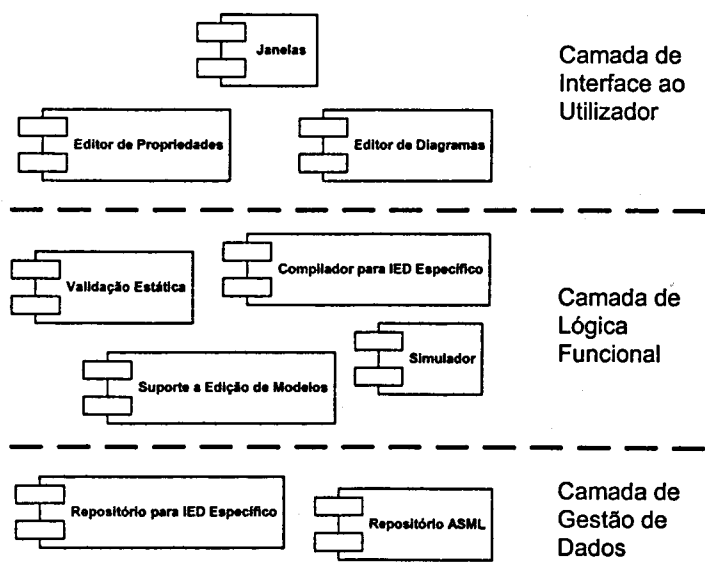


Figura 73. Arquitectura de três camadas.

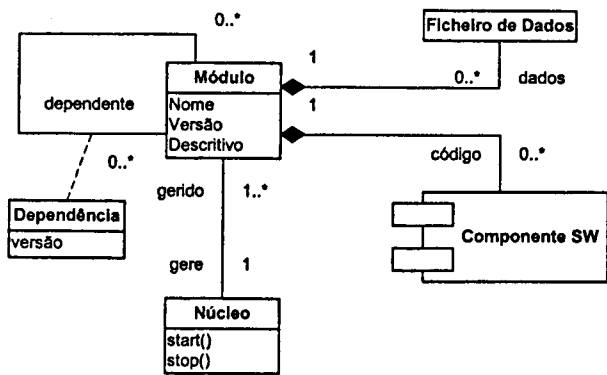


Figura 74. Arquitectura modular para implementação de ferramentas de engenharia.

Um módulo consiste não só num conjunto agregado de componentes de software executável, mas pode incluir também dados, como sejam modelos ASML ou ficheiros de configuração. Cada módulo deverá ser implementado à custa da integração de componentes de *software* variados, incluindo componentes implementados especificamente ou componentes COS, como bibliotecas de suporte.

1.2. Interface ao Utilizador

A interface ao utilizador da ferramenta de engenharia deve ser construída sobre o paradigma do ambiente de desenvolvimento integrado (IDE). Em exemplo de um ambiente IDE é apresentado na figura 75.

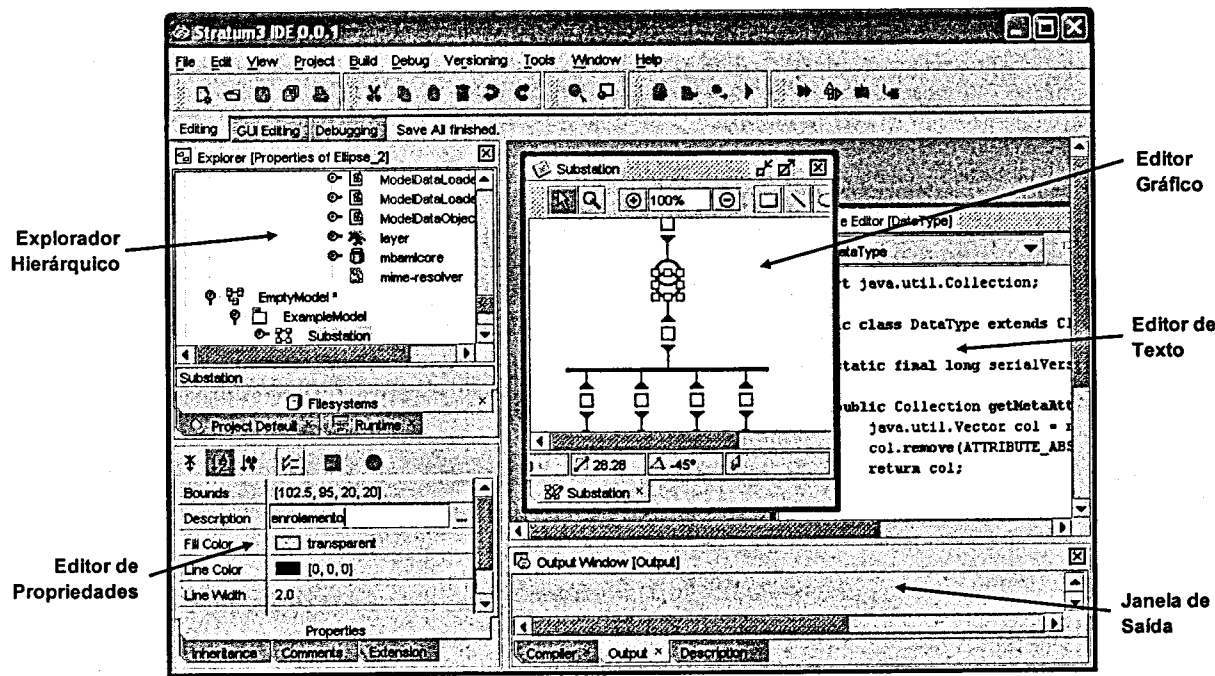


Figura 75. Exemplo de um ambiente IDE.

Este paradigma é uma variante do ambiente multi-janela que inclui essencialmente dois tipos de janelas, a janela de documento e a janela de suporte. As janelas de documento são as janelas convencionais, cujo objectivo é apresentar vistas complexas de um dado objecto. Destas destacam-se o editor de texto, o editor gráfico, o editor tabular ou outros. As janelas de suporte são agrupadas em painéis atracáveis, usualmente sempre visíveis independentemente da janela de documento activa. As janelas de suporte compreendem o editor de propriedades, o explorador hierárquico, a janela de saída de texto ou outras janelas com funcionalidades dedicadas. As janelas de suporte modificam vulgarmente o seu conteúdo de forma automática em resposta ao objecto em edição pelo utilizador. O editor de propriedades, por exemplo, apresenta sempre a lista de propriedades do objecto seleccionado pelo utilizador na janela activa. Outros elementos de interface ao utilizador vulgares como barras de ferramentas, menus ou caixas de diálogo são naturalmente também incluídos. Este modelo de interface ao utilizador é apresentado, de forma resumida, no diagrama da figura 76, podendo uma sua descrição aplicada ao ambiente NetBeans ser obtida em [49]. A implementação deste ambiente pode ser realizada com recurso a integração e especialização de componentes COS para interface ao utilizador, vulgarmente disponíveis nas plataformas aplicacionais como o Java Swing [52], a MFC [53] ou outros.

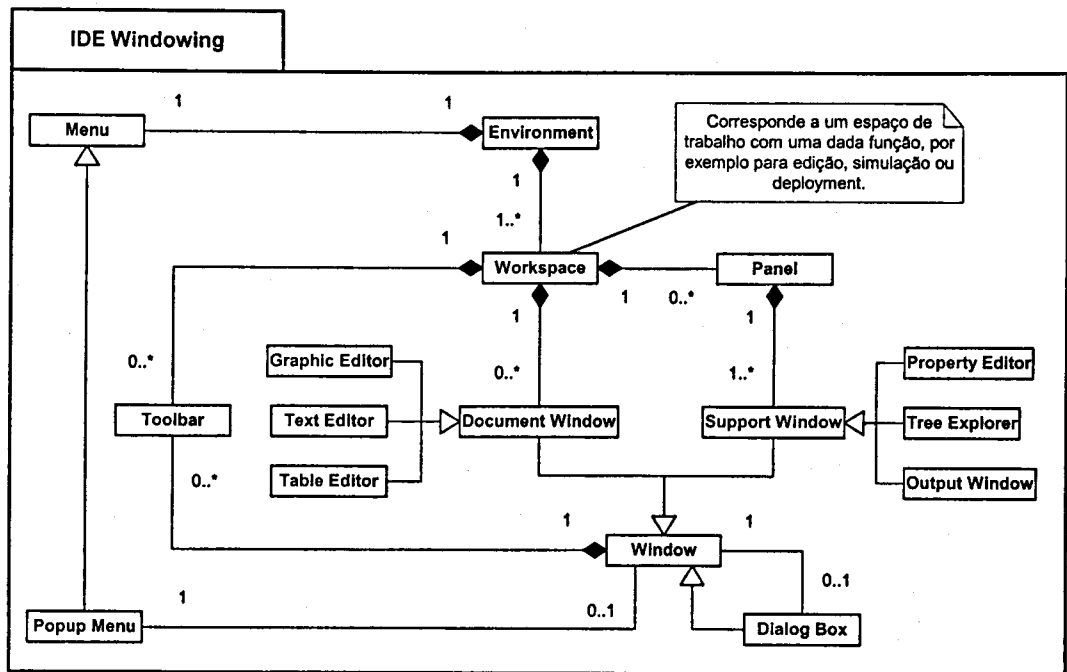


Figura 76. Modelo para o ambiente de interface ao utilizador de um IDE.

1.2.1. Aplicação do Padrão MVC à Interface do IDE

O explorador hierárquico e o editor de propriedades são componentes de interface naturais do IDE, uma vez que a maior parte dos modelos de dados associados aos sistemas de automação podem ser vistos como uma estrutura em árvore cujos nós apresentam atributos. O meta-modelo ASML, por exemplo, pode ser manipulado dessa forma. Dado que este conceito é intuitivo e vulgar torna-se interessante introduzir uma arquitectura para suportar a apresentação de objectos segundo este modelo. Uma arquitectura MVC baseada naquela descrita em [49] é apresentada na figura 77.

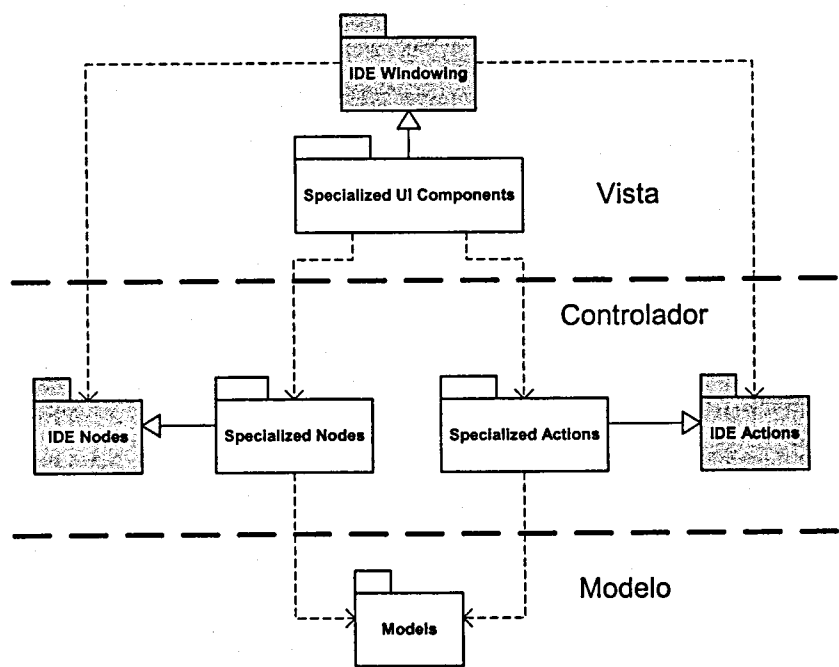


Figura 77. Arquitectura MVC aplicada à interface ao utilizador de um IDE.

As vistas de uma dada funcionalidade da ferramenta serão implementadas com recurso a janelas de documento, janelas de suporte ou outros elementos de interface dedicados. O editor de propriedades e explorador hierárquico corresponderão à vista por defeito, embora outras vistas de carácter genérico possam ser utilizadas como por exemplo uma vista tabular ou baseada em menus e barras de ferramentas.

O controlador corresponde ao modelo de nós e acções apresentado na figura 78, que deverá ser especializado na implementação de cada funcionalidade. Este modelo abstrai todos os objectos manipulados no IDE (nós) e as acções executadas sobre estes, permitindo assim a implementação de componentes de interface ao utilizador independente da lógica de cada funcionalidade específica da ferramenta de engenharia.

O modelo, nesta arquitectura MVC, inclui a implementação da lógica de uma dada funcionalidade, não sendo por isso considerado parte da camada de interface da ferramenta, mas sim da camada de lógica funcional ou da camada de dados.

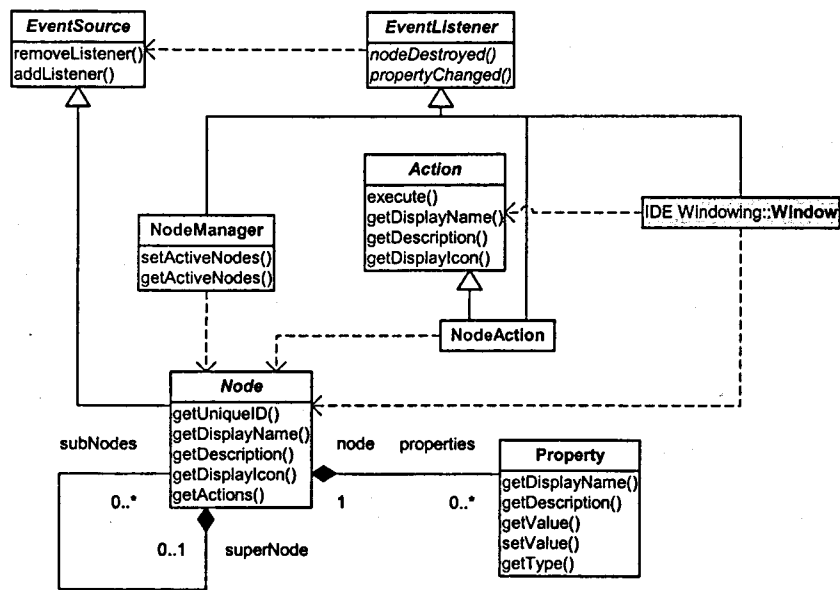


Figura 78. Modelo simplificado de nós e ações para implementação do controlador da arquitectura MVC.

1.2.2. Editor Gráfico

Dado que a ASML é uma linguagem essencialmente gráfica, a existência de um editor diagramático gráfico é fundamental. O editor gráfico é um componente associado à camada de interface da ferramenta que se pretende integrado com o modelo de nós. Propõe-se o modelo de editor gráfico apresentado na figura 79, construído também sobre o padrão MVC.

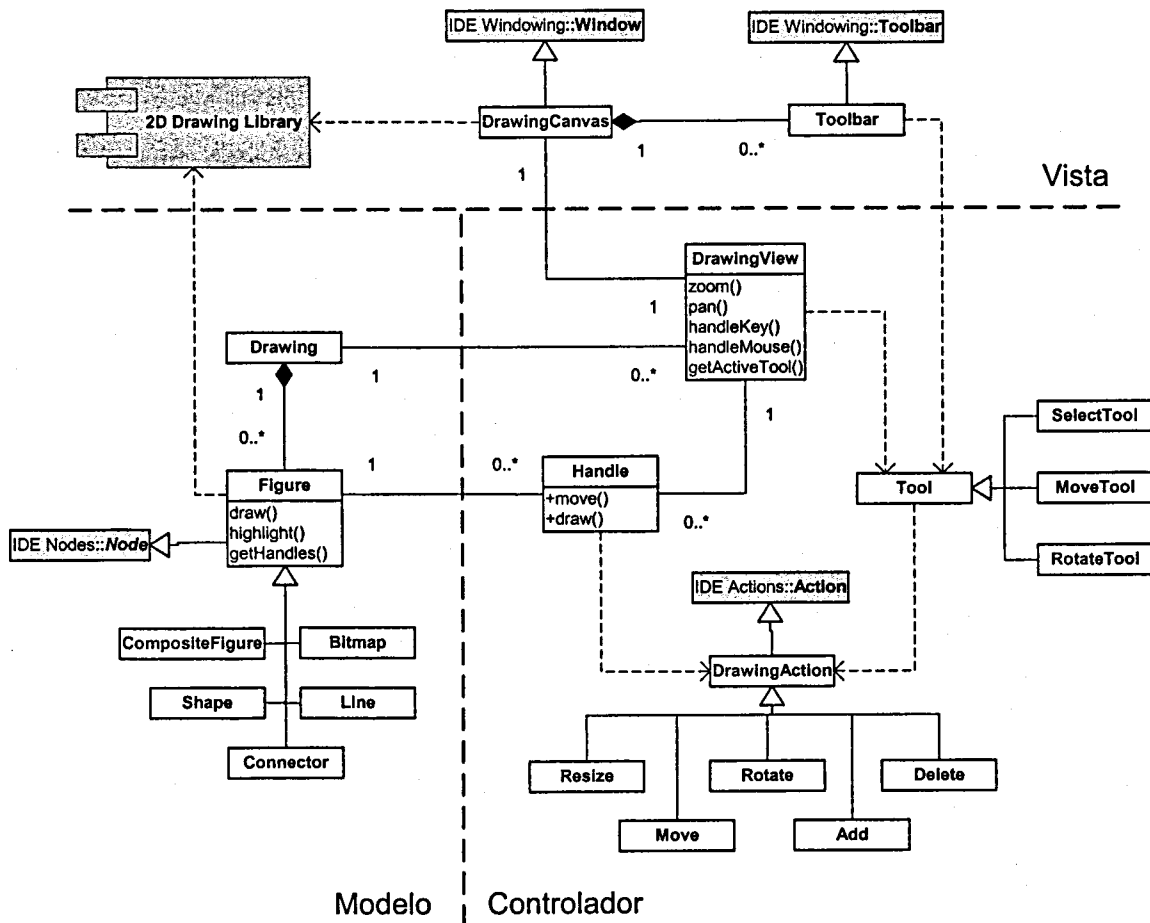


Figura 79. Modelo geral do editor gráfico.

O modelo de dados do editor corresponde ao modelo habitual de nós gráficos e ligações, semelhante ao modelo de elementos de interacção da ASML. O controlador compreende as funcionalidades de edição gráfica comuns, como as ferramentas de selecção, criação, redimensionamento e reposicionamento de objectos gráficos. As vistas deverão ser implementadas com recurso a bibliotecas de desenho bidimensional como a GDI+ ^[54] ou a Java 2D ^[55].

1.3. Repositório de Modelos

O repositório ASML é o componente fulcral da ferramenta de engenharia que permite o armazenamento de modelos durante execução da ferramenta, bem como a sua persistência. O modelo de repositório proposto compreende estruturas de dados para armazenamento em memória volátil, uma API reflectiva para acesso directo a modelos do repositório, um modelo de eventos para notificação de alterações e uma camada de gestão de persistência. Na figura 80 ilustra-se o modelo de camadas do repositório.

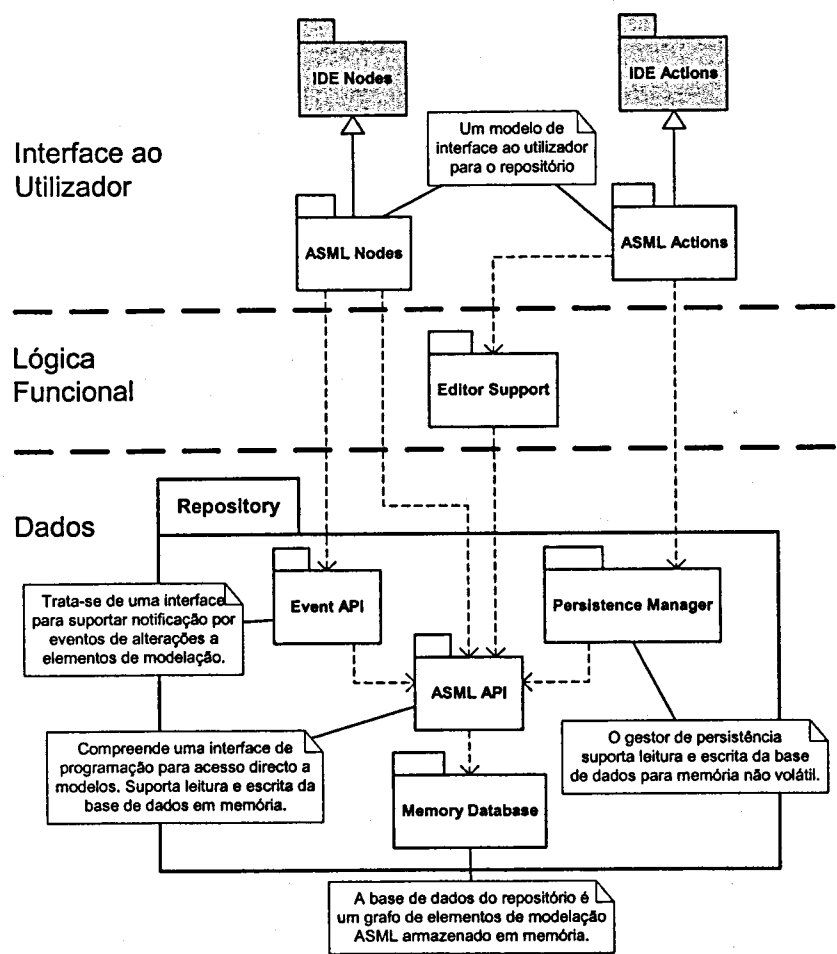


Figura 80. Modelo do repositório e relações com componentes das camadas funcional e de interface.

O modelo da base de dados do repositório corresponde à instância definida pela ASML de um meta-meta-modelo que pode ser visto como uma simplificação do MOF. Este meta-meta-modelo, apresentado de forma simplificada na figura 81, pode ser implementado de forma fácil recorrendo a uma linguagem orientada ao objecto com funcionalidades de introspecção.

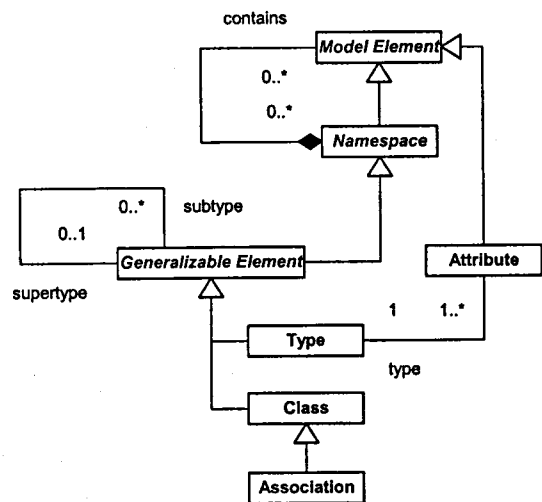


Figura 81. Meta-meta-modelo para o repositório.

A API de acesso a modelos corresponde à realização do meta-meta-modelo da figura 81 numa dada linguagem de programação e a sua posterior aplicação na definição do meta-modelo ASML. As características reflectivas desta API permitem o desenvolvimento simplificado não só do modelo de nós para edição de modelos ASML no IDE, da camada de gestão de persistência de modelos, mas também de outros componentes, como por exemplo componentes de transformação de modelos. O modelo do repositório é implementado de forma independente do componente de gestão de persistência. Este componente pode ser implementado de forma a permitir a persistência de modelos em múltiplos suportes desde ficheiros binários, bases de dados relacionais ou ficheiros XML, tomando eventualmente como exemplo a gramática XML.

Sobre a API do repositório deverá ser implementada uma camada de suporte a editores. Esta camada funcional apresentaria uma API baseada em acções com suporte para desfazer e refazer acções executadas, mas que deverá compreender uma funcionalidade essencial: a garantia de consistência de modelos ASML durante a sua alteração. Esta funcionalidade implica o conhecimento integral implícito da semântica ASML e garante, por exemplo, que se o tipo de um atributo de uma classe é alterado, então todos os objectos dessa classe são também actualizados.

1.3.1. Interface ao Repositório e Suporte a Editores

O modelo geral da API de acesso a modelos é apresentado na figura 82, sendo a sua relação com o meta-meta-modelo ilustrada na tabela 19. O elemento fundamental desta API é a classe *Element*, que deverá ser especializada sucessivamente segundo o meta-modelo da ASML apresentado na segunda parte da dissertação. A cada *DataType* ASML deverá corresponder uma sub-classe de *Type*, a cada elemento de modelação uma sub-classe de *ModelElement* e a cada associação entre elementos de modelação ASML uma sub-classe de *MetaAssociation*. Nesta API integra-se também o modelo de eventos para notificação de alterações a modelos, suportado pelas classes *Element* e *ElementListener*.

Tabela 19. A API de acesso a modelos e a sua relação com o meta-meta-modelo.

Meta-meta-classe	Significado e implementação na API
<i>ModelElement</i>	Esta meta-meta-classe corresponde a uma abstracção de todas as meta-meta-classes. A sua implementação é implícita na classe <i>ModelElement</i> da API.
<i>Namespace</i>	Esta meta-meta-classe bem como a sua associação de composição com a meta-meta-classe <i>ModelElement</i> possibilitam a definição de meta-elementos compostos e são realizadas na classe <i>ModelElement</i> da API. Os métodos <i>getOwnedElements</i> , <i>addOwnedElement</i> e <i>removeOwnedElement</i> suportam o acesso reflectivo a esta característica.
<i>Generalizable Element</i>	As relações de generalização entre meta-classes são suportadas pelas características da linguagem de programação (linguagem orientada ao objecto).
<i>Type</i>	Os tipos de dados a que se refere a meta-meta-classe <i>Type</i> correspondem aos tipos de dados utilizados na definição da ASML, aí generalizados pela meta-classe <i>DataType</i> (figura 26 da segunda parte). A implementação de tipos de dados deverá ser realizada através de classes da linguagem de programação derivadas da classe <i>Type</i> da API.
<i>Attribute</i>	As instâncias desta meta-meta-classe correspondem aos atributos dos meta-elementos da ASML e são implementados em especializações da classe <i>ModelElement</i> da API através da implementação dos métodos <i>getAttributes</i> , <i>setAttributeValue</i> e <i>getAttributeValue</i> .
<i>Class</i>	As classes <i>ModelElement</i> e <i>MetaAssociation</i> da API.
<i>Association</i>	Esta meta-meta-classe modela as meta-associações entre meta-elementos ASML e é implementada na classe <i>MetaAssociation</i> da API. A classe <i>ModelElement</i> , através do método <i>getAssociations</i> permite obter as meta-associações às quais o meta-elemento está associado, garantindo assim o acesso reflectivo a meta-associações.

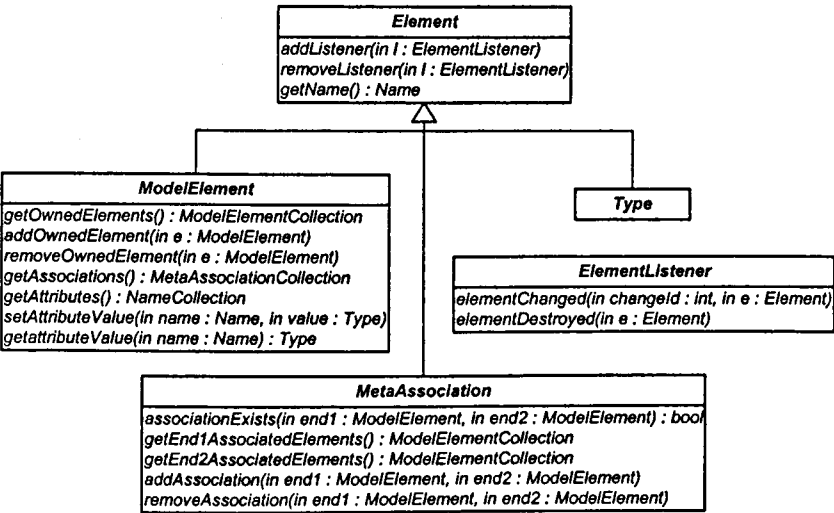


Figura 82. Interface da API de acesso a modelos.

A camada funcional para suporte a editores (figura 83) apresenta uma API baseada em acções cujo resultado implica alterações a modelos ASML do repositório. Esta API inclui suporte para desfazer e refazer acções executadas, bem como a garantia de consistência de modelos ASML.

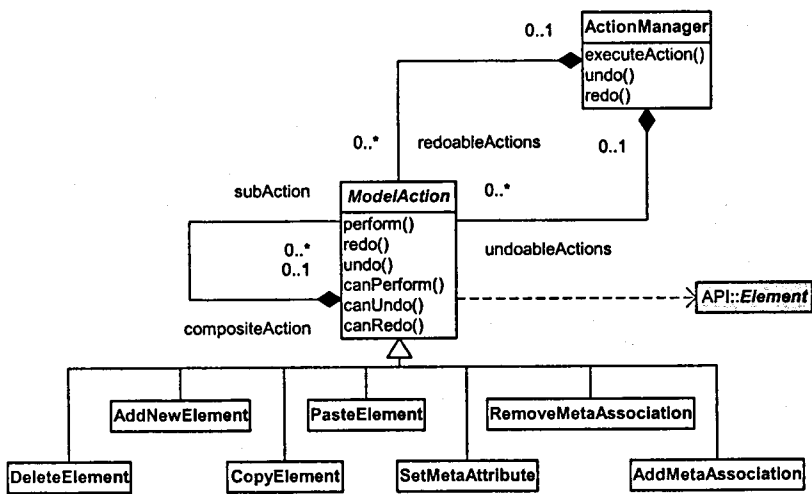


Figura 83. Modelo para a camada de suporte a editores.

1.4. Experiências de Implementação Conduzidas no Âmbito do Trabalho de Tese

Com o objectivo de ilustrar a aplicabilidade das propostas incluídas nesta dissertação foram conduzidas experiências de implementação essencialmente centradas nos princípios arquitecturais para implementação descritos.

1.4.1. O Protótipo Stratum3

O protótipo Stratum3 é uma experiência de implementação de dois componentes da ferramenta de engenharia, o repositório e o editor gráfico, sobre a plataforma para IDEs NetBeans. Uma imagem da interface ao utilizador resultante é apresentada na figura 75. O NetBeans é uma plataforma Java de código livre para IDEs que suportará um modelo semelhante ao apresentado no diagrama da figura 77. Sobre o Netbeans foi implementado um repositório segundo o modelo da figura 80, que inclui suporte para um subconjunto do modelo ASML. Foram também implementados componentes, baseados no modelo de nós do NetBeans, para suportar uma interface de edição de modelos não gráfica. Sobre o NetBeans foi implementado na íntegra um motor para edição gráfica suportado por Java Swing e Java 2D, adoptando o modelo da figura 79. Esta implementação experimental permite antever uma ferramenta de engenharia implementada de acordo com as propostas incluídas nesta dissertação.

1.4.2. Plataforma para IDEs

Como experiência inicial adicional foi desenvolvida uma plataforma experimental para IDEs (figura 84) em C++ para Win32 suportada pela biblioteca MFC (implementação da interface ao utilizador) e na biblioteca XercesC ^[63] como gerador e processador de XML (suporte para persistência do ambiente de janelas bem como de parâmetros do IDE).

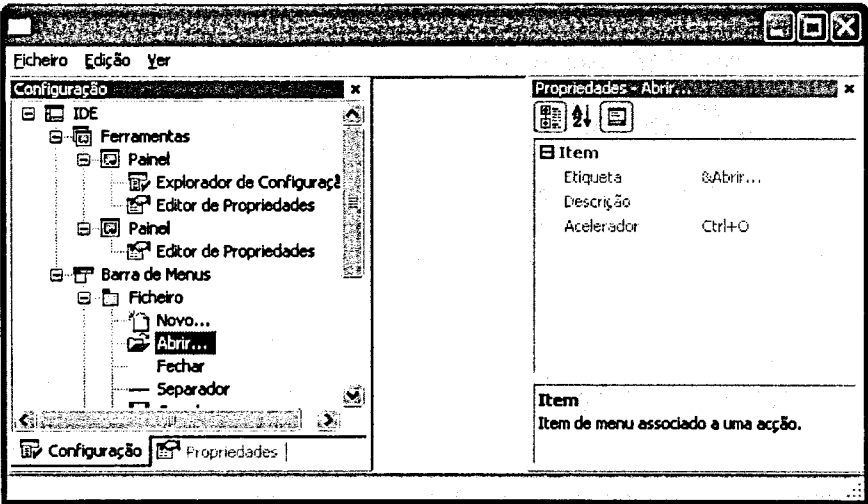


Figura 84. Vista da plataforma para IDEs implementada em C++.

A indisponibilidade de funcionalidades de introspecção adequadas em C++ implicou o desenvolvimento prévio de uma *framework* introspectiva com persistência XML de objectos, interfaceada através de um conjunto de classes e macros.

1.5. Conclusões

Demonstrou-se neste capítulo ser possível a implementação de uma ferramenta de engenharia para suporte ao projecto de sistemas de automação, de acordo com os requisitos apresentados na primeira parte da dissertação e de acordo com a linguagem ASML. A ferramenta pode ser implementada de forma integral, segundo os modelos e princípios arquitecturais apresentados, ou através da agregação de componentes de código livre, como a plataforma NetBeans ou a plataforma Eclipse ^[56]. O desenvolvimento do editor gráfico pode também ser acelerado pela integração de componentes como o JHotDraw ^[57] ou o GEF ^[58]. Quanto ao repositório de modelos existe hoje também suporte de código livre, nomeadamente o suporte integrado nas plataformas referidas ^[59] ^[60] ^[61], que tem vindo a ser desenvolvido durante o decurso deste trabalho de tese.

Parte IV

Conclusões

Sumário, Resultados e Considerações Finais

Sumário, resultados e conclusões do trabalho

Considerações finais

Orientações para trabalho futuro

Capítulo 1

Sumário, Resultados e Considerações Finais

Este trabalho teve como objectivo o estabelecimento de orientações para implementação de ferramentas de engenharia, tendo em vista o apoio ao projecto de sistemas de automação no âmbito particular da automação de instalações do sistema eléctrico de energia. Durante a análise inicial foram caracterizados os sistemas de automação e perspectivadas tendências de evolução. Desta análise concluiu-se que uma ferramenta deveria incidir particularmente sobre o desenho do sistema de automação e abranger um espectro suficientemente lato de actividades de projecto neste domínio de engenharia. A introdução do desenvolvimento conduzido por modelos nos vários sistemas de informação, bem como o aparecimento de modelos de domínio ao nível das actividades de normalização internacional, suportaram a opção da introdução destes conceitos de desenho também no projecto de sistemas de automação. Após a análise inicial foram estabelecidos os requisitos para a ferramenta de engenharia.

O trabalho prosseguiu de acordo com esta orientação, tendo como objectivo o estabelecimento de uma linguagem apropriada para o desenho do sistema caracterizado, considerada o elemento fundamental da ferramenta. Do estudo das linguagens comumente aplicadas na prática actual foi considerada solução mais interessante a adopção parcial de elementos de várias linguagens, como a IEC 61131-3 para definição de procedimentos e condições, as redes de Petri para definição de comportamento discreto, e uma linguagem para definição de interface que inclui elementos semelhantes ao modelo subjacente à gramática SVG. Estes elementos são integrados de forma coesa, segundo conceitos de orientação ao objecto, pela inclusão de elementos estruturais, de gestão e de extensão da linguagem UML.

O meta-modelo da linguagem denominada ASML foi definido na sua totalidade. Esta definição não formal inclui a sintaxe abstracta e a semântica, bem como um léxico e sintaxe gráfica e textual concretos. A definição do meta-modelo da linguagem seguiu orientação idêntica àquela segundo a qual a UML foi definida, de acordo com um modelo de níveis para meta-modelos. Deste trabalho resulta um modelo de uma linguagem baseado num meta-meta-modelo semelhante a um subconjunto do MOF e em gramáticas BNF. Este facto, bem como alguns cuidados na modelação da linguagem, permite a implementação económica de um repositório e uma API para acesso e edição de modelos ASML.

Outra preocupação presente na definição do meta-modelo foi conseguir que a linguagem seja suficientemente concreta para não apresentar ambiguidade, permitindo assim, por um lado, que os modelos ASML definidos possam ser facilmente transformáveis ou simuláveis e, por outro, possibilitando a implementação económica da ferramenta. O meta-modelo resultante é também suficientemente genérico para ser aplicável na modelação de uma multiplicidade de sistemas,

dentro do domínio dos sistemas de eventos discretos. A linguagem é desta forma considerada um compromisso interessante entre a abrangência da sua aplicabilidade a sistemas e dispositivos, a agilidade face à evolução dos modelos de domínio e a viabilidade do seu desenvolvimento.

Dada a abrangência da ASML foi também definida uma semântica de domínio concreto para automação do sistema eléctrico. Esta semântica está enquadrada nos modelos de domínio actuais e apresenta regras para a definição de modelos do sistema primário, do sistema secundário e das funções dos sistemas de automação, sejam estas comunicativas, de interface ou de controlo. Alguns exemplos de modelação foram incluídos na dissertação para exemplificar a aplicação da ASML. Diferentes semânticas poderão vir a ser introduzidas para suportar outros domínios de aplicação.

Em paralelo com o estabelecimento da ASML foram conduzidas experiências de implementação, com o objectivo de verificar opções de desenvolvimento da ferramenta de engenharia. Destas experiências destaca-se (i) um ensaio de implementação para Win32 de uma plataforma de IDE em linguagem C++ e (ii) a implementação Java de um repositório e editor não gráfico sobre a plataforma NetBeans, bem como a implementação de um motor de edição gráfica sobre Java Swing e Java 2D.

1.1. Sumário de Resultados

A tabela 20 apresenta um sumário dos resultados mais relevantes do trabalho desenvolvido no âmbito da presente dissertação.

Tabela 20. Sumário de resultados do trabalho.

Resultado	Descrição
Caracterização dos sistemas de automação	Modelo geral dos sistemas de informação da <i>utility</i> , considerações sobre tendências de evolução actuais, modelo de classificação funcional dos sistemas de automação, descrição de funções de automação típicas e de dispositivos de automação comuns.
Modelo para sistemas de automação de instalações	Modelo geral para sistemas de automação, vocacionado para ferramentas de engenharia, que integra as perspectivas física, funcional lógica, de parametrização e de projecto.
Requisitos para ferramentas	Requisitos genéricos e modelo de requisitos funcionais para ferramentas de engenharia.
Linguagem para modelação de sistemas de automação	Meta-modelo integral de linguagem de modelação aplicável à descrição de sistemas de eventos discretos, com génese na aplicação à automação do sistema eléctrico.
Semântica de domínio para automação de instalações	Modelo semântico aplicado à linguagem definida, com vista à aplicação directa no desenho de sistemas de automação de instalações do sistema eléctrico de energia.

Tabela 20. Sumário de resultados do trabalho (continuação).

Resultado	Descrição
Solução de implementação	Arquitectura e modelos de implementação de alguns componentes de uma ferramenta de engenharia.
Experiências de implementação	Implementação experimental de alguns dos componentes propostos para a ferramenta. Implementação sobre a plataforma Windows, utilizando a linguagem C++, e a plataforma Java, com recurso a componentes de código livre.

1.2. Conclusões

Os sistemas de automação de instalações do sistema eléctrico foram caracterizados sobre as perspectivas física, lógica, de projecto e de parametrização. No âmbito do estudo de ferramentas de engenharia considera-se particularmente relevante a perspectiva lógica, nomeadamente o conceito de função cooperante que integra as componentes estática, de comportamento e de interface.

Uma escolha de elementos da IEC 61131-3 e de redes de Petri, a inclusão de elementos para definição de interfaces gráficos e a integração coesa destes elementos através de conceitos chave de orientação ao objecto UML é uma solução abrangente de linguagem de modelação aplicável ao desenho de sistemas de automação. O meta-modelo de uma linguagem, definido sobre esta perspectiva, pode suportar o desenho de sistemas de supervisão, controlo e automação de vários sistemas industriais, no domínio lato dos sistemas de eventos discretos.

A introdução de uma semântica de domínio, definida sobre o meta-modelo da linguagem e enquadrada numa arquitectura descrita em normas internacionais recentes, permite o desenvolvimento de uma ferramenta de engenharia directamente aplicável à automação de instalações do sistema eléctrico. Esta ferramenta pode assim integrar componentes de desenho e manipulação de modelos, documentação, simulação, *deployment* e ensaio.

O desenvolvimento de uma ferramenta baseada na ASML é possível através das orientações de implementação apresentadas, nomeadamente o IDE gráfico modular que integra componentes específicos como o repositório e o editor de modelos ou compiladores dedicados.

Considera-se assim, como conclusão final do trabalho, que a abordagem apresentada indica uma orientação viável para o desenvolvimento de ferramentas de engenharia capazes de suportar o desenho conduzido por modelos de sistemas de automação de terceira geração. Ferramentas de engenharia desenvolvidas segundo esta abordagem possibilitarão o desenvolvimento integrado de sistemas de automação de forma eficaz e eficiente, com vantagens na

redução do custo e no incremento da operacionalidade das actividades de engenharia de automação, bem como na promoção da qualidade do sistema final.

1.3. Considerações Finais

A preponderância dos dispositivos terminais, como elementos fundamentais do controlo automático do sistema eléctrico, dota o projecto dos sistemas de automação de instalações de importância fulcral na eficiência global do sistema eléctrico de energia. Os sistemas de automação desempenharão um papel primordial no suporte às novas necessidades de informação da *utility*, como servidores e processadores de informação de tempo real, bem como repositórios de meta-informação.

À data da conclusão da escrita desta dissertação estavam aprovadas treze das catorze partes da IEC 61850, hoje oficialmente consideradas normas electrotécnicas internacionais. A adopção da arquitectura comunicativa IEC 61850 na prática industrial, o alargamento desta norma a outras áreas do sistema eléctrico (monitorização de qualidade de energia^[46], recursos distribuídos^[47], produção eólica^[7] e produção hidroeléctrica^[48]), bem como a harmonização dos vários modelos IEC para o sistema eléctrico, indicam uma clara tendência para o estabelecimento de modelos de informação abrangentes e aceites de forma generalizada pela comunidade internacional.

O estabelecimento desta base de modelação comum suporta o aparecimento de plataformas de automação de terceira geração e sustenta o projecto de sistemas de automação conduzido por modelos, numa perspectiva semelhante à MDA. O desenvolvimento conduzido por modelos de sistemas de automação, suportado por ferramentas de projecto dedicadas, permitirá controlar o sistema eléctrico do futuro de forma segura, flexível, ágil e eficiente.

1.4. Orientações para Trabalho Futuro

Na sequência do trabalho desenvolvido na presente tese são apresentadas duas orientações para trabalho futuro: (i) a revisão e evolução da ASML e (ii) o desenvolvimento e aplicação de um protótipo funcional.

A revisão e evolução da ASML, discutida na avaliação da linguagem (parte II da dissertação), tem como objectivo colmatar algumas das suas lacunas, bem como expandir a linguagem a alguns elementos ou sistemas não equacionados ou não aprofundados no âmbito deste trabalho. Destes destacam-se domínios da automação do sistema eléctrico, como sejam os sistemas de produção distribuída, a produção hidroeléctrica, térmica ou os parques eólicos, ou outros domínios industriais como os sistemas de comunicações, as redes de águas e as redes de gás. O desenvolvimento deste tipo de trabalhos poderá contribuir para a definição de uma linguagem e de modelos de domínio para projecto conduzido por modelos de sistemas de supervisão, controlo e automação de sistemas industriais.

O desenvolvimento de uma ferramenta prototipal, com a inclusão da possibilidade de validação de modelos e *deployment* para sistemas reais, poderá constituir uma segunda orientação de evolução futura deste trabalho. A utilização desse protótipo no desenvolvimento de um sistema de automação real permitiria verificar na prática a adequabilidade das soluções propostas neste trabalho, através da recolha de indicadores de desempenho da abordagem de projecto conduzida por modelos.

Referências

- [1] M. Samotyj, D. Dollen, B. Howe (2002). **Powering the Digital Revolution: Electric Power Security, Quality, Reliability, and Availability in the Digital Age**. International Conference on Power Systems and Communications Infrastructures for the Future, Beijing, Setembro de 2002. Disponível em www.cris-inst.com/publications/beijing_2002/cris2002.htm a 10 de Janeiro de 2004.
- [2] G. Ziegler (2003). **Protection and Substation Automation - State of the Art and Development Trends**. Electra, 206, 14-23.
- [3] T. W. Cease, S.H. Horowitz, J.S. Thorp (2002). **Protection and Control - A Blueprint for the Future of a Secure Power System Infrastructure**. International Conference on Power Systems and Communications Infrastructures for the Future, Beijing, Setembro de 2002. Disponível em www.cris-inst.com/publications/beijing_2002/cris2002.htm a 10 de Janeiro de 2004.
- [4] IEEE Power Engineering Society – Power System Relaying Committee – System Protection Subcommittee – Working Group C-6 (2003). **IEEE PSRC Final Report: Wide Area Protection and Emergency Control**. IEEE. Disponível em www.pes-psrc.org/Reports a 15 de Fevereiro de 2004.
- [5] R. Paulo, A. Carrapatoso (2003). **Arquitetura de Comunicações Baseada em Ethernet para Sistemas Integrados de Automação de Instalações do Sistema Eléctrico de Energia**. 6º Encontro Nacional do Colégio de Engenharia Electrotécnica da Ordem dos Engenheiros, Lisboa, Maio de 2002.
- [6] K. Schwarz (2002). **Standard IEC 61850 for Substation Automation and Other Power System Applications**. International Conference on Power Systems and Communications Infrastructures for the Future, Beijing, Setembro de 2002. Disponível em www.cris-inst.com/publications/beijing_2002/cris2002.htm a 10 de Janeiro de 2004.
- [7] International Electrotechnical Commission, Technical Committee 88, Project Team 25 (2003). **IEC 61400-25: Wind turbines - Part 25: Communications for monitoring and control of wind power plants (88/179/CD)**.
- [8] International Electrotechnical Commission, Technical Committee 57 (2001). **IEC 61850-1: Communication networks and systems in substations – Part 1: Introduction and overview (57/524CDV)**.
- [9] International Electrotechnical Commission, Technical Committee 57 (2003). **IEC 61850-5: Communication networks and systems in substations – Part 5: Communication requirements for functions and device models (57/641/FDIS)**.
- [10] K-P. Brand, V. Lohmann, W. Wimmer (2003). **Substation Automation Handbook**. Utility Automation Consulting Lohmann, Bremgarten, Suíça, ISBN 3-85758-951-5.
- [11] EFACEC Sistemas de Electrónica, S.A. (2003). **Automação de Sistemas de Energia – Produtos**. Catálogo Electrónico.
- [12] D. J. Dolezilek (1999). **Power Systems Automation**. Schweitzer Engineering Laboratories. Disponível em www.selinc.com/techpprs/6091.pdf a 13 de Fevereiro de 2004.

- [13] International Electrotechnical Commission, Technical Committee 57 (2002). **IEC 61850-6: Communication networks and systems in substations – Part 6: Configuration description language for communication in electrical substations related to IEDs (57/614/CDV)**.
- [14] A.V. Ratzer, L. Wells, et. al. (2003). **CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets**. Disponível em wiki.daimi.au.dk/cpntools/publications.wiki a 20 de Fevereiro de 2004.
- [15] R. P. Rocha (1998). **Desenvolvimento de um Sistema de Gestão de AGVs**. Faculdade de Engenharia da Universidade do Porto, Dissertação de Mestrado.
- [16] GPC mbH (2002). **Poses++ Documentation**. Disponível em www.gpc.de a 20 de Fevereiro de 2004.
- [17] Alejandro Ramirez et. al. (2003). **ArgoUML User Manual: A tutorial and reference description of ArgoUML**. Disponível em argouml.tigris.org/documentation/defaulthtml/manual/ a 15 de Fevereiro de 2004.
- [18] M. Boger et. al. (2003). **Poseidon for UML Users Guide**. Disponível em www.gentleware.com/support a 15 de Fevereiro de 2004.
- [19] Visual Paradigm (2003). **Visual Paradigm for UML 3.0 User Guide**. Disponível em www.visual-paradigm.com a 15 de Fevereiro de 2004.
- [20] Altersys Europe (2002). **ISaGRAF Version 3.46 User's Guide**. Disponível em www.altersys.com/support/manuals.asp a 5 de Dezembro de 2004.
- [21] J. Poole (2001). **Model-Driven Architecture: Vision, Standards and Emerging Technologies**. Disponível em www.omg.org/mda/model_driven_architecture.htm a 5 de Janeiro de 2004.
- [22] J. Christensen (2000). **Design patterns for systems engineering with IEC 61499**. Disponível em www.holobloc.com/papers/1499_despat.zip a 12 de Fevereiro de 2004.
- [23] R. David, H. Alla (1992). **Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems**. Prentice Hall. Reino Unido. ISBN 0-13-327537-X.
- [24] F. Bonfatti, P. Monari, U. Sampieri (1997). **IEC 1131-3 Programming Methodology**. CJ International, Seyssins, França, ISBN 2-9511585-0-5.
- [25] G. Booch, J. Rumbaugh, I. Jacobson (1999). **The Unified Modeling Language User Guide**. Addison Wesley Longman, Reading, EUA, ISBN 0-201-57168-4.
- [26] J. Rumbaugh, I. Jacobson, G. Booch (1999). **The Unified Modeling Language Reference Manual**. Addison Wesley Longman, Reading, EUA, ISBN 0-201-30998-X.
- [27] M. Zhou, K. Venkatesh (1999). **Modeling, Simulation and Control of Flexible Manufacturing Systems – A Petri Net Approach**. World Scientific, Singapura, ISBN 981023029X.
- [28] W3C (2003). **About SVG: 2D Graphics in XML**. Disponível em www.w3.org/Graphics/SVG/About a 17 de Fevereiro de 2004.
- [29] Object Management Group (2003). **OMG Unified Modeling Language Specification (March 2003, Version 1.5)**. Disponível em www.omg.org/technology/documents/formal/uml.htm a 14 de Fevereiro de 2004.

- [30] J. Moody, P. Antsaklis (1998). **Supervisory Control of Discrete Event Systems Using Petri Nets**. Kluwer Academic Publishers, Norwell, EUA, ISBN 0-7923-8199-8.
- [31] A. Desrochers, R. Al-Jaar (1994). **Applications of Petri Nets in Manufacturing Systems – Modeling, Control and Performance Analysis**. IEEE Press, New York, EUA, ISBN 0-87942-295-5.
- [32] L. Barruncho (1986). **Redes de Petri na Concepção e Desenvolvimento de Sistemas – Aplicação a um Autômato de Subestações**. Instituto Superior Técnico, Dissertação de Mestrado.
- [33] J. Pinto de Sá (1988). **Automatismos Comunicantes em Subestações de Distribuição**. Instituto Superior Técnico, Dissertação de Doutoramento.
- [34] J. Tang, F. Wang (1998). **Modeling of a Transmission Network Protection System Using Petri Nets**. Electric Power Systems Research, 44, 175-181.
- [35] EFACEC Sistemas de Electrónica, S.A. (2003). **AGR500: Autômato de Grupo – Manual de Utilização**. Documento Interno.
- [36] EFACEC Sistemas de Electrónica, S.A. (2002). **ARTA500: Automatic Voltage Regulator – User Requirements Document**. Documento Interno.
- [37] K. Jesen (1997). **Coloured Petri Nets – Basic Concepts**. Springer, Berlin, Alemanha, ISBN 3-540-60943-1.
- [38] K. Jesen (1997). **Coloured Petri Nets – Analysis Methods**. Springer, Berlin, Alemanha, ISBN 3-540-58276-2.
- [39] W3C (2003). **Scalable Vector Graphics (SVG) 1.1 Specification**. Disponível em www.w3.org/TR/SVG11 a 28 de Fevereiro de 2004.
- [40] EFACEC Sistemas de Electrónica, S.A. (2000) **Manual do Editor de Sinópticos SCATEX**.
- [41] R. Fernandes (2003). **Interfaces Web para aplicações SCADA**. Faculdade de Ciências da Universidade do Porto, Dissertação de Mestrado.
- [42] International Electrotechnical Commission, Technical Committee 57 (2003). **IEC 61850-7-1: Communication networks and systems in substations – Part 7-1: Basic communication structure for substation and feeder equipment – Principles and models (57/637/FDIS)**.
- [43] International Electrotechnical Commission, Technical Committee 57 (2002). **IEC 61850-7-2: Communication networks and systems in substations – Part 7-2: Basic communication structure for substation and feeder equipment – Abstract communication service interface (ACSI) (57/612/FDIS)**.
- [44] International Electrotechnical Commission, Technical Committee 57 (2001). **IEC 61850-7-3: Communication networks and systems in substations – Part 7-3: Basic communication structure for substation and feeder equipment – Common data classes (57/519/CDV)**.
- [45] International Electrotechnical Commission, Technical Committee 57 (2003). **IEC 61850-7-4: Communication networks and systems in substations – Part 7-4: Basic communication structure for substation and feeder equipment – Compatible logical node classes and data classes (57/622/FDIS)**.
- [46] International Electrotechnical Commission, Technical Committee 57 (2003). **New work item proposal: Power Quality Monitoring Addendum to IEC 61850 (57/624/NP)**.

- [47] International Electrotechnical Commission, Technical Committee 57 (2003). **New work item proposal: Communications Systems for Distributed Energy Resources (57/660B/NP)**
- [48] International Electrotechnical Commission, Technical Committee 57 (2003). **New work item proposal: Hydroelectric power plants – communication for monitoring and control (57/661/NP)**
- [49] T. Boudreau et. al. (2002). **NetBeans: The Definitive Guide (Draft)**. Disponível em www.netbeans.org/download/books/definitive-guide/index.html a 10 de Maio de 2003.
- [50] Object Management Group (2002). **Meta-Object Facility (MOF) Specification (April 2002, Version 1.4)**. Disponível em www.omg.org/technology/documents/formal/mof.htm a 14 de Fevereiro de 2004.
- [51] Object Management Group (2002). **OMG XML Metadata Interchange (XMI) Specification (January 2002, Version 1.2)**. Disponível em www.omg.org/technology/documents/formal/xmi.htm a 14 de Fevereiro de 2004.
- [52] Sun (2003). **Creating a GUI with JFC/Swing**. Disponível em java.sun.com/docs/books/tutorial/uiswing a 10 de Maio de 2003.
- [53] Microsoft Corporation (2003). **MFC**. Disponível em msdn.microsoft.com/library/en-us/vccore/html/vcoriMFC.asp a 10 de Julho de 2003.
- [54] Microsoft Corporation (2003). **GDI+ Documentation**. Disponível em msdn.microsoft.com/library/en-us/gdicpp/GDIPlus/AboutGDIPlus.asp a 10 de Julho de 2003.
- [55] Sun (2001). **Java 2D API Specification**. Disponível em java.sun.com/j2se/1.4.2/docs/guide/2d/spec.html a 10 de Maio de 2003.
- [56] Object Technology International, Inc. (2003) **Eclipse Overview**. Disponível em www.eclipse.org/whitepapers/eclipse-overview.pdf a 14 de Fevereiro de 2004.
- [57] E. Gamma. **JHotDraw Documentation**. Disponível em jhotdraw.sourceforge.net/online-docs/documentation/documentation.html a 10 de Maio de 2003.
- [58] Eclipse. **GEF Project Information**. Disponível em www.eclipse.org/gef/ a 14 de Fevereiro de 2004.
- [59] R. Dirckze et. al. (2002). **Java Metadata Interface (JMI) Specification Version 1.0**. Disponível em java.sun.com/products/jmi a 14 de Fevereiro de 2004.
- [60] Netbeans (2003). **MDR – Architecture**. Disponível em mdr.netbeans.org/architecture.html a 14 de Fevereiro de 2004.
- [61] A. Powel (2004). **Model-driven development with the Eclipse Modeling Framework**. Disponível em www-106.ibm.com/developerworks/opensource/library/os-ecemf1/ a 10 de Abril de 2004.
- [62] Sun (2003). **The Java Tutorial**. Disponível em java.sun.com/docs/tutorial a 10 de Maio de 2003.
- [63] Apache (2004). **Xerces-C++ API documentation v2.5.0**. Disponível em xml.apache.org/xerces-c/index.html a 14 de Fevereiro de 2004.
- [64] International Electrotechnical Commission, Technical Committee 57 (2003). **IEC 60870-5-101 Ed.2: Telecontrol equipment and systems - Part 5-101: Transmission protocols - Companion standard for basic telecontrol tasks**.

- [65] International Electrotechnical Commission, Technical Committee 57 (2000). **IEC 60870-5-104: Telecontrol equipment and systems - Part 5-104: Transmission protocols - Network access for IEC 60870-5-101 using standard transport profiles.**

Anexo I. IEDs Comuns

Tabela 21. IEDs com funções essencialmente comunicativas.

IED	Descrição
Gateway	Neste tipo genérico de IED incluem-se dispositivos com funções de <i>firewall</i> , conversão de protocolos, encaminhamento, comutação de pacotes, <i>proxy</i> ou outras. Os parâmetros associados a <i>gateways</i> são em regra parâmetros da classe de sistema e da categoria de configuração. Permitem estabelecer regras de conversão de tipos, nomes e endereçamento de dados, bem como a parametrização de pilhas de comunicações. As <i>gateways</i> não apresentam vulgarmente HMI's nem funções programáveis.
URT	A URT convencional é um dispositivo com funções de interface à instalação e de envio de dados de processo a centros de comando remotos, bem como de recepção de ordens de manobra de equipamentos primários. Na vertente comunicativa as URTs apresentam parâmetros idênticos àqueles das <i>gateways</i> e, na vertente de interface ao processo, parâmetros de processo.

Tabela 22. IEDs essencialmente associados à interface com o utilizador.

IED	Descrição
Estação de trabalho	Uma estação de trabalho, por vezes única numa dada instalação, apresenta funções de HMI complexas: quadros sinópticos múltiplos e ricos, arquivo de dados, gráficos de tendência e gestão de alarmes, entre outros. Normalmente baseadas em PCs, fixos ou móveis, operam no nível funcional de instalação e integram todo o modelo da instalação. Através destes IEDs é possível obter informação de processo, agir sobre o equipamento primário, modificar a parametrização de outros IEDs e monitorizar a operação de dispositivos. A parametrização de estações de trabalho caracteriza-se por apresentar um volume de parâmetros elevado, maioritariamente da classe de sistema e da categoria de configuração.
IED com HMI integrado	IEDs com funções menos abrangentes, de nível de painel ou processo, apresentam frequentemente interfaces ao utilizador baseadas em sinópticos simples parametrizáveis.

Tabela 23. IEDs essencialmente associados ao processo.

IED	Descrição
Transformador de medida	Os transformadores de medida são dispositivos usualmente não parametrizáveis, cuja função é a conversão de nível de grandezas analógicas eléctricas (tensão e corrente) do sistema primário para níveis mais reduzidos, tratáveis pelos dispositivos do sistema secundário, vulgarmente de 0 a 1A ou 5A.
Transdutor	Para dispositivos com necessidades de medição de grandezas analógicas que se resumem a valores RMS são por vezes utilizados dispositivos que executam filtragem e cálculo de grandezas provenientes de transformadores de medida. Estes valores são disponibilizados a outros IEDs do sistema de automação, através de uma interface comunicativa série ou sinal eléctrico. Estes dispositivos apresentam parametrização simples de sistema e de configuração.
Sensor inteligente	Surgem hoje dispositivos não convencionais de sensorização de correntes, tensões ou posição de órgãos, baseados em princípios electro-ópticos e que incluem já pré-filtragem e conversão A/D. Estes dispositivos apresentam parametrização simples e reduzida, disponibilizando a informação sensorial em formato digital. Incluem-se também nesta categoria os equipamentos primários inteligentes que disponibilizam interfaces não eléctricos, mas baseados em barramentos de comunicação série.
PLC	Os PLCs são autómatos programáveis, ou seja, dispositivos cuja função convencional é a execução de funções automáticas discretas. Apresentam interface ao processo e, como a quase totalidade dos IEDs, funções comunicativas, essencialmente locais. As necessidades de parametrização deste tipo de dispositivos dependem do conjunto de funcionalidades que integram, mas podem abranger todos as classes e categorias.
URT	As URTs têm vindo a englobar capacidade de executar funções automáticas programáveis e de HMI, mas caracterizam-se essencialmente por realizarem a interface ao processo. Apresentam arquitecturas centralizadas ou distribuídas e podem integrar milhares de pontos de interface à instalação.
Protecção	As protecções são dispositivos cuja função básica é a protecção de equipamentos primários. Executam vulgarmente um conjunto de funções dedicadas de análise com algoritmos pré-implementados. Diferenciam-se pelo dispositivo para qual estão vocacionadas: protecção de linha, protecção de transformador, protecção de gerador, protecção de barramento, etc. Apresentam interface directa ao processo, sem necessidade de transdutores, com um número de pontos reduzido correspondente às grandezas cuja medição é essencial à função de protecção. Podem integrar funções de automação programável e integram frequentemente HMI local e interfaces comunicativas. Todos os tipos de parâmetros estão presentes na parametrização de protecções.
Controlador de instalação	O controlador de instalação é um dispositivo, dedicado ou baseado numa URT ou PLC, que executa funções, normalmente programáveis, de nível de instalação. Apresenta todos os tipos de parametrização.
Controlador de painel	Um controlador de painel é uma classificação de dispositivos de controlo automático de nível de painel. Podem ser baseados em URTs, PLCs, protecções ou controladores dedicados. Apresenta todos os tipos de parametrização.
Contador (Meter)	Trata-se de um dispositivo de medição de fluxos de energia, com principal aplicação na aquisição e arquivo de dados, tendo em vista o custeio de trocas de energia.
Qualímetro	Dispositivo responsável pela aquisição de dados de qualidade de energia, com funções de aquisição (com frequências de amostragem elevadas), filtragem, tratamento estatístico e arquivo, para pós-processamento. Inclui parametrização de configuração e operação, mas essencialmente de sistema e processo.

Anexo II. Funções do Sistema de Automação

Tabela 24. Funções básicas.

Função	Descrição
Aquisição de dados e actuação	Trata-se das funções mais características dos sistemas de automação de instalações. Compreendem a aquisição de informação do processo controlado: informação de estado discreto, como a posição de órgãos, e de grandezas contínuas como temperaturas, pressões, frequências, tensões, correntes ou potências. A aquisição de informação inclui o estado (valor actual), determinação de acontecimentos (alterações ao estado actual ou eventos dissociados do estado) e processamento estatístico inicial. A actuação sobre o processo, que recai também sobre as interfaces do nível funcional inferior, compreende essencialmente ordens de manobra discretas de órgãos ou outros equipamentos.
Controlo remoto	A operação remota da instalação é o papel convencional de uma subestação e função importante nas restantes instalações do sistema eléctrico. Compreende o envio de informação adquirida ou processada e a execução de ordens de manobra remotas provenientes de centros de comando ou outros elementos externos à instalação.
HMI	A interface ao operador, local e remota, suporta outro papel básico de qualquer instalação: a operação manual do processo. Toda a informação disponível na instalação está vulgarmente acessível no HMI local. Este inclui informação de tempo-real ou diferida do processo e da operação do sistema secundário. A função de interface para operação manual local é cada vez mais usada como solução de recurso ou para apoio a intervenções de manutenção ou remodelação, visto que a operação remota e automática abrangem hoje a quase totalidade das funções do sistema. A função de HMI revela-se também particularmente relevante na análise diferida, quer na análise de incidentes, quer na elaboração de planos de manutenção ou de contingência, bem como no prognóstico de comportamentos futuros do sistema. Esta função envolve a monitorização de estado em tempo-real, o controlo directo, a gestão de eventos, a gestão de alarmes, o arquivo de dados e sua recuperação, a aquisição e gestão de perturbografia, o registo de operações sobre o sistema e a autenticação de utilizadores e controlo de acesso. Os paradigmas de interface comuns incluem os painéis sinópticos gráficos, as listas, os gráficos de tendência e os relatórios sumários. Para aplicações específicas são usadas ferramentas de análise dedicadas.
Controlo automático	O controlo automático compreende uma diversidade de funções que envolvem o controlo de equipamentos primários, mas tendo como objectivo essencial o controlo do desempenho global do sistema eléctrico: controlo de tensão, frequência, topologia de rede, prevenção de instabilidade, contenção de incidentes, etc. Na tabela 25 apresentam-se algumas funções concretas deste tipo.
Comunicação	Trata-se de um função de importância elevada no suporte à operação do sistema de automação, inerentemente distribuído, quer na vertente local à instalação, quer em associações comunicativas exteriores como a comunicação remota com centros de operação ou outras instalações.

Tabela 25. Funções de controlo automático comuns.

Função	Descrição
Protecção	É uma função de controlo automático particular, cujo objectivo é a garantia de segurança de operação de dispositivos primários como geradores, linhas, transformadores, etc. A protecção inclui a identificação de condição de falha para remoção ou desactivação do equipamento afectado. Várias funções de protecção são distinguidas pelos seus princípios de identificação de falha que, frequentemente, se podem associar à protecção de dispositivos distintos. Alguns princípios de funcionamento comuns são: (i) a sobre-corrente, instantânea ou constante (curto-circuito), (ii) a análise diferencial, de tensões ou correntes (transformador), (iii) a determinação de impedância (protecção de distância), (iv) o cálculo de intervalos temporais (falha de disjuntor), (v) o sobreaquecimento, (vi) a quebra de pressão, entre outras.
Inibição de comandos	A execução de comandos é apenas permitida em determinadas condições de operação do sistema. Um dado comando pode ser inibido por inúmeras razões: estado incorrecto de funcionamento do dispositivo primário a actuar, ausência de privilégios de acesso, modo de operação do sistema (local, remoto, etc.), estado de topologia, entre outros. Esta função, descrita por equações lógicas simples, compreende vulgarmente relações com múltiplas funções de níveis funcionais distintos.
Controlo de tomada de transformador	Controlo incremental da posição de tomadas de transformadores, com vista à alteração das razões de transformação.
Comutação sincronizada	Controlo do instante de manobra de órgãos, com vista à redução do impacto transitório da comutação no sistema eléctrico.
Religação	Descrita por equações lógicas ou máquinas de estados, envolve a religação rápida ou lenta de um dispositivo, vulgarmente após a acção de uma protecção. Esta função permite restabelecer o fornecimento de energia no caso de perturbações transitórias.
Verificação de sincronismo	Função cujo objectivo é a detecção da diferença de fase e frequência entre tensões ou correntes presentes em dispositivos distintos. Suporta a função de comutação sincronizada.
Encravamento	O objectivo desta função consiste na protecção contra a destruição ou danificação de equipamento primário na execução de comandos. Compreende a determinação de condições de operação de órgãos, baseada na topologia da instalação, ou seja, nas posições dos vários órgãos. Esta função pode estar associada a níveis funcionais distintos e é vulgarmente descrita por equações lógicas simples (que incluem o conhecimento das sequências comuns de operação) ou por máquinas de estado (com base na topologia dinâmica). Compreende vulgarmente relações com múltiplas funções de níveis funcionais distintos.
Estimação	Com o objectivo de minimizar o número de sensores, pode ser relevante a determinação de valores de grandezas eléctricas ou mesmo posição de órgãos, com base na informação sensorial disponível. Esta função depende da organização do equipamento primário e é vulgarmente descrita por equações aritméticas e máquinas de estado.
Sequências de manobras	A colocação em serviço (ou remoção) de um dispositivo primário como uma linha, transformador ou gerador, ou a alteração de topologia implica vulgarmente a execução de sequências condicionadas. As máquinas de estados são vulgarmente utilizadas para definir sequências de manobras. Uma sequência de manobras pode ser hierárquica e abranger mais do que um nível funcional.

Tabela 25. Funções de controlo automático (continuação).

Função	Descrição
Protecção adaptativa	Modificação automática de parâmetros de protecções após alteração de fluxos de potência ou da topologia da instalação.
Bloqueio inverso de protecções	Função que abrange mais do que uma instalação e que possibilita a coordenação de disparos de protecções, de forma a minimizar o impacto de falhas. É vulgarmente aplicada em redes radiais.
Deslastre	Para contrariar variações bruscas da relação consumo/produção, com implicações na estabilidade de frequência e níveis de tensão, podem ser selectivamente removidos consumidores, evitando assim quebras generalizadas da disponibilidade de energia. Esta função é particularmente importante em redes isoladas e a sua especificação implica a definição de máquinas de estado para diversos esquemas de deslastre, dependentes de condições de operação de rede como cargas e topologia.
Reposição	Após a declaração de uma falha permanente, ocorre o isolamento de partes do sistema eléctrico. Uma vez removidas as condições de falha, tem lugar a reposição faseada de dispositivos primários, baseada em regras especificadas em equações lógicas temporizadas ou máquinas de estado.
Controlo de tensão ou potência reactiva	Controlo incremental de tomadas de transformadores ou bancos de condensadores com vista à manutenção de níveis de tensão ou potência reactiva. Estes algoritmos são vulgarmente definidos com recurso a máquinas de estado.
Comutação de linhas e transformadores	Troca rápida de transformadores ou linhas de entrada em situações de falha, de forma a assegurar o fornecimento continuado de energia a consumidores críticos. Envolve a detecção de condições de comutação e a execução de sequências de comutação.
WAP	As funções de controlo automático anteriores têm um âmbito essencialmente local à instalação controlada ou a uma linha (envolvendo interacção limitada entre duas instalações). Têm vindo a ser implementados recentemente algoritmos de controlo automático distribuídos que operam com informação de rede e que compreendem a interacção entre funções automáticas de múltiplas instalações. Têm em vista melhorar o desempenho da rede em situações de incidentes como disparos em cascata, colapso de tensão ou frequência, fluxos oscilantes de energia ou outros. Estas funções compreendem a determinação de condição anómala, com base num sistema de decisão, e a actuação para adaptação do sistema. As acções são essencialmente discretas e esporádicas. Os algoritmos de actuação são vulgarmente pré-definidos e o controlo não apresenta realimentação. Alguns exemplos deste tipo de funções são: (i) o deslastre preventivo ou curativo de carga ou produção e (ii) o bloqueio ou adaptação de funções de controlo local, como a regulação de tensão, a religação ou a protecção. Estas funções permitirão a operação segura do sistema eléctrico perto dos seus limites, com um impacto directo na gestão de recursos e qualidade de fornecimento de energia.

Tabela 26. Funções de suporte.

Função	Descrição
Gestão de <i>software</i>	Envolve a gestão de versões de <i>firmware</i> , dos IEDs e da parametrização. Permite rastrear alterações e identificar versões em funcionamento.
Adaptação de <i>software</i>	Envolve a substituição de versões anteriores de <i>firmware</i> ou de parametrização.
Gestão de comunicações	Envolve a verificação de validade de ligações, sincronização de relógios, gestão de redundância, etc. A verificação de funcionamento anormal do sistema de comunicações pode ser tomada em consideração pelas funções de controlo automático.
Auto-verificação	A auto-verificação possibilita a determinação de condições de funcionamento degradado ou indisponível de determinadas funções num dado dispositivo, o que poderá implicar alterações nos esquemas de controlo automático e permite apoiar as actividades de gestão de manutenção.
Modo de operação	Uma dada instalação ou painel pode apresentar modos de operação diferenciados com impacto nas funções de controlo automático, nomeadamente se um dado sistema está em operação remota, em operação local ou em manutenção.

Anexo III. Modelo do Sistema de Automação

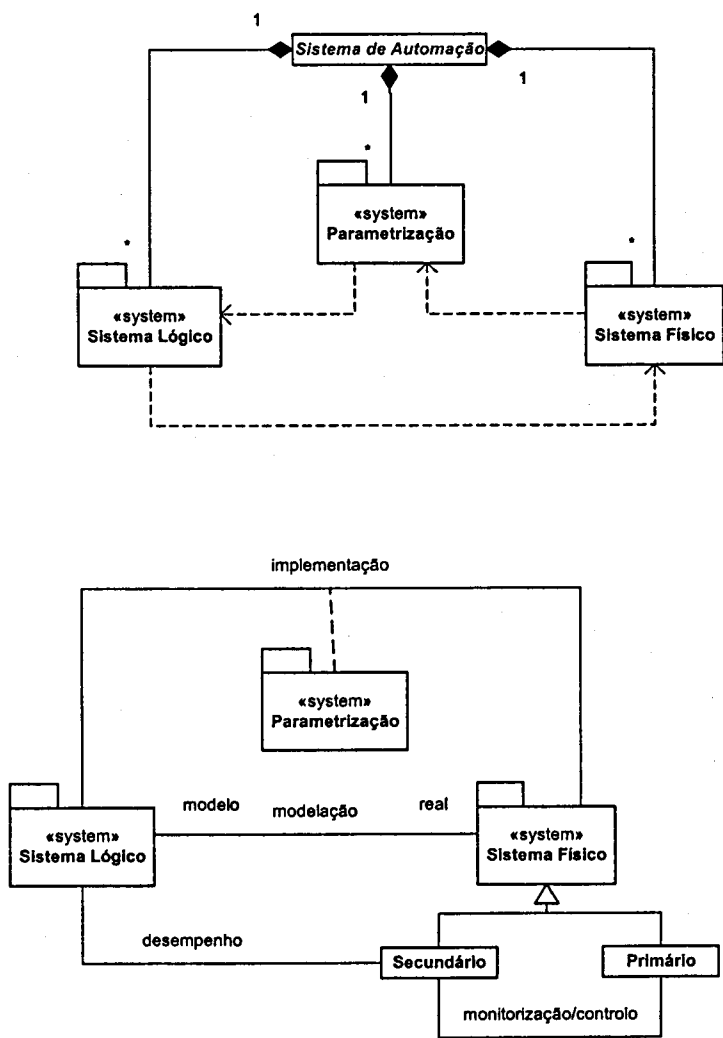


Figura 85. Sistema de automação.

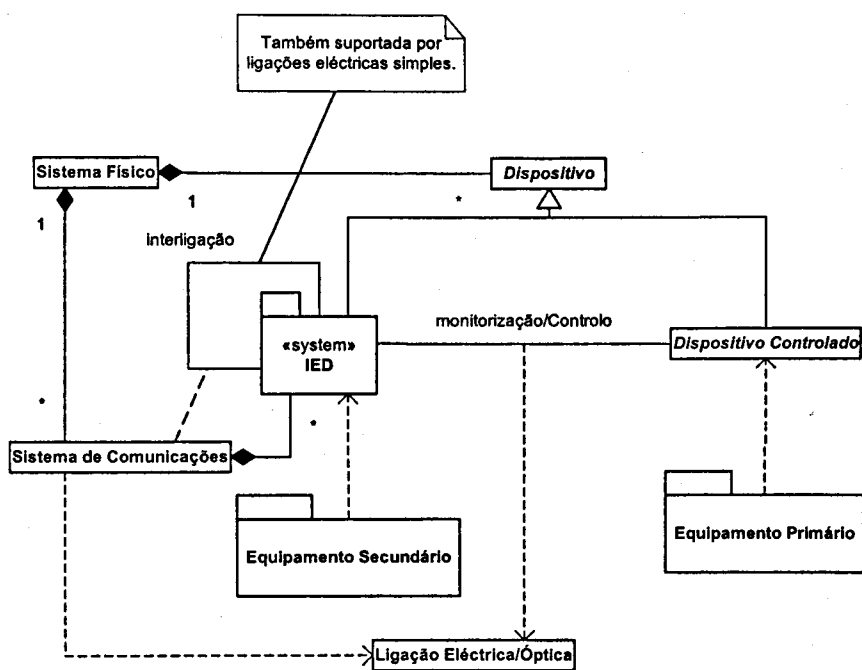


Figura 86. Sistema físico.

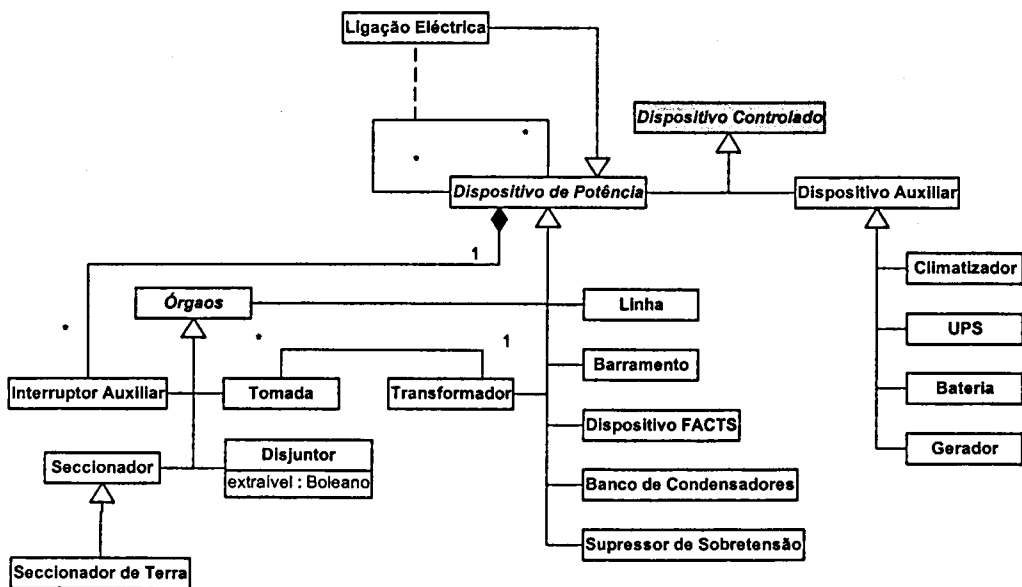


Figura 87. Equipamento primário vulgarmente encontrado em subestações.

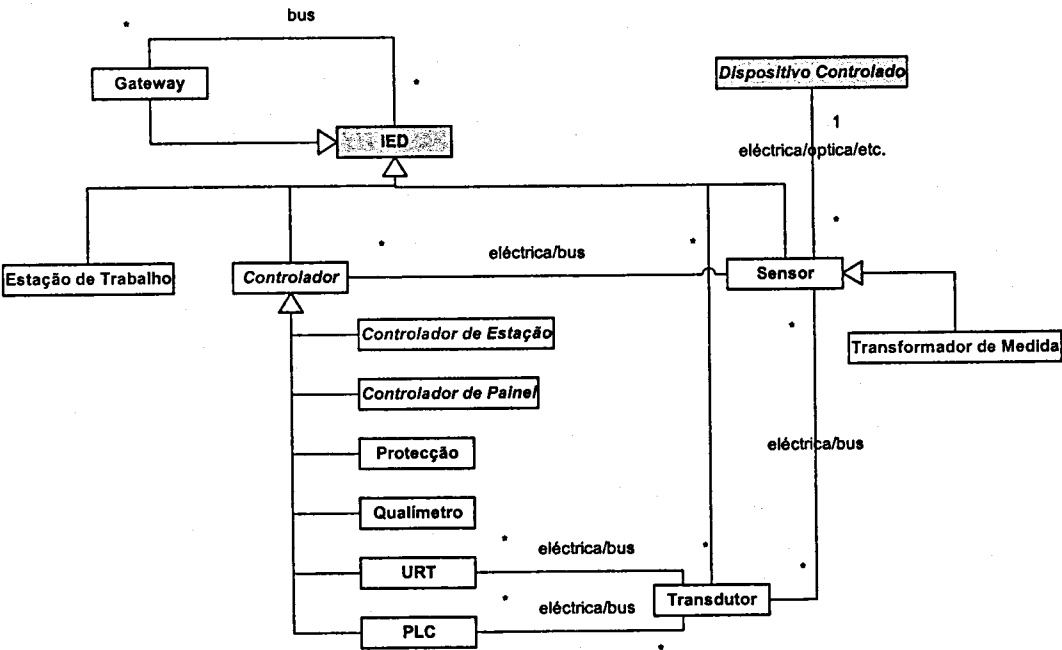


Figura 88. Equipamento secundário.

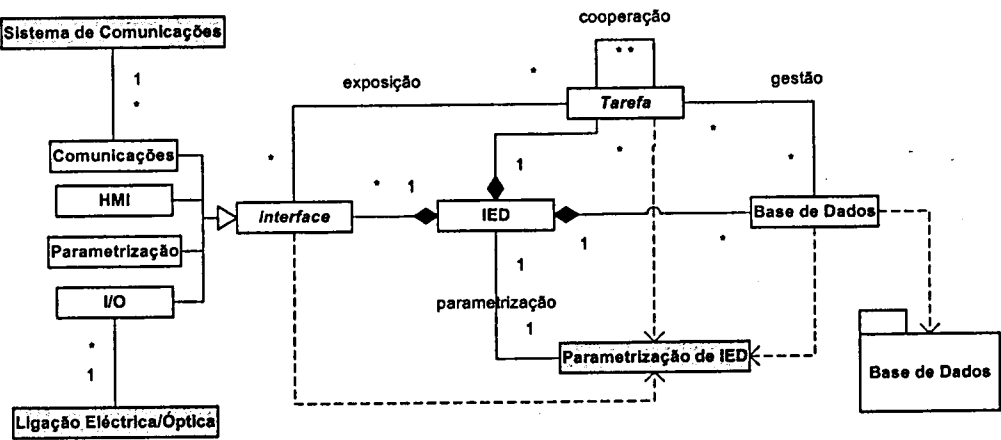


Figura 89. IED.

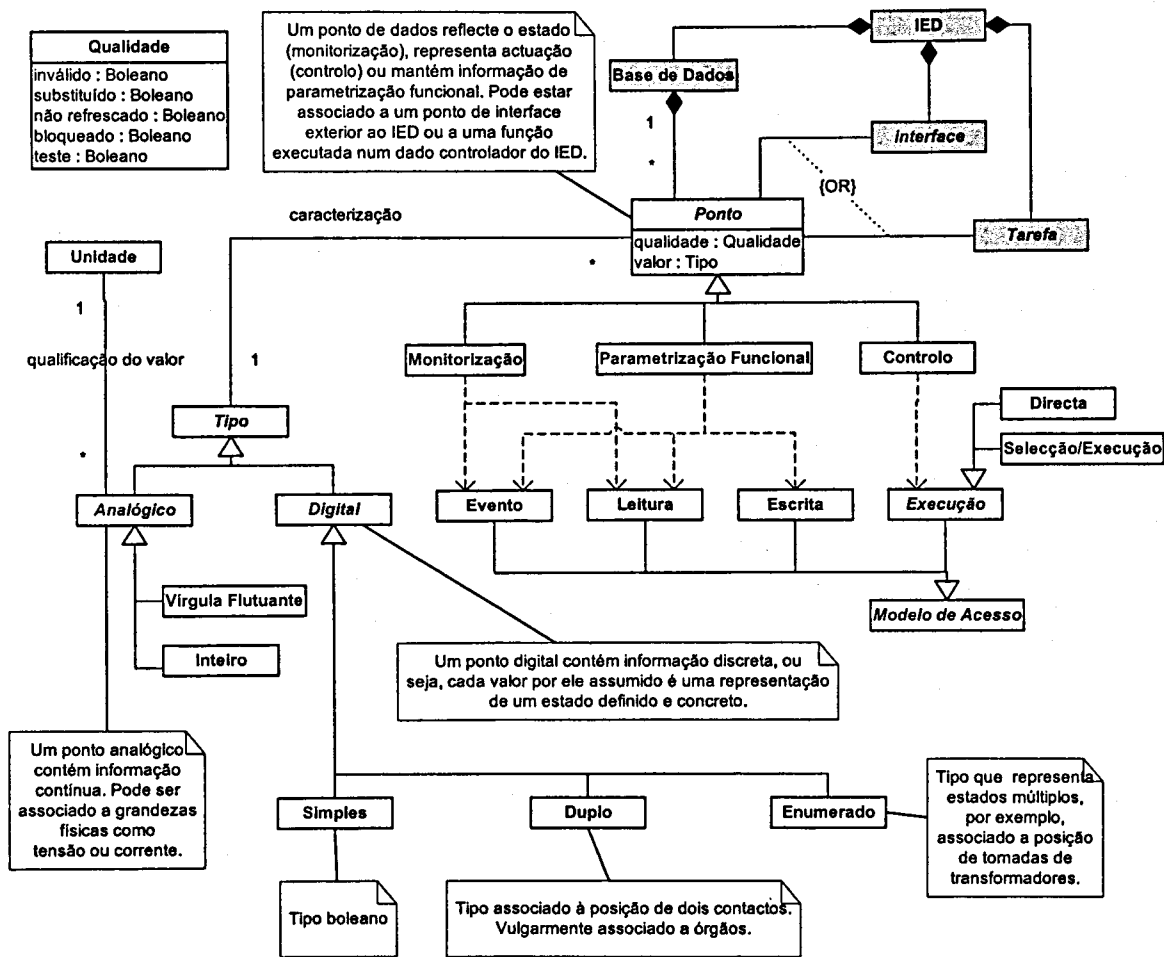


Figura 90. Base de dados de IED.

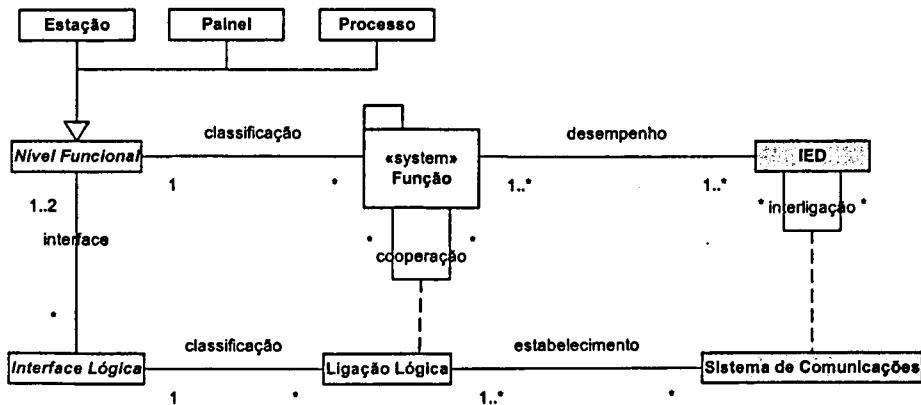


Figura 91. Sistema lógico.

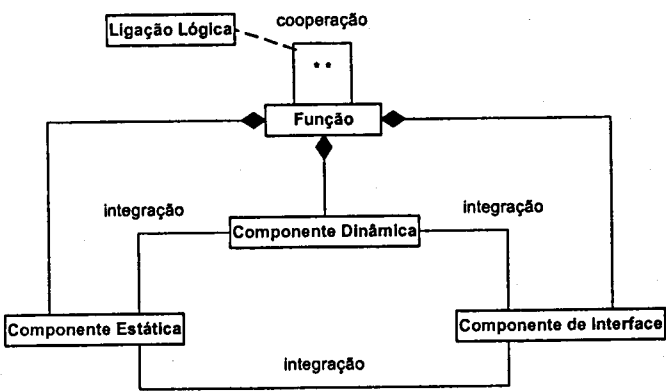


Figura 92. Função.

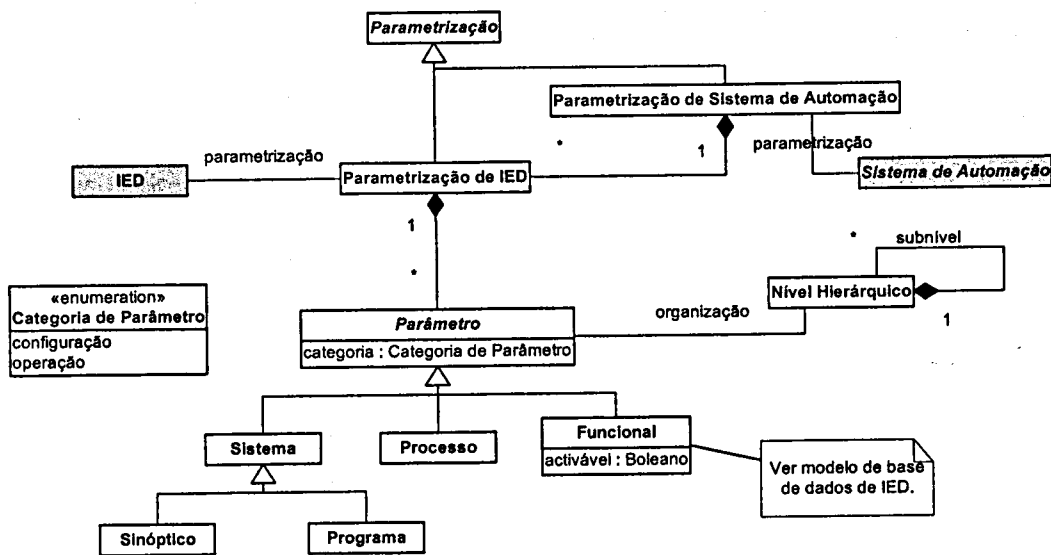


Figura 93. Parametrização.

Anexo IV. Gramática para Expressões e Procedimentos ASML

A gramática é apresentada, neste anexo, com uma notação BNF estendida. Esta notação BNF incorpora símbolos semelhantes aos símbolos utilizados na definição de gramáticas lex/yacc. O axioma da gramática é o símbolo *procedure*.

Tabela 27. Símbolos não terminais.

Sintaxe
<pre>procedure : expression block ; block : ";" statement ";" block statement ";" ; statement : declaration assignment call control return EXIT ; declaration : variableName ":" classifierReference (":=" constant)? ; variableName : ID ; assignment : assignableReference ":=" expression ; call : operationReference "(" parameters? ")" ; parameters : simpleParameters namedParameters ;</pre>

Tabela 27. Símbolos não terminais (continuação).

Sintaxe
simpleParameters : expression
simpleParameters "," expression
; '
namedParameters : namedParameter
namedParameters "," namedParameter
;
namedParameter : parameterName "!=" expression
;
parameterName : ID
;
return : RETURN expression?
;

Tabela 28. Expressões.

Sintaxe
cast : classifierReference "(" expression ")"
;
referenceSize : SIZE "(" (readableReference assignableReference) ")"
;

⁵ Lista ordenada de todos os parâmetros de entrada da operação.

Tabela 28. Expressões (continuação).

Sintaxe
expression : "(" expression ")"
call
constant
cast
readableReference
referenceSize
expression BINARY_LOGICAL_OR expression
expression BINARY_LOGICAL_XOR expression
expression BINARY_LOGICAL_AND expression
expression BINARY_COMPARISON_NOT_EQUAL expression
expression BINARY_COMPARISON_EQUAL expression
expression BINARY_COMPARISON_LARGER_EQUAL expression
expression BINARY_COMPARISON_LARGER expression
expression BINARY_COMPARISON_SMALLER_EQUAL expression
expression BINARY_COMPARISON_SMALLER expression
expression BINARY_ARITMETIC_SUBTRACTION expression
expression BINARY_ARITMETIC_ADDITION expression
expression BINARY_ARITMETIC_MODULO expression
expression BINARY_ARITMETIC_DIVISION expression
expression BINARY_ARITMETIC_MULTIPLICATION expression
UNARY_LOGICAL_NOT expression
UNARY_ARITMETIC_NEGATION expression
expression BINARY_ARITMETIC_EXPONENTIATION expression
;

Tabela 29. Símbolos terminais para operadores ordenados por ordem crescente de precedência.

Precedência da expressão	Léxico	Associatividade da expressão
1	BINARY_LOGICAL_OR : "OR" ;	esquerda
2	BINARY_LOGICAL_XOR : "XOR" ;	
3	BINARY_LOGICAL_AND : "AND" "&" ;	
4	BINARY_COMPARISON_NOT_EQUAL : "<>" ; BINARY_COMPARISON_EQUAL : "=" ;	
5	BINARY_COMPARISON_LARGER_EQUAL : ">=" ; BINARY_COMPARISON_LARGER : ">" ; BINARY_COMPARISON_SMALLER_EQUAL : "<=" ; BINARY_COMPARISON_SMALLER : "<" ;	
6	BINARY_ARITMETIC_SUBTRACTION : "-" ; BINARY_ARITMETIC_ADDITION : "+" ;	
7	BINARY_ARITMETIC_MODULO : "MOD" ; BINARY_ARITMETIC_DIVISION : "/" ; BINARY_ARITMETIC_MULTIPLICATION : "*" ;	
8	UNARY_LOGICAL_NOT : "NOT" ; UNARY_ARITMETIC_NEGATION : "-" ;	direita
9	BINARY_ARITMETIC_EXPONENTIATION : "****" ;	

Tabela 30. Instruções de controlo.

Sintaxe
<pre>control : if while for repeat case ; if : IF expression THEN block (ELSIF expression THEN block)? (ELSE block)? ENDIF ; case : CASE expression OF caseValues (ELSE block)? ENDCASE ; caseValues : caseValueList ":" block ; caseValueList : INTEGER caseValueList "," INTEGER ; while : WHILE expression DO block ENDWHILE ; repeat : REPEAT block UNTIL expression ENDREPEAT ; for : FOR variableReference ":-" INTEGER TO INTEGER (BY INTEGER)? DO block ENDFOR ;</pre>

Tabela 31. Referências.

Sintaxe
readableReference : variableReference instanceReference enumerationLiteralReference ;
assignableReference : variableReference instanceReference ;
operationReference : reference ;
instanceReference : reference ;
classifierReference : reference TYPENAME ;
enumerationLiteralReference : reference ;
variableReference : ID ;
reference : name ("[" expression "]")? ;
name : ID name SEPARATOR ID ;

Tabela 32. Símbolos terminais para valores constantes de tipos primitivos.

Léxico
constant : structure BOOL INTEGER REAL STRING TIME TIMEPERIOD;
BOOL : "FALSE" "TRUE" "0" "1" ;
INTEGER : DECIMAL HEXADECIMAL OCTAL BINARY ;
DECIMAL : ("-" "+")? [_0-9]+ ;
HEXADECIMAL : "16#" [_0-9A-Fa-f]+ ;
OCTAL : "8#" [_0-7]+ ;
BINARY : "2#" [0-1_]+ ;
REAL : ("-" "+")? DIGIT+ "." DIGIT+ (("F" "E") ("-" "+")? DIGIT+)? ;
STRING "'[^\$]'" ;
TIME : ("T" "t")"#"DIGIT{4}"-"DIGIT{1,2}"-"DIGIT{1,2} DIGIT{1,2}" ":"DIGIT{1,2}" ":"DIGIT{1,2}" ":" ("."DIGIT{1,3})* ;
TIMEPERIOD : (("T" "t")"#"?)?"timeComponent+ ;
timeComponent : DIGIT+TIMEUNIT ;
TIMEUNIT : "Y" "M" "D" "h" "m" "s" "ms" "us" "ns" ;
structure : "{" attributes? "}" ;
attributes : simpleAttributes namedAttributes;
simpleAttributes: constant simpleAttributes "," constant ; ⁶
namedAttributes : namedAttribute namedAttributes ","namedAttribute ;
namedAttribute : attributeName ":@" constant;
attributeName : ID ;

⁶ Lista ordenada de todos os atributos da estrutura.

Tabela 33. Restantes símbolos terminais.

Léxico
DIGIT : [0-9] ;
ID : [A-Za-z]([A-Za-z0-9_]) ⁷ ;
SEPARATOR : "." ":" ;
COMMENT : "(*" (."\\n") "*)" ; ⁷
EXIT : "EXIT" ;
IF : "IF" ;
THEN : "THEN" ;
ELSIF : "ELSIF" ;
ELSE : "ELSE" ;
ENDIF : "END_IF" ;
CASE : "CASE" ;
OF : "OF" ;
ELSE : "ELSE" ;
ENDCASE : "END_CASE" ;
WHILE : "WHILE" ;
DO : "DO" ;
ENDWHILE : "END_WHILE" ;
REPEAT : "REPEAT" ;
UNTIL : "UNTIL" ;
ENDREPEAT : "END_REPEAT" ;
FOR : "FOR" ;
TO : "TO" ;
BY : "BY" ;
DO : "DO" ;
ENDFOR : "END_FOR" ;
TYPENAME : "BOOL" "INT8" "INT16" "INT32" "INT128" "UINT8" "UINT16" "UINT32" "FLOAT32" "FLOAT64" "STRING" "TIME" "TIMEPERIOD" ;
SIZE : "SIZE" ;
RETURN : "RETURN" ;

⁷ Ignorado na gramática apresentada, tal como os caracteres brancos.

Tabela 34. Caracteres especiais para uso em constantes do tipo *String*.

Léxico
DOLLARSIGN : "\$\$" ;
APOSTROPHE : "\$'" ;
LINEFEED : "\$L" ;
CARRIAGERETURN : "\$R" ;
NEWLINE : "\$N" ;
NEWPAGE : "\$P" ;
TABULATION : "\$T" ;
ANYCHARACTER : "\$"CODE ;
CODE : [0-9A-Fa-f]+ ;

Anexo V. Elementos Textuais da Notação Gráfica ASML

Neste anexo é apresentada a sintaxe e léxico para os elementos textuais da notação gráfica da linguagem. Esta definição é apresentada com a mesma notação da gramática do anexo IV, recorrendo a algumas das suas produções.

Tabela 35. Sintaxe e léxico para elementos textuais da notação gráfica da linguagem.

Elemento textual	Sintaxe
Nomes	name Nomes de elementos de modelação podem ser apresentados de forma singular ou completa por concatenação de todos os <i>Namespace</i> hierarquicamente superiores. Para elementos abstractos o nome deve ser apresentado em itálico. Exemplos: <i>Disjuntor</i> SVC::PL::60KV
Palavras-chave	keyword : "<" ID ">" ; keywords : keyword keywords keyword ; Palavras-chave podem representar nomes de estereótipos ou ser utilizados para distinguir elementos de modelação com notações gráficas idênticas. Exemplos: «model»
Multiplicidade	single : DIGIT "*" DIGIT ".." DIGIT DIGIT ".." "*" ; multiplicity : single multiplicity "," single ; Exemplos: 0..1 0..*, pode ser substituído por: * 2..4, 6..8 1 1, 3..5
Valores	value : constant ; Valores, apenas para tipos primitivos, são representados com a sintaxe textual definida para constantes do pacote <i>Methods</i>. Exemplos: T#28h9m8s320ms7us -345 'uma cadeia de caracteres'

Tabela 35. Sintaxe e léxico para elementos textuais da notação gráfica da linguagem (continuação).

Elemento textual	Sintaxe
Lista de propriedades	<pre>prop : ID ID "=" value value; propset : prop propset "," prop ; propList : "(" propset ")" ;</pre> Uma lista de propriedades é normalmente utilizada para enumerar meta-atributos ou etiquetas atribuídas ao elemento, com a sintaxe <i>nome=valor</i>. No caso de meta-atributos pode aplicar-se uma sintaxe alternativa. Os meta-atributos booleanos podem estar apenas presentes no caso em que o seu valor for verdadeiro, figurando apenas o seu nome. No caso de meta-atributos enumerados pode apresentar-se somente o valor e omitir-se por completo caso o seu valor seja aquele atribuído por defeito. Exemplo: <pre>{leaf, author='Luís Silva', persistent}</pre>
Restrição	<pre>TEXT : (.. \n)* ; constraintBody : procedure TEXT ; constraint : '('constraintBody ')'</pre> Uma restrição é apresentada pelo seu corpo ou condição. Exemplo: <pre>{ Ua <= 0 }</pre>
Propriedade estrutural (atributo, valor de enumerado, parâmetro de sinal, extremo de associação ou definição de etiqueta de estereótipo).	<pre>VISIBILITY : "+" "#" "-" "~" ; featureName: ID ; typeName : name ; initialValue : "=" constant ; structuralFeature : VISIBILITY? featureName (":" typeName)? multiplicity? initialValue? propList? ;</pre> O meta-atributo <i>visibility</i> é representado por: + (<i>public</i>), # (<i>protected</i>), - (<i>private</i>) ou ~ (<i>package</i>). Elementos cujo âmbito é a classe e não a instância são apresentados em sublinhado. Exemplos: <pre>+contagem:UInt32 = 4 { read } -<u>numeroInstancias:UInt8</u></pre>
Comentário	<pre>comment : TEXT ;</pre> Um comentário é apresentado pelo seu corpo. Exemplo: <pre>Um comentário.</pre>

Tabela 35. Sintaxe e léxico para elementos textuais da notação gráfica da linguagem (continuação).

Elemento textual	Sintaxe
Propriedade comportamental (operação ou porto)	<pre>returnType: ":" typeName ; paramName : ID ; param : (("in" "out" "inout")? paramName (":" typeName)?) ; paramSet : param (paramSet "," param); paramList : "(" ")" "(" paramSet ")"; behaviouralFeature: VISIBILITY? featureName paramList? returnType? propList? ;</pre> A lista de parâmetros é constituída por todos os parâmetros da propriedade comportamental pela sua ordem. Operações abstractas figuram em itálico e operações cujo âmbito é a classe e não a instância em sublinhado. Exemplos: <u>+sin(In angulo:Float32) : Float32</u> <u>+abrir()</u>
Nomes de instâncias	<pre>instanceName : name (":" typeName)? name? ":" typeName ;</pre> Quer o nome da instância quer o nome do seu classificador podem ser omitidos. Os nomes de instâncias são apresentados em sublinhado. Exemplos: <u>SVC::PL::60KV:ST : Seccionador</u> <u>ST : Seccionador</u>
Nós de rede de Petri	<pre>nodeName : name; triggerEventName : name; delay : procedure ; guard: procedure ; effect : name ; node : nodeName propList? ; transition : node nodeName ":" triggerEventName ("[" (delay guard) "]")? ("/" effect)? propList? ;</pre> No caso de uma transição o nome pode apresentar informação adicional sobre o evento, condição e acção associados ao nó. Exemplo: <u>T1 : E1 [U < 60] / A1</u>

Anexo VI. Gramática para Definição de Caminhos

Esta gramática BNF é definida segundo a notação para gramáticas apresentada no anexo IV, sendo compatível com a SVG [39].

Tabela 36. Sintaxe para definição de caminhos.

Sintaxe
<pre>path : moveTo moveToDrawToCommand * ; moveToDrawToCommand : moveTo drawTo ; moveTo : (MOVERELATIVE MOVEABSOLUTE) coordinatePairSequence ; drawTo : closePath lineTo horizontalLineTo verticalLineTo curveTo smoothCurveTo quadraticBezierCurveTo smoothQuadraticBezierCurveTo ; closePath : CLOSEPATH ; lineTo : (LINERELATIVE LINEABSOLUTE) coordinatePairSequence ; horizontalLineTo : (HLINERELATIVE HLINEABSOLUTE) coordinateSequence ; verticalLineTo : (VLINERELATIVE VLINEABSOLUTE) coordinateSequence ; curveTo : (CURVERELATIVE CURVEABSOLUTE) curveToTupleSequence ;</pre>

Tabela 36. Sintaxe para definição de caminhos (continuação).

Sintaxe
<pre>curveToTupleSequence : curveToTuple curveToTupleSequence ","? curveToTuple ; curveToTuple : coordinatePair ","? coordinatePair ","? coordinatePair ; smoothCurveTo : (SCURVERELATIVE SCURVEABSOLUTE) smoothCurveToTupleSequence ; smoothCurveToTupleSequence : smoothCurveToTuple smoothCurveToTupleSequence ","? smoothCurveToTuple ; smoothCurveToTuple : coordinatePair ","? coordinatePair ; quadTo : (QUADRELATIVE QUADABSOLUTE) quadToTupleSequence ; quadToTupleSequence : quadToTuple quadToTupleSequence ","? quadToTuple ; quadToTuple : coordinatePair ","? coordinatePair ; smoothQuadTo : (SQUADRELATIVE SQUADABSOLUTE) coordinatePairSequence ; coordinatePairSequence : coordinatePair coordinatePairSequence ","? coordinatePair ; coordinatePair : coordinate ","? coordinate ; coordinateSequence : coordinate coordinateSequence ","? coordinate ; coordinate : NUMBER ;</pre>

Tabela 37. Léxico para definição de caminhos.

Léxico

MOVERELATIVE : "m" ;
MOVEABSOLUTE : "M" ;
CLOSEPATH : "z" | "Z" ;
LINERELATIVE : "l" ;
LINEABSOLUTE : "L" ;
HLINERELATIVE : "h" ;
HLINEABSOLUTE : "H" ;
VLINERELATIVE : "v" ;
VLINEABSOLUTE : "V" ;
CURVERELATIVE : "c" ;
CURVEABSOLUTE : "C" ;
SCURVERELATIVE : "s" ;
SCURVEABSOLUTE : "S" ;
QUADRELATIVE : "q" ;
QUADABSOLUTE : "Q" ;
SQUADRELATIVE : "t" ;
SQUADABSOLUTE : "T" ;
NUMBER : ("-" | "+")? DIGIT+ "." DIGIT+ (("e" | "E") ("-" | "+")? DIGIT+)? ;
DIGIT : [0-9] ;

Anexo VII. Modelo da Notação Gráfica ASML

Este anexo apresenta um modelo de detalhe adicional como complemento à definição da notação gráfica apresentada no capítulo 3 da segunda parte. A notação gráfica da ASML é uma especialização dos elementos de interação que compreende uma sintaxe concreta e uma semântica própria. Todo o modelo assenta em construções sintáticas válidas dos elementos de interação, pelo que qualquer instância deste modelo é também uma instância desse modelo, podendo ser manipulada de forma idêntica por uma ferramenta de engenharia.

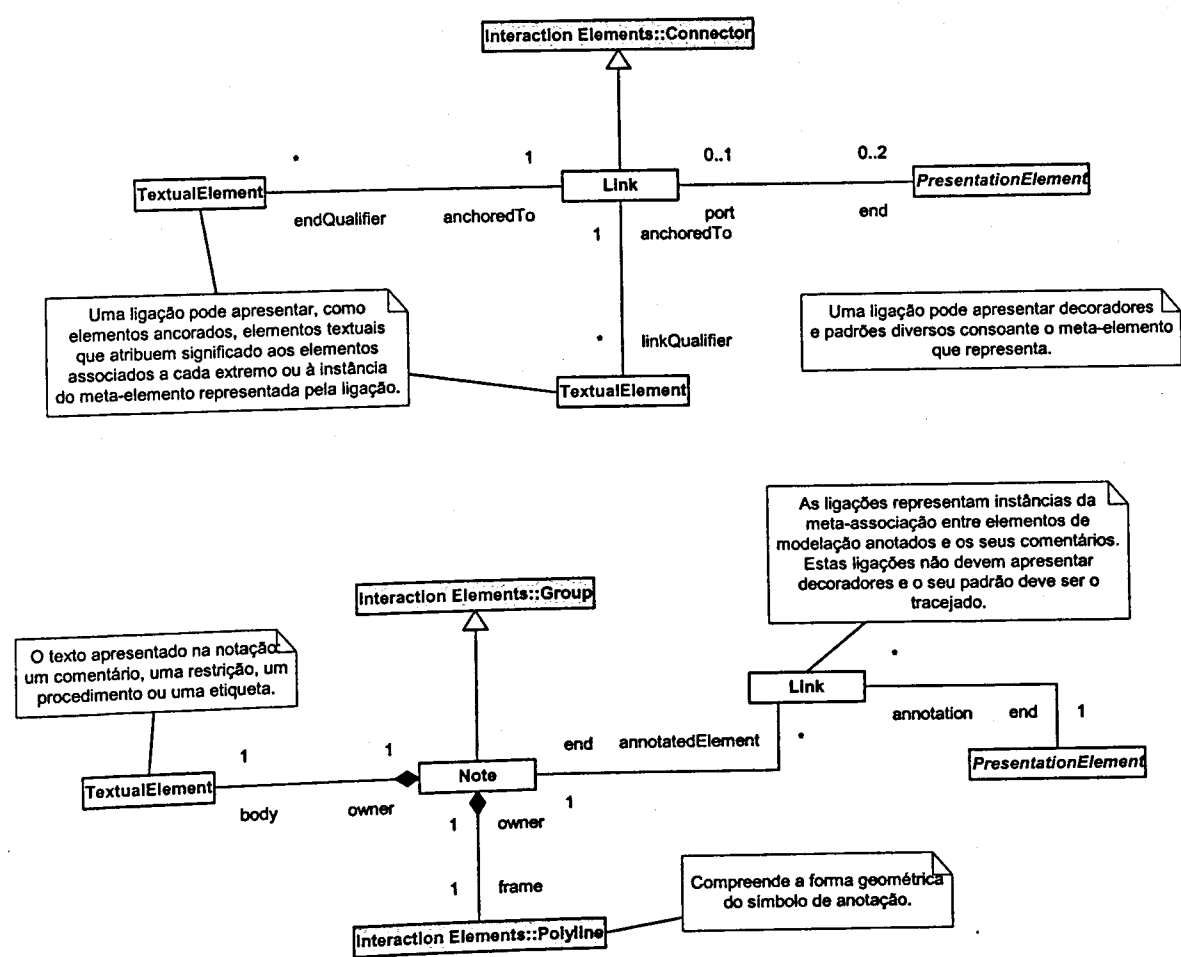


Figura 94. Elementos comuns da notação gráfica (Link, Note).

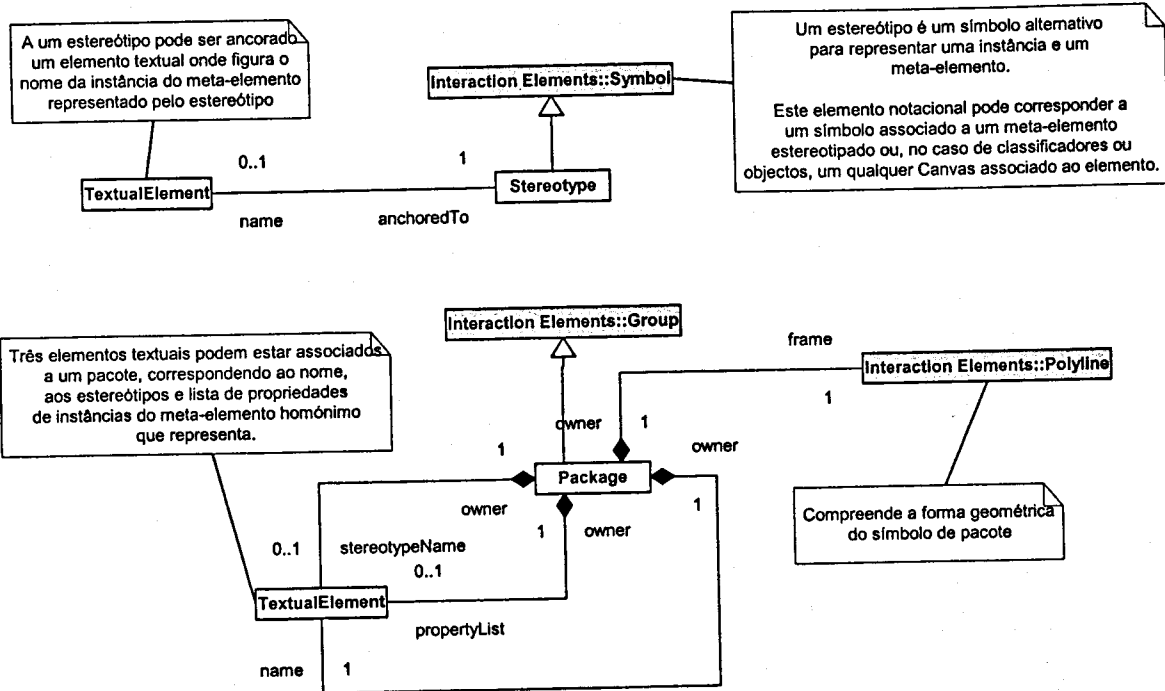


Figura 95. Elementos comuns da notação gráfica (*Stereotype, Package*).

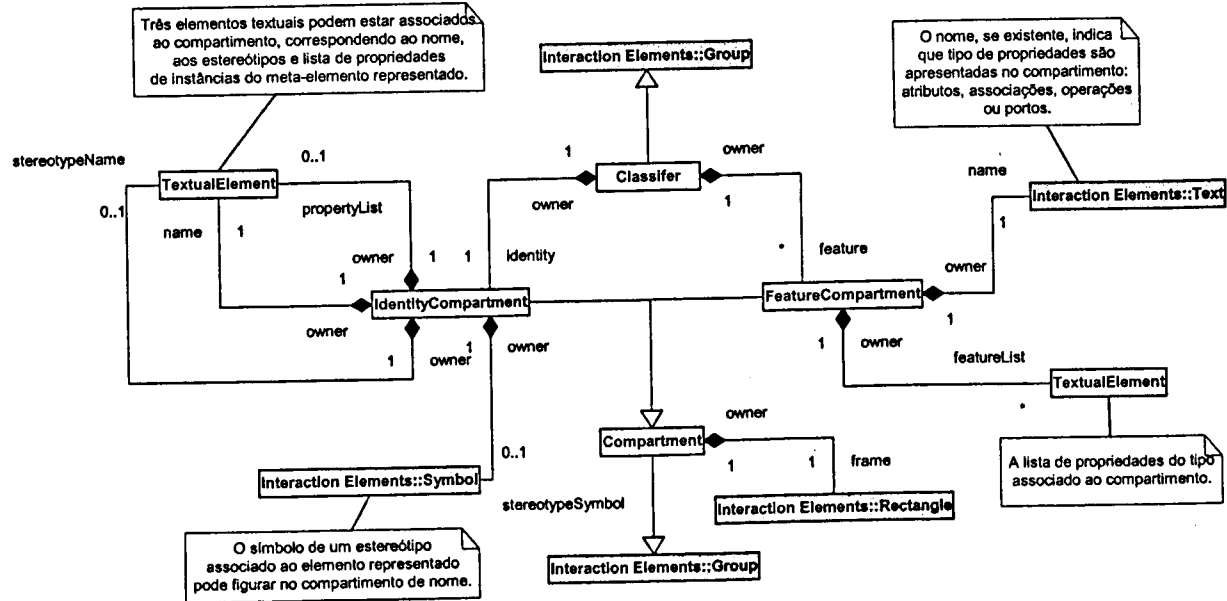


Figura 96. Representação de classificadores.

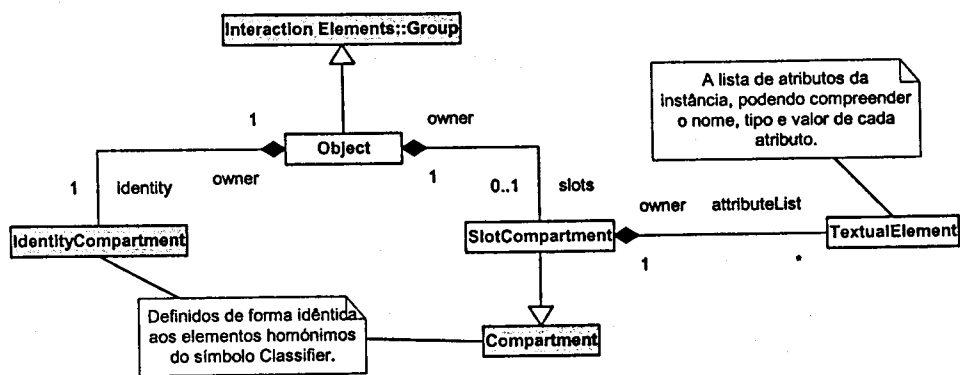


Figura 97. Representação de objectos.

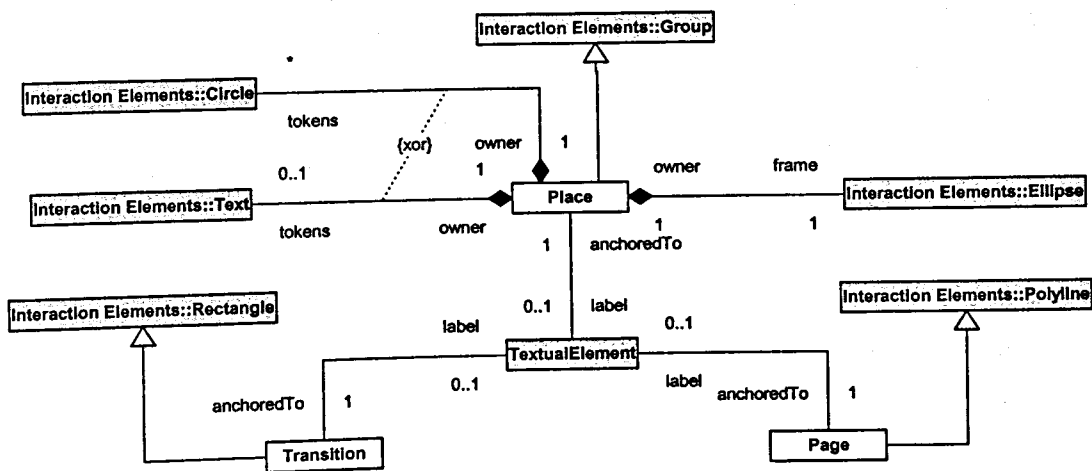


Figura 98. Representação de redes de Petri.

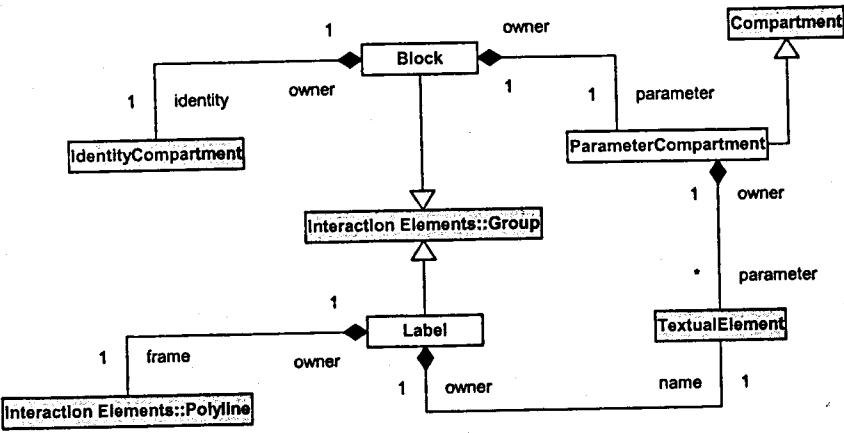


Figura 99. Representação de expressões e procedimentos.

Anexo VIII. Classes de Dados Comuns

Este anexo apresenta um conjunto de classes de dados comuns que constituem definições de componentes elementares de modelos de sistemas de automação. Este modelo de classes de dados comuns é baseado nos modelos apresentados em [44] e [45]. Uma vez que os documentos nos quais se baseia este modelo não estão, à data da escrita deste documento, aprovados, o modelo deverá ser alvo de revisão de forma a estar alinhado com a versão definitiva das referidas normas. Os princípios fundamentais deste modelo são contudo aplicáveis.

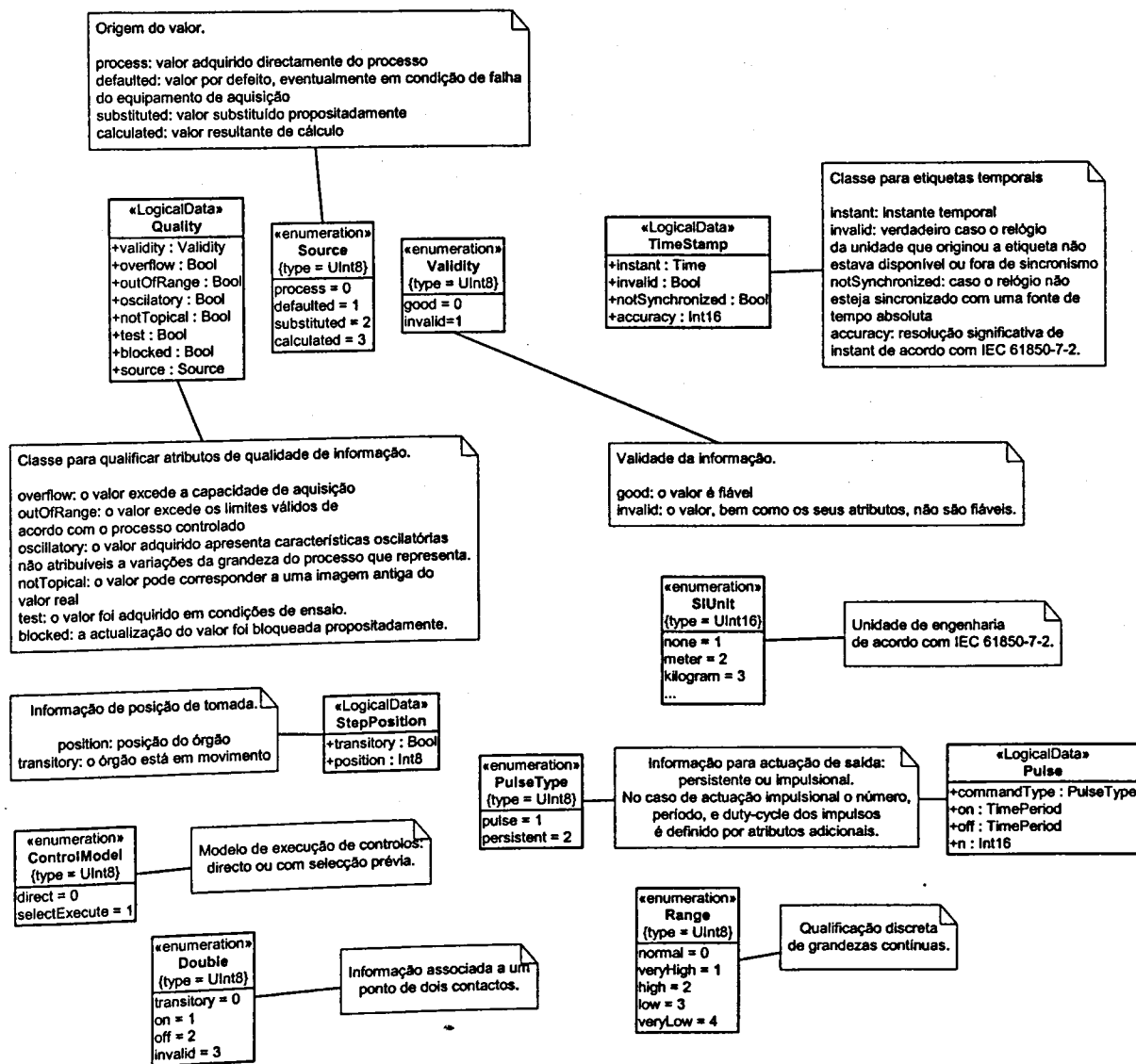


Figura 100. Classes de dados elementares para definição de classes de dados comuns.

A figura 101 apresenta classes de dados comuns para informação de estado. O modelo compreende apenas classes para valores instantâneos, podendo vir a ser melhorado com a introdução de classes específicas para informação trifásica absoluta ou interfásica simultânea, activação e disparo de protecções, sequências curtas de valores amostrados, informação de harmónicos ou informação estatística elementar. O modelo de acesso a este tipo de pontos de informação compreende apenas leitura ou escrita, modelada por acesso directo a atributos.

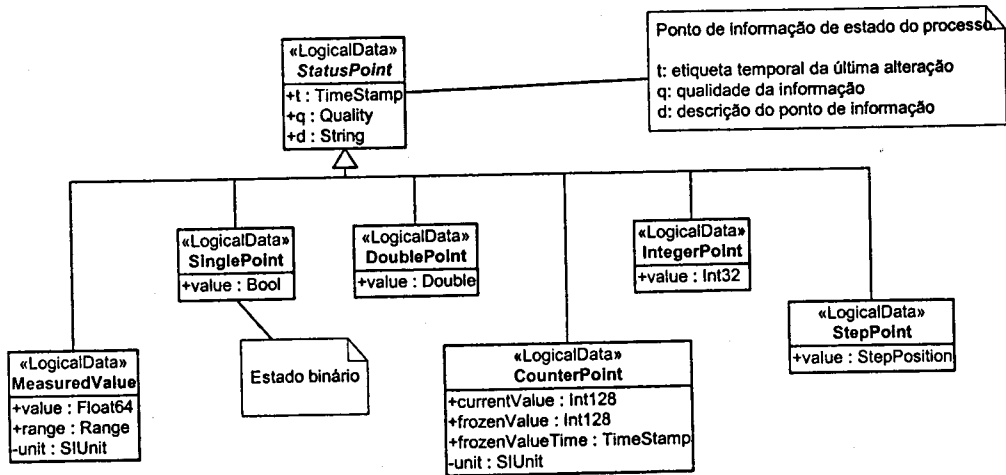


Figura 101. Classes de dados comuns para informação de estado.

Os pontos de informação de controlo são apresentados no modelo da figura 103. O modelo compreende apenas pontos controláveis convencionais. Os modelos de controlo suportados são o controlo directo e o controlo com selecção, modelados com recurso a portos e sinais (figura 102).

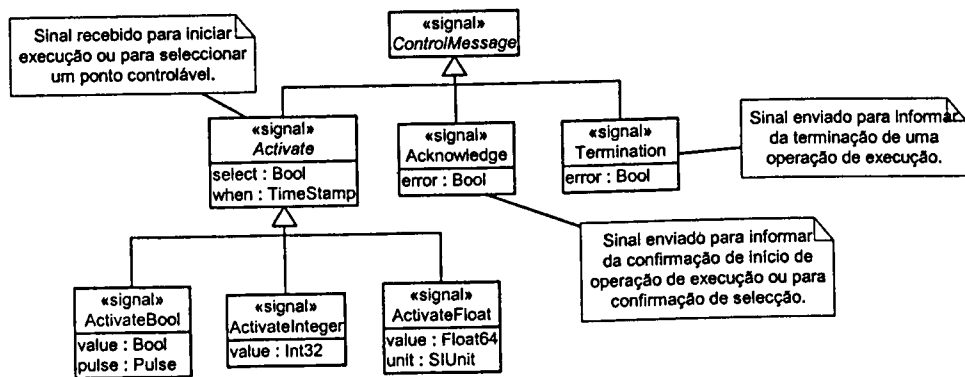


Figura 102. Sinais do modelo de controlo.

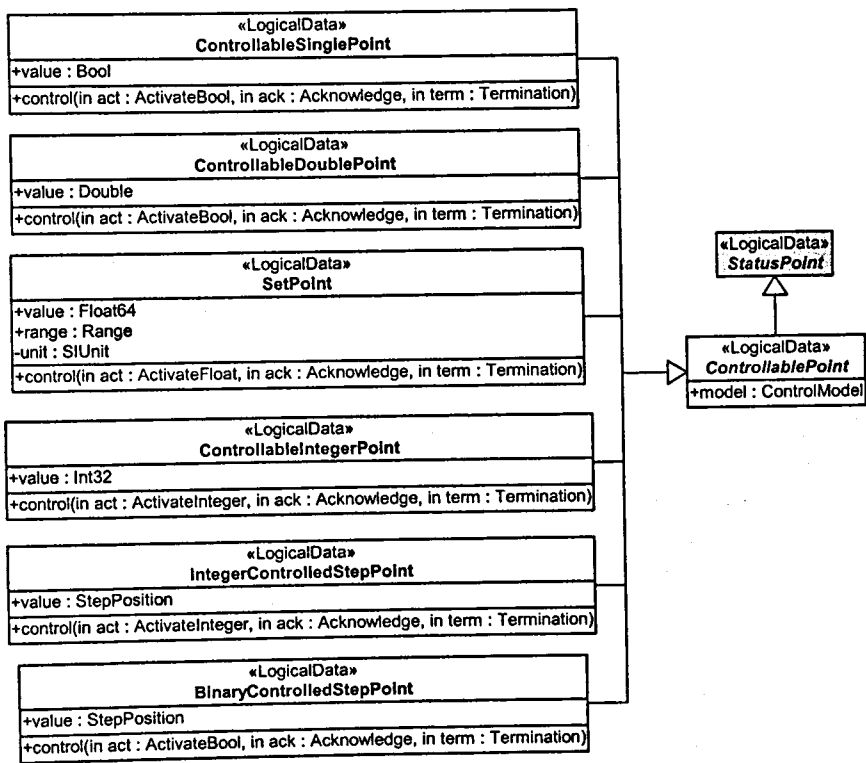


Figura 103. Classes de dados comuns para informação de controlo.

Em [45] e [7] são apresentados modelos de interface de nós lógicos com os quais o modelo apresentado neste anexo é compatível e que podem ser utilizados como base para a modelação de funções de controlo automático, controlo de supervisão, do equipamento secundário, do equipamento primário, de protecção, de medida e de sensorização, entre outros.

Anexo IX. Modelos dos Protocolos IEC 60870-5-101/104

Este anexo apresenta um exemplo limitado de modelação de dois protocolos de telecontrolo vulgarmente aplicados na automação do sistema eléctrico, o protocolo IEC 60870-5-101 [64] e o protocolo IEC 60870-5-104 [65]. O protocolo IEC 60870-5-101 é um protocolo de telecontrolo convencional suportado por sistemas de comunicações série assíncronos. Este protocolo suporta a veiculação assíncrona de eventos datados, a actualização de estado, a execução de controlos, a transferência de parâmetros e de ficheiros e a sincronização de relógios. O protocolo IEC 60870-5-104 é uma variante do IEC 60870-5-101 sobre TCP/IP. O modelo de dados destes protocolos é simples, sendo composto por pontos de tipos elementares (booleano, inteiros, vírgula flutuante), identificados por endereços de tipo inteiro e aos quais se associam atributos de qualidade.

Um modelo ASML para estes protocolos deve suportar o mapeamento do modelo de dados ASML para o seu próprio modelo, uma vez que não suportam algumas das características dos protocolos de terceira geração, como por exemplo: tipos estruturados e cadeias de caracteres, organização hierárquica do modelo de dados e recuperação do modelo de dados. No caso de protocolos de terceira geração, a introdução de mapeamento de dados será eventualmente desnecessária ou as suas características muito limitadas.

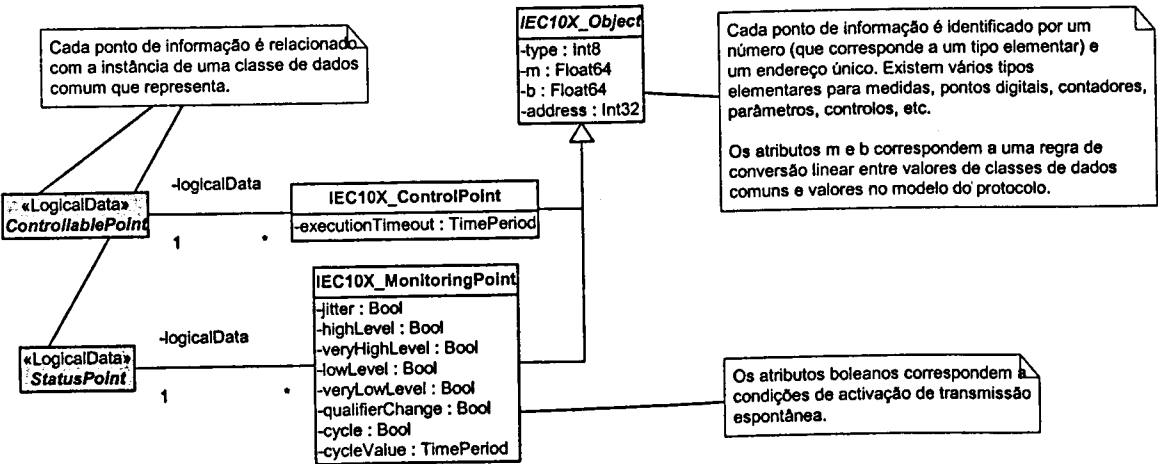


Figura 104. Modelo de dados para os protocolos IEC 60870-5-101/104.

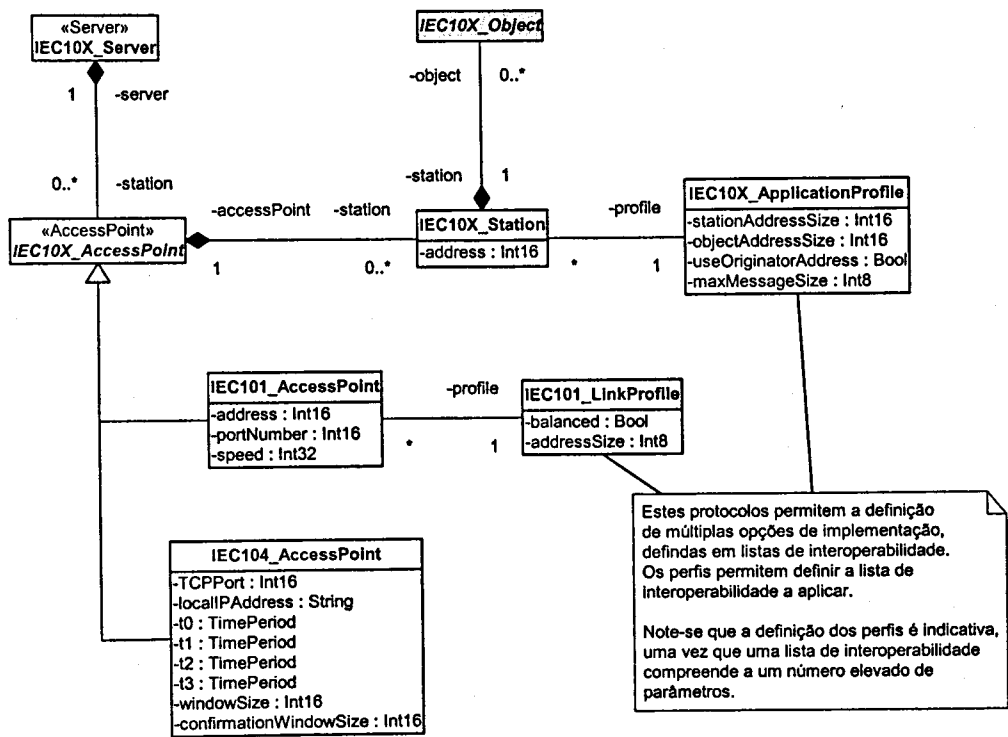


Figura 105. Modelo para servidores IEC 60870-5-101/104.

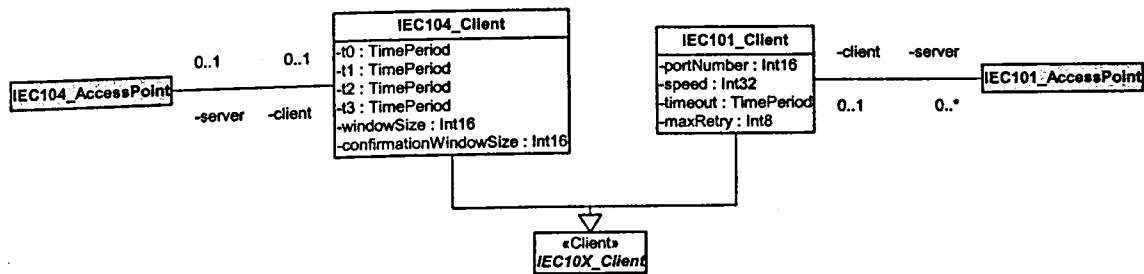


Figura 106. Modelo para clientes IEC 60870-5-101/104.

Anexo X. Exemplo de Modelo de um IED

Este anexo apresenta um exemplo de um modelo de um IED fictício. O IED corresponde a uma URT modular, sem HMI, com capacidade comunicativa IEC 60870-5-101 e com possibilidade de execução de algoritmos de controlo. Este modelo ASML pressupõe os modelos dos anexos VIII e IX e compreende um modelo para parametrização e configuração interna do IED, bem como um modelo lógico do próprio IED para integração em modelos lógicos de sistemas de automação.

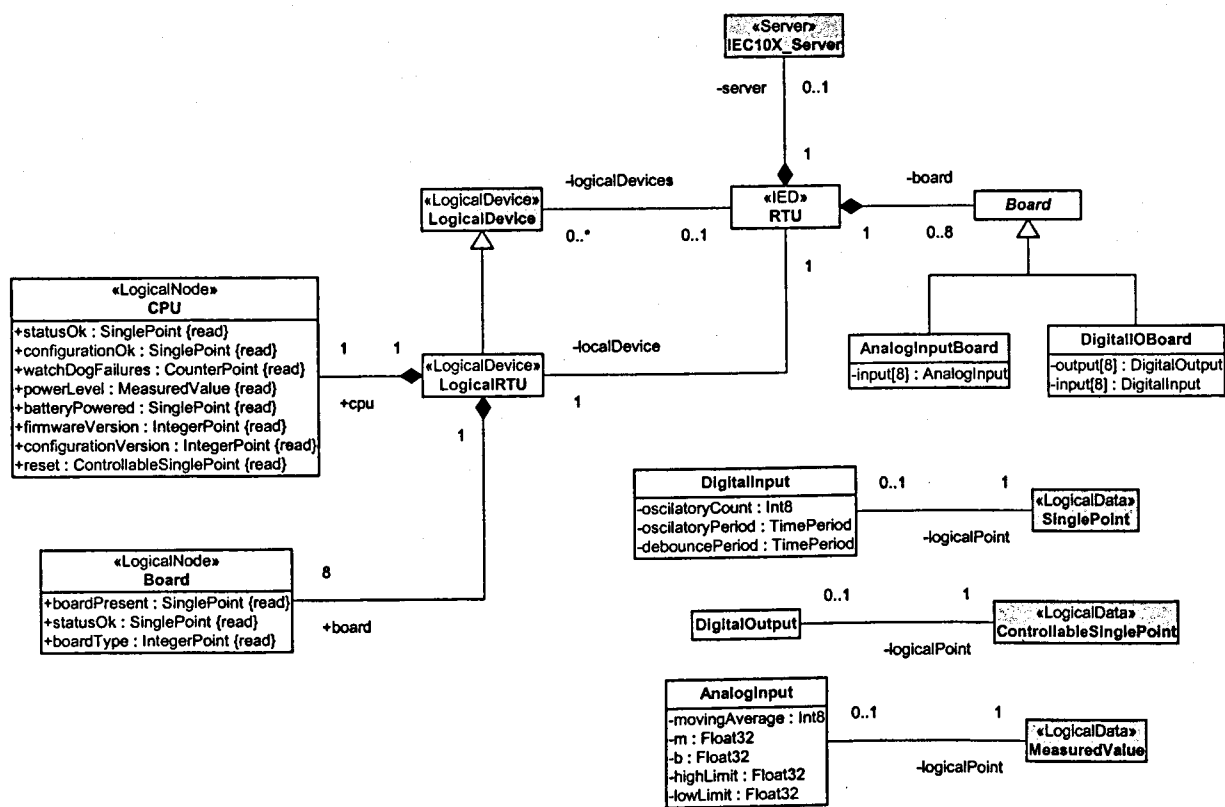


Figura 107. Exemplo de um modelo simples de um IED.





FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

BIBLIOTECA



0000073417