FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Ambiente gráfico de modelação para DSL

Tiago Daniel Ferreira da Silva Monteiro

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Ana Paiva (Doutora)

16 de Julho de 2012

Ambiente gráfico de modelação para DSL

Tiago Daniel Ferreira da Silva Monteiro

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Raul Fernando de Almeida Moreira Vidal
Arguente: José Francisco Creissac Freitas Campos
Vogal: Ana Cristina Ramada Paiva Pimenta
16 de Julho de 2012

Resumo

A utilização de interfaces gráficas nas aplicações informáticas é vista como indispensável, nos dias que correm, no entanto, o teste destes componentes ainda não está ao nível do teste dos outros componentes das aplicações informáticas, sendo feito de forma essencialmente manual.

Embora existam abordagens que procurem automatizar este processo, existe possibilidade de melhorar esta área, tornando-a ainda mais automática e sistematizada, por exemplo, através de teste baseado em modelos. Melhorar o processo de teste poderá fomentar uma melhoria da qualidade das interfaces gráficas e por consequência a melhoria do software em geral.

Um dos problemas de teste baseado em modelos é o esforço necessário para a construção desse modelo. Este projeto visa aumentar a reutilização nos modelos diminuindo o esforço na sua construção utilizando padrões de comportamento dos componentes das interfaces gráficas.

Para tal criou-se uma linguagem específica do domínio que visa facilitar a construção dos modelos em causa. No entanto, para a utilização desta linguagem torna-se necessário proceder à criação de um ambiente de modelação que permita construir modelos que respeitem as regras da linguagem e gerar casos de teste a partir dos modelos construídos. Este trabalho desenvolve o ambiente de modelação para a construção de modelos de GUIs com base na linguagem definida.

Durante a realização deste trabalho efetuou-se o estudo do estado da arte na área de teste de GUIs e o estudo de ferramentas de modelação existentes. No seguimento destes estudos, procedeu-se à criação do ambiente de modelação mencionado anteriormente e fizeram-se alguns testes através de casos de estudo.

Abstract

Nowadays, Graphical User-Interfaces are seen as an indispensable part of a computer application, however, the techniques used to test it aren't at the same level as those used to test other components, once GUI testing is essencially manual.

Although there are approaches that look to automate this process, there is the possibility to improve in this area, making it even more automatic and systematic, especially through model-based testing. These improvements will, ultimatelly, allow an improvement in GUI development and consequently in software development in general.

One of the problems of model-based testing is the effort neccessary to create this model. This project aims to increase the reuse in models decreasing the effort to construct them by using behavioural patterns of some GUI components.

With this in mind, a domain specific language was created to facilitate the construction of these models. But to use this language it is essential to create a modeling environment that allows the construction of models that respect this language rule's and that allows test case generation based on the models created.

During the execution of this work two studies were conducted: one on the state of the art in the GUI testing area and another on existing modeling tools. Following those studies, the modeling environment mentioned earlier was implemented and some case studies were made to test it.

Agradecimentos

Em primeiro lugar gostaria de agradecer à minha orientadora, Professora Ana Paiva, pelo constante apoio, acompanhamento e entusiasmo prestado ao longo de todo o desenvolvimento deste projeto.

À minha namorada, Cíntia Almeida, agradeço pela paciência e pelo amor que me tem devotado durante estes meses em que, devido ao trabalho, não lhe tenho dispensado a atenção e o carinho que ela tanto merece.

Aos meus amigos, Luís Silva, João Santos, Alexandre Perez e Francisco Silva, pela amizade e camaradagem ao longo deste divertido, mas nem sempre fácil percurso que foi o Mestrado Integrado em Engenharia Informática e Computação.

Quero também agradecer à minha família. À minha mãe, Maria da Luz, por todo o carinho e doçura. Ao meu pai, Carlos, por me fazer sempre procurar ser melhor e não me contentar com o que já tinha alcançado. À minha irmã, Joana, pela sua traquinice e brincadeiras que sempre me ajudaram a ver as coisas de uma forma mais positiva. Ao meu segundo pai, Paulo, por acreditar sempre em mim e me mostrar que o amor não está no sangue, mas no convívio.

Finalmente, não posso deixar de agradecer aos meus avós, Francisco e Deolinda, pela educação, apoio , no fundo fazendo de mim o homem que sou hoje.

Tiago Daniel Ferreira da Silva Monteiro

Conteúdo

1 Introdução **1**
- 1.1 Contexto/Enquadramento . . . 1
- 1.2 Projecto . . . 1
- 1.3 Descrição do Problema . . . 2
- 1.4 Motivação e Objetivos . . . 2
- 1.5 Estrutura da Dissertação . . . 3

2 Revisão Bibliográfica **5**
- 2.1 Linguagem específica do domínio . . . 5
- 2.2 Teste baseado em modelos . . . 6
- 2.3 Análise de Ferramentas . . . 8
  - 2.3.1 Processo de análise . . . 8
  - 2.3.2 Ferramentas analisadas . . . 9
  - 2.3.3 Sumário . . . 14
- 2.4 Resumo . . . 15

3 PARADIGM - Modeling Environment **17**
- 3.1 Linguagem . . . 17
- 3.2 Requisitos . . . 19
- 3.3 Cenários de utilização . . . 19
- 3.4 Geração de caminhos de teste . . . 27
- 3.5 Modelos de Criação . . . 28
  - 3.5.1 Modelo do domínio . . . 28
  - 3.5.2 Modelo gráfico . . . 29
  - 3.5.3 Modelo de ferramentas . . . 30
  - 3.5.4 Modelo de mapeamento . . . 31
- 3.6 Ambiente . . . 34
- 3.7 Resumo e Conclusões . . . 40

4 Casos de estudo **41**
- 4.1 Standvirtual . . . 41
- 4.2 ERA . . . 47
- 4.3 Resumo . . . 53

5 Conclusões e Trabalho Futuro **55**
- 5.1 Satisfação dos Objetivos . . . 55
- 5.2 Trabalho Futuro . . . 56

Referências **57**

CONTEÚDO

Lista de Figuras

1.1 Exemplo de modelo 3

2.1 GMF Overview [GT06] 9
2.2 Arquitetura do ambiente de modelação StarUML [LKKL05] 10
2.3 Arquitetura do ambiente de modelação Open Modelshpere [Gra08] 12
2.4 Arquitetura do ambiente de modelação ArgoUML [RR00] 13
2.5 Interface de extensão do ArgoUML [LSTM04] 13

3.1 Meta Modelo da Linguagem 17
3.2 Casos de uso 20
3.3 Diagrama do domínio 29
3.4 Modelo gráfico 30
3.5 Modelo gráfico (detalhe) 30
3.6 Modelo de ferramentas 31
3.7 Modelo de mapeamento 32
3.8 Propriedades do mapeamento do nó Input 32
3.9 Listagem de regras 33
3.10 Ambiente de modelação 35
3.11 Modelo Exemplo 36
3.12 Diálogo *"Find entries"* 36
3.13 Diálogo para adicionar um *"find entry"* 37
3.14 Diálogo para adicionar *"Field/Value entries"* 37
3.15 Código de identificação da propriedade a ser chamada 38
3.16 Menu para geração de caminhos de teste 39

4.1 Página inicial do sítio standvirtual.com 42
4.2 Modelo da aplicação standvirtual 42
4.3 Ecrã de apresentação de resultados da pesquisa da aplicação standvirtual 43
4.4 Modelo do formulário Resultados 43
4.5 Modelo do formulário Conteúdo 44
4.6 Configurações do nó "Pesquisa [1]" 45
4.7 Configurações do nó "Lista de Inputs [2.2]" 45
4.8 Alerta de erros no modelo 47
4.9 Página inicial do sítio ERA.pt 48
4.10 Modelo da aplicação ERA.pt 48
4.11 Detalhe do formulário de pesquisa da aplicação ERA 49
4.12 Página de resultados da aplicação ERA.pt 50
4.13 Detalhe do formulário de resultados da aplicação ERA 50
4.14 Configurações do nó "Tipo Imóvel [1.1]" 51

4.15 Configurações do nó "Preço [1.4]" 51
4.16 Configurações do nó "Distrito/Concelho [1.5]" 52
4.17 Configurações do nó "Ordenação [2.1]" 52
4.18 Diálogo de erro por falta do nó *"Start"* 53

Lista de Tabelas

2.1 Critérios de avaliação para as ferramentas analisadas 8
2.2 Sumário GMF 10
2.3 Sumário StarUML 11
2.4 Sumário Open Modelsphere 12
2.5 Sumário ArgoUML 14
2.6 Resumo comparativo dos ambientes estudados 14

3.1 Caso de uso "Criar projeto" 21
3.2 Caso de uso "Criar nó" 22
3.3 Caso de uso "Criar relação" 23
3.4 Caso de uso "Verificar integridade do modelo" 24
3.5 Caso de uso "Configurar dados de teste" 25
3.6 Caso de uso "Gerar caminhos de teste" 26
3.7 Caso de uso Detalhar formulário" 27

Abreviaturas e Símbolos

DSL	*Domain Specific Language*
GMF	*Graphical Modeling Framework*
GUI	*Graphical User Interface*
MBT	*Model-Based Testing*
OCL	*Object Constraint Language*
SWT	*Standard Widget Toolkit*
UML	*Unified Modeling Language*

Capítulo 1

Introdução

Este primeiro capítulo apresenta o contexto do trabalho de investigação, o projecto do qual faz parte, por que motivo surge e quais os objetivos que se propõe atingir.

1.1 Contexto/Enquadramento

As interfaces gráficas (GUIs) nas aplicações informáticas são atualmente uma forma de interação humano-computador largamente utilizada. A criação de uma GUI é quase indispensável na criação de uma aplicação informática, no entanto, o teste de GUIs continua a ser uma tarefa essencialmente manual [Mem02], apresentando-se como uma tarefa demorada e complicada devido ao considerável número de interações possíveis. Apesar de possivelmente apresentar um número alto de interações, a maioria das GUIs apresentam elementos com padrões de comportamento similares e portanto o seu comportamento torna-se previsível [vW12]. É com base na necessidade de automatizar o processo de teste e na previsibilidade das GUIs que surge o projeto "Pattern-Based GUI Testing" do qual este trabalho faz parte.

1.2 Projecto

O projeto "Pattern-Based GUI Testing" [CPFA10] é um projeto criado pela Faculdade de Engenharia da Universidade do Porto (FEUP), com a colaboração da Universidade do Minho (UM) e da empresa TelBit e aprovado e financiado pela Fundação para a Ciência e Tecnologia (FCT).

Este projeto visa melhorar os métodos de teste de interfaces gráficas, contribuindo para a construção de uma ferramenta que permita automatizar a criação e execução de casos de teste sobre GUIs, através da técnica de teste baseado em modelos.

Um dos aspetos menos positivos do teste baseado em modelos prende-se com o esforço necessário para construir os modelos a partir dos quais se geram os testes.

Este projeto pretende, por um lado obter um modelo a partir da própria GUI através de engenharia reversa, e por outro lado permitir a construção ou alteração do modelo num nível de

abstração elevado através de um ambiente de modelação e de uma linguagem específica do domínio (DSL) com base nos padrões de comportamento da própria GUI, procurando assim minimizar o esforço de construção do modelo.

1.3 Descrição do Problema

Neste trabalho pretende-se criar um ambiente de modelação que permita criar/editar modelos de GUIs com base numa linguagem específica de domínio (DSL) e permita gerar casos de teste a partir desses modelos.

Este ambiente de modelação, em conjunto com a ferramenta de engenharia reversa desenvolvida no âmbito do projeto PBGT, permitirá a criação e manutenção de modelos de interfaces gráficas, que por sua vez levam à criação e manutenção de casos de teste de forma mais rápida e com menor esforço, do que se estes fossem criados de forma manual.

1.4 Motivação e Objetivos

Como descrito anteriormente, o teste de GUIs conforme atualmente é executado, trata-se de uma tarefa complicada e demorada. Assim torna-se importante procurar formas de automatizar o processo de teste e daí surge o projecto descrito na secção anterior. Inserido neste projeto este trabalho tem como objetivo a construção de um ambiente de modelação que permita reduzir o esforço de construção dos modelos necessários à geração de casos de teste. Este ambiente de modelação deverá permitir:

- Construir modelos com base numa DSL (exemplo na figura 1.1);
- Verificar a integridade do modelo construído;
- Configurar dados de teste;
- Gerar os caminho de teste a partir dos modelos criados.

Na procura de atingir os objetivos descritos chegou-se a algumas questões de pesquisa, nomeadamente:

- O que é uma DSL, para que serve e que passos são necessários para criar uma?
- O que significa teste baseado em modelos em geral? E mais concretamente no domínio das GUIs?
- Que ambientes de modelação existem atualmente?
- Dos ambientes que existem, qual seria o mais adequado para suportar modelação com base na DSL definida para modelação de GUIs?

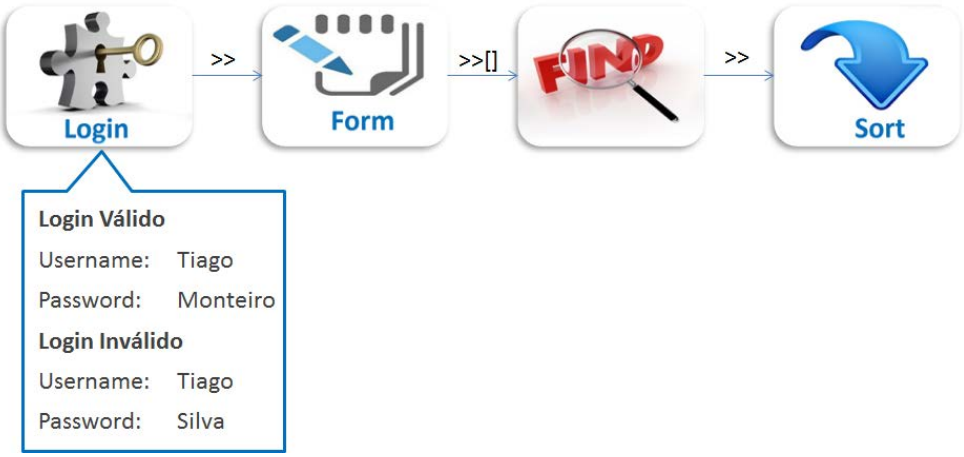


Figura 1.1: Exemplo de modelo

Quanto às duas primeiras questões procedeu-se a um estudo teórico do estado da arte. No caso da terceira e quarta perguntas conduziu-se um estudo comparativo de alguns ambientes. Com base no estudo efetuado escolheu-se uma das ferramentas para criar uma extensão da mesma que possibilitasse atingir os objetivos traçados.

1.5 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais 4 capítulos.

No capítulo 2 é descrito o estado da arte, começando por linguagens específicas do domínio, passando por teste baseado em modelos e terminando com uma análise de um conjunto de ferramentas do qual é escolhida uma para a fase de implementação.

O capítulo 3 apresenta uma pequena descrição da linguagem para a qual o ambiente é criado, bem como os requisitos e cenários de utilização. Ainda neste capítulo são também apresentados detalhes da implementação e o aspeto geral do ambiente de modelação.

No capítulo 4 são documentados pequenos casos de estudo que permitem aferir das potencialidades do sistema.

Finalmente, são expostas as conclusões do trabalho efectuado e é apontado o trabalho a efectuar no futuro no capítulo 5.

Capítulo 2

Revisão Bibliográfica

Este capítulo apresenta o estudo teórico de base, começando por uma breve análise sobre DSLs, seguindo-se uma secção sobre teste baseado em modelos, em particular quando aplicado a GUIs, e, por fim, uma análise de um conjunto de ferramentas.

No final do capítulo são apresentadas algumas reflexões sobre o estudo realizado.

2.1 Linguagem específica do domínio

Linguagem específica do domínio (DSL, do inglês *Domain Specific Language*) é uma linguagem de expressividade limitada adaptada para um domínio aplicacional particular [Hud98]. É uma linguagem que permite trabalhar com um número limitado de elementos dentro de uma área de conhecimento muito específica. Exemplos reconhecidos de DSL são: SQL (*Standard Query Language*), uma linguagem para interrogação de bases de dados relacionais e HTML (*Hyper-Text Markup Language*) [VKV00].

Segundo [FP10] é possível dividir DSLs em dois tipos: internas e externas. DSL externa é uma linguagem completamente separada da linguagem principal da aplicação com que a DSL trabalha, sendo "interpretada ou traduzida para um programa na linguagem da aplicação"[Cun08]. Ao contrário, uma DSL Interna utiliza código válido de uma linguagem generalista de uma forma particular. As DSL internas são também conhecidas como linguagens embutidas [Hud98].

A utilização de uma linguagem deste género deve ser ponderada, percebendo se realmente é benéfica. Tendo em conta as vantagens e desvantagens da utilização de DSL apontadas em [VKV00] pode perceber-se que o seu contexto de utilização deve limitar-se a ambientes em que se garanta uma considerável escala de utilização, para que compense o esforço necessário ao desenho, implementação e manutenção da mesma. Para além disso é também importante que a linguagem seja tão simples quanto possível para diminuir o esforço de aprendizagem dos seus utilizadores.

Para além da divisão em DSL internas ou externas, pode também considerar-se uma divisão pelo tipo de notação: visual ou textual. O que isto significa é que existem DSL em que é necessário escrever texto, enquanto existem outras DSL cujos programas ou modelos se criam através de diagramas ou representações gráficas.

Em [EEHT05] e [EJ01] apresentam-se exemplos de definições de linguagens visuais. A definição de DSL visuais passa essencialmente por três aspetos: definir os conceitos do domínio, ou seja definir os nós e as arestas que os ligam; a notação a utilizar na representação gráfica, ou seja a imagem associada a cada nó e aresta; e finalmente, um conjunto de regras que permite definir a forma como os nós e as arestas se ligam.

2.2 Teste baseado em modelos

Teste baseado em modelos (MBT) é definido por [UL07] como uma técnica de teste de software em que, a partir de um modelo que descreve o sistema a testar, se geram casos de teste. Ainda segundo [UL07] destacam-se quatro abordagens a MBT:

- Geração de dados de entrada a partir de um modelo do domínio - modela-se os domínios dos valores de entrada e geram-se testes por combinação desses valores.

- Geração de casos de teste a partir de um modelo de ambiente - modela-se o que se espera que seja o ambiente em que o sistema vá ser utilizado. A partir deste modelo criam-se sequências de chamadas ao sistema para tentar validar o seu comportamento.

- Geração de casos de teste com *oracles* a partir de modelos comportamentais - nesta abordagem é possível gerar casos de teste executáveis com informação dos resultados esperados incluída (*oracles*). Para tal é necessário modelar o comportamento esperado do sistema, o que nem sempre é fácil, uma vez que dependendo da complexidade do sistema pode ser complicado saber a relação entre dados de entrada e dados de saída.

- Geração de scripts de teste a partir de testes abstratos - esta abordagem baseia-se na transformação de uma descrição abstrata de casos de teste (como um diagrama de sequência UML) num script de casos de teste executáveis .

Para além da geração automática de casos de teste, uma das principais vantagens de MBT, também a exaustividade dos testes gerados e a facilidade em adaptar o sistema a mudanças são mais-valias desta técnica [BBN04].

No entanto, também apresenta desvantagens. Desde logo a construção de um modelo pode requerer demasiado esforço e portanto não ser compensatório. Existe ainda o problema de explosão do espaço de estados, uma vez que os modelos podem levar a um número de estados completamente impossivel de gerir [EFW01].

Na área do teste de GUIs baseado em modelos, existem dois trabalhos na bibliografia e que serão aqui analisados, os trabalhos de Ana Paiva [PFTV05] e de Atif Memon [MBN03],[BM07].

O trabalho de Memon utiliza uma ferramenta para fazer engenharia reversa da GUI a testar e com isso, construir o modelo dessa GUI. Inicialmente esta ferramenta cria o que o autor chama de "GUI Forest"que é uma representação da estrutura das janelas (nós) e a sua relação hierárquica. Cada nó desta estrutura apresenta os seus componentes e respetivas propriedades e valores.

A partir dessa primeira representação é criado um grafo de fluxo de eventos (EFG). Com este grafo de fluxo de eventos e com perfis de utilização de utilizadores finais da aplicação são calculadas probabilidades para cada caminho existente no grafo de fluxo de eventos, criando um grafo de fluxo de eventos probabilístico (PEFG). Então, a partir deste grafo, são gerados casos de teste como sequências de eventos segundo uma de três abordagens possíveis: um evento altamente provável; sequências que consideram a probabilidade de todo o caminho a percorrer; sequências com os caminhos menos prováveis para encontrar erros que de outra forma dificilmente seriam detetados.

Ainda que com muitos méritos, o trabalho de Memon apresenta ainda alguns problemas. Baseando-se em traços de execução de utilizadores finais para criar o modelo probabilístico leva a que este trabalho seja particularmente apto para teste de regressão, no entanto, quando se pensa em desenvolvimento de software de raiz o trabalho de Memon acaba por esbarrar no problema de obter dados para os cálculos probabilísticos. Para além disso, sofre ainda de outro problema que tem que ver com o tipo de problemas que encontra. Baseando-se no fluxo de eventos na interface os erros que encontra são essencialmente erros fatais que representam a avaria do sistema.

Quanto ao trabalho de Paiva, trata-se de uma extensão da ferramenta Spec Explorer. Esta extensão permite o mapeamento entre as acções descritas no modelo e os objetos físicos (botões, caixas de texto, etc.) da GUI a testar.

O processo começa pela modelação da GUI com recurso à linguagem Spec#, com métodos que modelam as ações do utilizador e variáveis de estado que modelam o estado da GUI. A partir deste modelo utiliza-se Spec Explorer para a geração de um conjunto de casos de teste. Esta geração faz-se através de dois passos: primeiro é gerada uma máquina de estados finita (FSM, do inglês *Finite State Machine*) por exploração limitada do espaço de estados; de seguida casos de teste que cumpram o critério de cobertura escolhido são gerados a partir da FSM do passo anterior.

Para tornar possível a automatização da execução de testes, é necessária a criação de código que simule as acções do utilizador e é então que entra em ação a ferramenta desenvolvida de raiz. Esta ferramenta gera este código automaticamente com base num mapeamento efetuado pelo testador entre as ações do modelo e os controlos da GUI onde as ações ocorrem. O mapeamento é efetuado através de um front-end da ferramenta em que se seleciona a ação do modelo que se quer mapear e depois se escolhe o objeto físico da interface a que a ação corresponde através de uma ferramenta chamada "GUI Spy Tool".

Com o código de mapeamento gerado, basta depois compilá-lo como uma biblioteca e referenciar essa biblioteca no projeto de Spec Explorer. Podem executar-se os testes automaticamente, sem intervenção do testador, e esperar o relatório de erros gerado comparando o resultado do teste com o resultado esperado pelo modelo.

Sendo uma abordagem que permite descobrir diversos tipos de erros, este trabalho apresenta um problema: o esforço necessário para a construção do modelo em Spec# é demasiado.

2.3 Análise de Ferramentas

Para proceder ao trabalho de criação de um ambiente de modelação para a linguagem específica de domínio existente é necessário primeiramente escolher uma ferramenta que permita a criação dos elementos da DSL ou uma ferramenta de modelação que possa ser estendida/adaptada com as funcionalidades pretendidas. Para tal procedeu-se a uma fase de análise de várias aplicações de modelação existentes por forma a perceber qual a mais indicada para ser utilizada durante a fase de implementação do projecto. Estas ferramentas foram estudadas segundo uma série de parâmetros e comparadas para que fosse possível escolher uma para desenvolver o trabalho pretendido.

A secção termina com uma breve reflexão sobre as ferramentas analisadas e a escolha da ferramenta a utilizar para a implementação.

2.3.1 Processo de análise

Para uma análise cuidada das ferramentas foi necessário começar por definir uma lista de critérios sobre os quais as ferramentas seriam avaliadas. Na tabela 2.1 são apresentados os critérios, bem como uma pequena descrição do que significa cada um. Nos critérios em que se avalia o grau de possibilidade, utiliza-se uma escala de classificação entre 1 e 4, sendo 1 não é possível e 4 é totalmente possível. Estes critérios foram definidos tendo em conta os requisitos pretendidos para o ambiente de modelação.

possibilidade de criação/extensão	possibilidade de criar ou estender novas funcionalidades.
integração com outros ambientes	a possibilidade de integrar a ferramenta com outros ambientes de desenvolvimento.
definição de propriedades para elementos	possibilidade de definir propriedades para configurar elementos da linguagem.
definição de regras	possibilidade de definir regras de construção da linguagem para garantir integridade do modelo.
gravar modelos em XML	possibilidade de guardar o modelo construído num ficheiro com formato XML.
cross-platform	utilização em várias plataformas, nomeadamente em termos de sistemas operativos.
projeto activo	indica se o projeto ainda apresenta atualizações.

Tabela 2.1: Critérios de avaliação para as ferramentas analisadas

Com estes parâmetros em mente procedeu-se então à análise de algumas aplicações através da pesquisa de documentação, mas também fazendo uso de alguma experimentação com as mesmas. Na próxima secção são apresentadas as ferramentas estudadas.

2.3.2 Ferramentas analisadas

Eclipse Graphical Modeling Framework

Graphical Modeling Framework (GMF) foi lançada no ano 2006 e faz parte do *Eclipse Modeling Project*. Com diversas melhorias desde o lançamento, esta ferramenta permite a criação de editores gráficos para linguagens de modelação.

O processo de criação da DSL e do respetivo editor pode ser visto na figura 2.1.

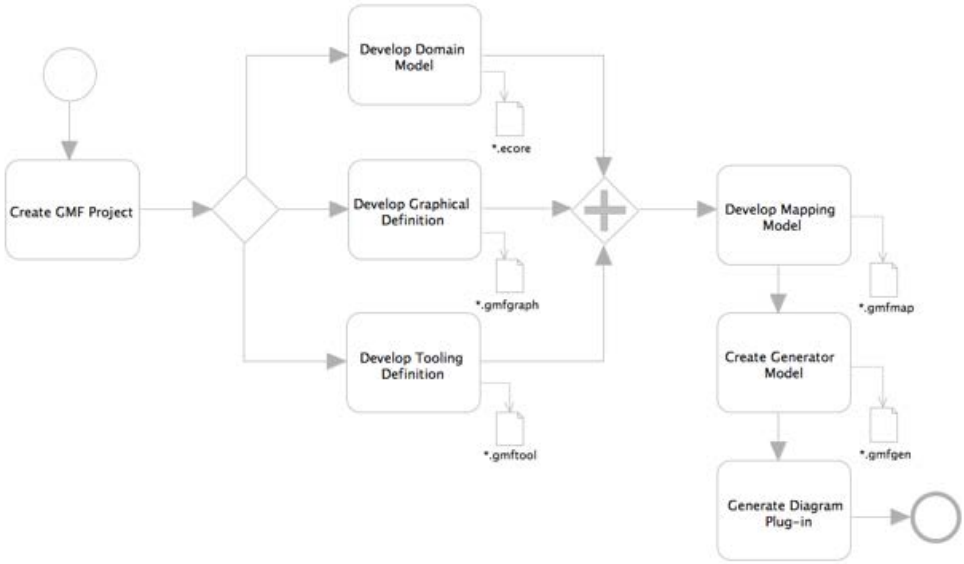


Figura 2.1: GMF Overview [GT06]

Este processo começa pela criação do modelo do domínio que define todos os elementos do domínio, nós, ligações e respetivas propriedades. De seguida, trata-se do desenvolvimento da definição gráfica que define o aspecto das figuras, dos nós e das ligações, da definição das ferramentas que define os menus, as ações, as barras de ferramentas, etc. e do modelo de mapeamento que especifica as relações entre os elementos do domínio, os elementos gráficos e os elementos das ferramentas. Procede-se ainda à criação do modelo de geração que especifica os parâmetros de geração de código do ambiente de modelação e então gera-se finalmente o ambiente e todas as ferramentas associadas.

A definição de regras para validação do modelo pode ser feita através de OCL ou Java [Gro09].

Visto pertencer ao projeto Eclipse tem suporte e documentação consideráveis e é uma ferramenta em constante evolução.

Um possível senão desta ferramenta, segundo a experimentação efetuada, prende-se com a performance, uma vez que para modelos mais complexos parece tornar-se um pouco lento.

facilidade de criação	4
integração com outros ambientes	Eclipse
definição de propriedades para elementos	4
definição de regras	3 (OCL ou Java)
gravar modelos em XML	Sim
cross-platform	Sim
projeto activo	Sim

Tabela 2.2: Sumário GMF

StarUML

StarUML é um ambiente de modelação que suporta arquitetura baseada em modelos e baseia-se na notação UML versão 1.4, aceitando também notação UML versão 2.0.

É possível estender este ambiente através da criação de módulos. A criação de um módulo possibilita o suporte a processos ou linguagens de programação específicos, a integração com outras ferramentas, extensão para outras funcionalidades ou criação de ambientes individuais específicos [LKKL05].

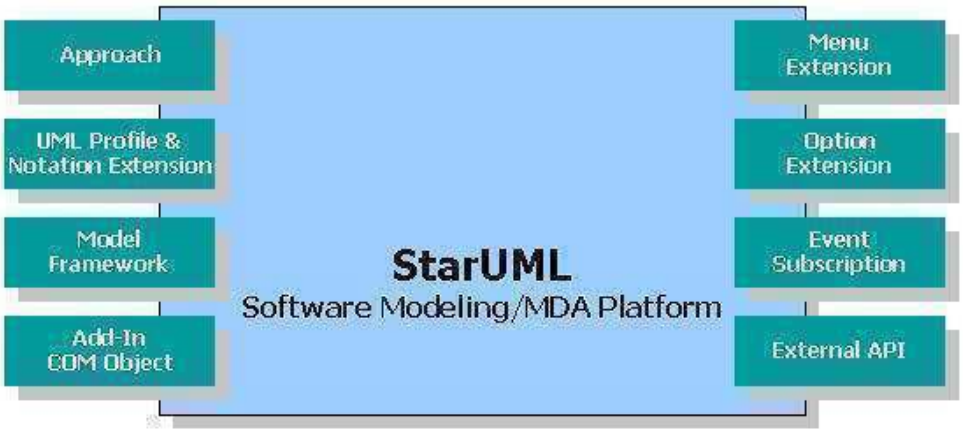


Figura 2.2: Arquitetura do ambiente de modelação StarUML [LKKL05]

Na figura 2.2, pode ver-se a organização do ambiente, sendo que na parte central surge a plataforma principal e os outros elementos são as partes extensíveis, que juntas definem o tal módulo que se cria quando se pretende estender as funcionalidades. Numa breve descrição das partes extensíveis:

- *Approach* - permite definir o modelo do projeto e a organização dos diagramas.
- Perfis e Notação UML - extensão para adaptar modelos UML para um domínio específico.
- *Model Framework* - permite a reutilização de modelos de software.
- Extensão de Menus - criação de novos menus.

- Extensão de opções - criação de novas opções.
- Subscrição de eventos - permite a criação de notificações para eventos da ferramenta.
- API externa - permite estender a API da ferramenta.

facilidade de extensão	2
integração com outros ambientes	Possível
definição de propriedades para elementos	3
definição de regras	3 (OCL)
gravar modelos em XML	Sim
cross-platform	Apenas Windows
projeto activo	Desativado (última versão: 2005)

Tabela 2.3: Sumário StarUML

Esta aplicação apresenta como pontos positivos as capacidades de modelação e suporte a arquitetura baseada em modelos permitindo até geração de código.

Como pontos negativos apresentam-se a necessidade de conhecer bem a arquitetura da aplicação para a estender e principalmente o facto de ser já um projeto desativado cuja última versão data de 2005.

Open Modelsphere

O Open Modelsphere é um ambiente de modelação, completamente escrito em Java, lançado com este nome em 2002. Esta ferramenta permite modelação de processos de negócio, modelação de dados e modelação UML [Gra09].

Este ambiente apresenta a arquitetura representada na figura 2.3 e baseia-se em três camadas: JACK (*Java Abstract Class Kit*), SMS (*Shared Modelling Software*) e Plug-ins. Nesta arquitetura a camada superior depende sempre das camadas abaixo, mas as camadas debaixo nunca dependem das camadas acima, isto é, a dependência é só num sentido (como indicam as setas na figura).

A camada de baixo, JACK é uma camada independente que pode ser usada para construir outras aplicações que não o Modelsphere. Esta camada utiliza métodos da biblioteca padrão de Java e quando estes métodos não fazem exatamente o que se pretende, então implementam-se aqui as funcionalidades necessárias. Um exemplo deste caso seria a implementação de um diálogo para escolha de tipos de letra que não existia à data de criação desta camada na biblioteca padrão de Java.

A camada seguinte, SMS, é a chamada camada de aplicação. Nesta camada estão as implementações das funcionalidades diretamente relacionadas com a aplicação Modelsphere, nomeadamente o meta-modelo utilizado pela aplicação. Esta camada divide-se em três partes: parte comportamental, parte relacional e parte orientada a objetos. Cada uma destas partes agrupa os tipos de diagramas correspondentes.

A camada superior, chamada camada de plug-ins, como o próprio nome indica, apresenta plug-ins para a aplicação, isto é, módulos para extensão de funcionalidades, como por exemplo,

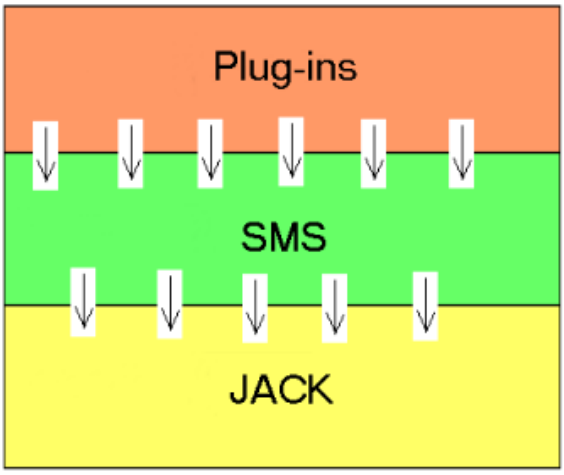


Figura 2.3: Arquitetura do ambiente de modelação Open Modelshpere [Gra08]

plug-ins de validação de modelos já existentes, plug-ins de geração para geração de código a partir dos modelos ou plug-ins para geração de modelos a partir de código.

Como se pôde perceber pela breve descrição da arquitetura, para a criação do ambiente que se pretende a implementação ocorreria essencialmente na camada do meio, denominada *Shared Modeling Software*. O que isto significa é que se torna necessário modificar o núcleo da aplicação para permitir a extensão a um novo tipo de notação, como é o caso do que se pretende implementar.

facilidade de extensão	2
integração com outros ambientes	Possível
definição de propriedades para elementos	3
definição de regras	3 (OCL)
gravar modelos em XML	Sim
cross-platform	Sim
projeto activo	Ativo

Tabela 2.4: Sumário Open Modelsphere

É uma aplicação bem estruturada e consistente com considerável divisão entre os pacotes de implementação, no entanto, a necessidade de modificar o núcleo da aplicação é um ponto menos positivo quando se pensa na extensão de funcionalidades.

ArgoUML

ArgoUML é um ambiente de modelação UML desenvolvido em Java por Jason Elliot Robbins. A arquitetura inicial desta ferramenta era simples como descrito em [RR00] e representado pela figura 2.4.

Para além desta arquitetura, é possível estender esta aplicação através da criação de módulos. É disponibilizada uma interface bem definida para a implementação dos módulos como apresentado na figura 2.5. Para além das interfaces apresentadas na figura, existem também facilidades para

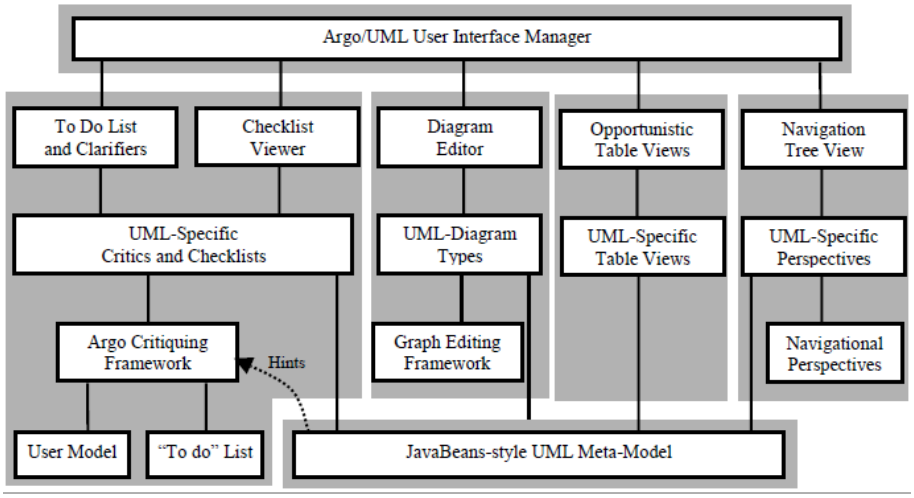


Figura 2.4: Arquitetura do ambiente de modelação ArgoUML [RR00]

a criação de menus e barras de ferramentas o que possibilita uma extensão bastante consistente e conseguida.

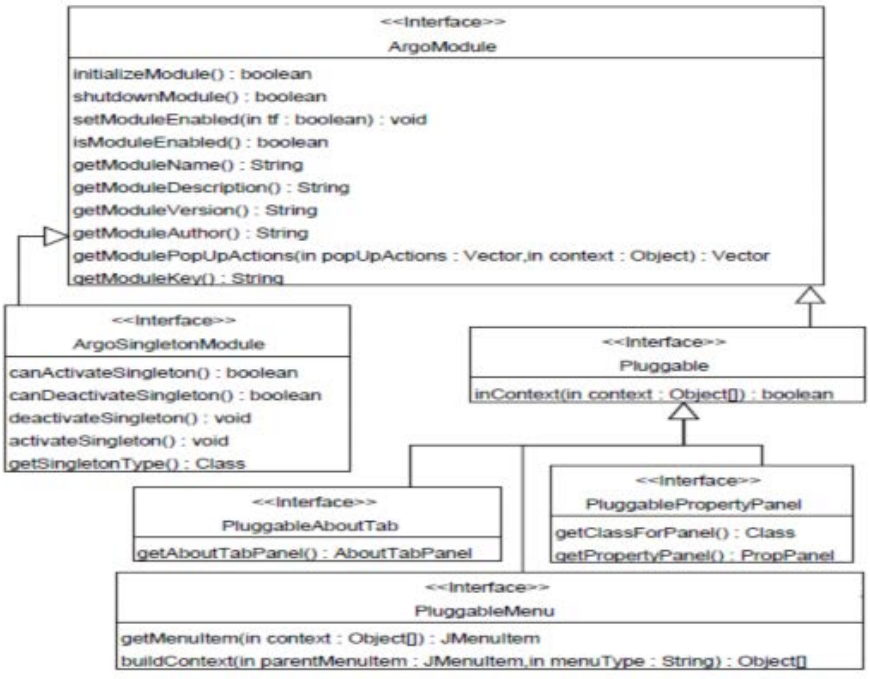


Figura 2.5: Interface de extensão do ArgoUML [LSTM04]

A criação de módulos para extensão da ferramenta, com a existência de interfaces preparadas para essa extensão apresentam uma forma interessante para a condução do trabalho a implementar. Por outro lado, o facto de se apresentar na versão 0.34 indica que possivelmente não será um software tão estabilizado quanto necessário para as pretensões do projeto.

facilidade de extensão	3
integração com outros ambientes	Possível
definição de propriedades para elementos	3
definição de regras	3 (OCL)
gravar modelos em XML	Sim
cross-platform	Sim
projeto activo	Ativo

Tabela 2.5: Sumário ArgoUML

2.3.3 Sumário

Após a análise individual de cada ambiente, a tabela 2.6 apresenta o resumo das classificações para cada ambiente nos diferentes critérios.

Critério	Eclipse Graphical Modeling Framework	StarUML	Open Modelsphere	ArgoUML
facilidade de criação/extensão	4	2	2	3
integração com outros ambientes	Possível	Possível	Possível	Possível
definição de propriedades para elementos	4	3	3	3
definição de regras	3	3	3	3
gravar modelos em XML	Sim	Sim	Sim	Sim
cross-platform	Sim	Windows	Sim	Sim
projecto activo	Sim	Não	Sim	Sim

Tabela 2.6: Resumo comparativo dos ambientes estudados

Fazendo uma breve análise da tabela é possível perceber que o ambiente de modelação StarUML falha nos dois últimos critérios uma vez que não é "cross-platform"(só trabalha em windows) e é um projeto que já não está ativo. Como tal, este ambiente fica desde logo fora das possibilidades.

Quanto às restantes ferramentas, o que as separa é essencialmente a facilidade em criar as funcionalidades pretendidas.

A necessidade de alterar o núcleo da aplicação no caso do Open Modelsphere apresenta-se como um caso mais complicado comparado com as outras ferramentas e, portanto, a dificuldade

de extensão é superior. Quanto ao ArgoUML, a existência de interfaces bem definidas para possibilitar a extensão é um ponto muito positivo, mas ainda assim é necessário conhecer a arquitetura interna da aplicação. Assim, a ferramenta que apresenta melhores perspetivas de desenvolvimento porque facilita a criação de um ambiente dedicado à linguagem em causa e permitirá mais facilmente implementar os requisitos existentes acaba por ser o Eclipse Graphical Modeling Framework.

2.4 Resumo

O estudo sobre linguagens específicas do domínio (DSL) permitiu perceber o interesse deste tipo de linguagens e o seu contexto no projeto, como uma forma de facilitar a criação dos modelos e diminuir o esforço de criação dos mesmos, diminuindo consideravelmente o domínio da notação utilizada, tornando mais simples a sua percepção.

O estudo de teste baseado em modelos, sobretudo na vertente de teste de interfaces gráficas leva a compreender as boas perspetivas de sucesso desta abordagem. Para além disso, a forma como são gerados os casos de teste depende do modelo usado na abordagem MBT. Os modelos serem encarados como máquinas de estados leva a uma técnica de geração que acaba por ser uma travessia sobre o modelo. Por outro lado, este estudo permitiu também perceber que os trabalhos nesta área não estão isentos de problemas e é preciso algum cuidado ao utilizar esta abordagem.

Relembrando, o trabalho de Memon [MBN03],[BM07] apresenta problemas por depender de perfis de utilização para a criação dos cálculos estatísticos, uma vez que quando se trata do desenvolvimento de raiz de uma GUI é complicado obter esses perfis. Para além disso, um problema ainda maior tem que ver com o tipo de erros que esta abordagem consegue detetar, já que, apenas consegue detetar erros fatais que representam a avaria do sistema.

Quanto ao trabalho de Ana Paiva [PFTV05], apesar de permitir detetar mais erros, sofre do problema de exigir muito esforço na construção dos modelos que são depois utilizados para a geração dos casos de teste.

Com as falhas apresentadas, surge a possibilidade de procurar uma solução que ataque estes problemas, nomeadamente a procura de uma abordagem que permita encontrar diversos tipos de erros, mas que ao mesmo tempo não represente um esforço tão grande na construção de modelos.

Quanto à análise das ferramentas, e de acordo com o sumário da secção anterior, a ferramenta escolhida para a fase de implementação é o Eclipse Graphical Modeling Framework. A escolha desta ferramenta deve-se à facilidade de criação e costumização de todos os elementos necessários à DSL. O facto de estar integrado num ambiente de desenvolvimento altamente utilizado, como é o caso do Eclipse, é também uma mais-valia interessante.

Capítulo 3

PARADIGM - Modeling Environment

Após o estudo de informação importante para a perceção do que está em causa neste trabalho, bem como do que será necessário para a criação de um ambiente de modelação, importa agora passar à fase de detalhe do que é pretendido e à sua conceção.

Este capítulo começará com uma descrição da linguagem PARADIGM para a qual o ambiente de modelação será criado. De seguida serão apresentados os requisitos, os cenários de utilização e casos de uso e uma pequena reflexão sobre a geração de caminhos de teste.

Depois será descrita a fase de implementação, apresentando os modelos de criação e o aspeto e funcionalidades do ambiente criado.

No final, breves considerações acerca do que se apresentou neste capítulo.

3.1 Linguagem

O meta modelo da linguagem pode ser visto na figura 3.1. Esta linguagem, construída numa fase anterior do projeto, tem nós e relações entre nós.

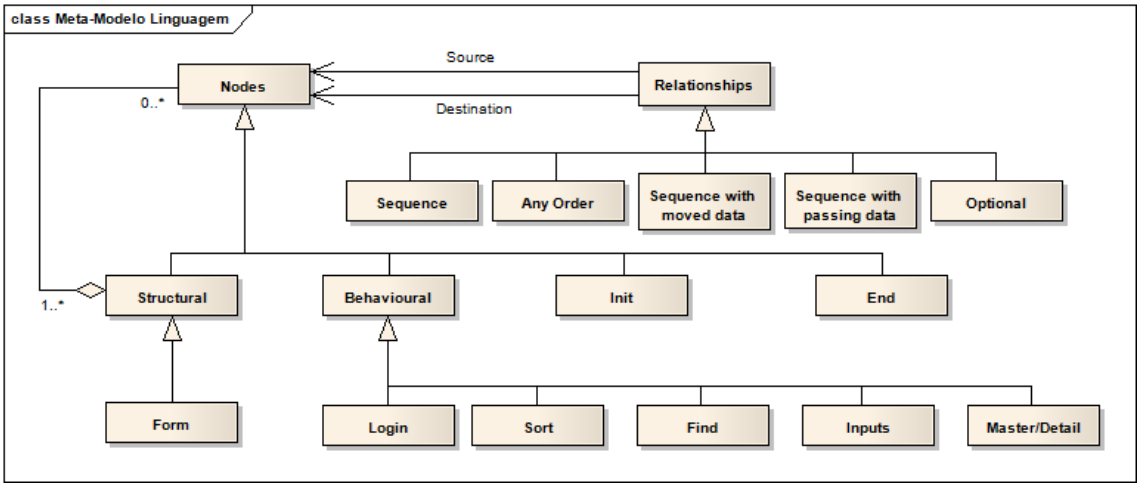


Figura 3.1: Meta Modelo da Linguagem

Quanto aos nós, estes podem ser *"Structural"*, *"Behavioural"*, *"Start"* ou *"End"*. Os nós *"Start"* e *"End"* servem para demarcar o início e fim do diagrama, respetivamente. Quanto aos nós *"Structural"*, estes permitem a criação de nós no seu interior, o que isto significa é que podemos criar uma hierarquia em que temos elementos contidos noutros elementos. Finalmente, os nós do tipo *"Behavioural"* compreendem, como o próprio nome indica, nós que exibem algum tipo de comportamento, e que estão ligados aos padrões de comportamente já referidos anteriormente. Os nomes deverão ser auto-explicativos do que cada um dos elementos representa.

No que se refere às relações, existem 5 tipos de relações *"Sequence"*, *"Sequence with passing data"*, *"Sequence with moved data"*, *"Optional"* e *"Any Order"*. *"Sequence"* refere uma ordem nos acontecimentos. Na relação *"Sequence with passing data"* existe passagem de informação do elemento de origem para o elemento de destino. A relação *"Sequence with moved data"* modela uma situação em que o elemento de origem passa informação para o elemento de destino perdendo, o primeiro essa mesma informação. Quanto a *"Optional"*, significa que o elemento de destino é opcional, e pode não ser usado. Finalmente, a relação *"Any Order"* indica que os elementos de destino podem ser executados em qualquer ordem.

Para além dos elementos descritos, a linguagem apresenta algumas regras que não expostas no meta modelo. Estas regras são:

- Só pode existir um nó do tipo *Start* e um nó do tipo *End* por modelo;
- Nós do tipo *Start* não podem ser destino de uma relação;
- Nós do tipo *End* não podem ser origem de uma relação;
- É obrigatório que os nomes e números identificativos dos nós sejam todos diferentes;
- Não podem existir nós sem nome ou número identificativo;
- Não podem existir modelos sem título;
- Para geração de caminhos de teste é necessário que existam configurações de dados de teste em todos os nós existentes;
- Quando existe passagem de informação entre nós, as configurações presentes na relação de passagem de informação devem ser coerentes com as configurações existentes nos nós.

Esta linguagem poderá, no futuro, sofrer alterações/extensões, em especial no que concerne aos nós apresentados, permitindo o aparecimento de nós representativos de outros tipos de padrões.

3.2 Requisitos

Em [Som06] requisito é definido como "a descrição dos serviços fornecidos por um sistema".

Nesta secção são descritas as funcionalidades que se espera que a aplicação (ambiente de modelação) apresente. De notar que, neste caso, deve entender-se requisitos como sendo os requisitos funcionais. São enumerados, de seguida, os requisitos.

1. Deve ser possível criar modelos.
2. Deve ser possível criar os elementos da linguagem no modelo (nós e relações).
3. Deve ser possível configurar dados de teste para cada nó existente.
4. Deve ser possível gerar os caminhos de teste a partir do modelo.
5. Deve ser possível detalhar um formulário num novo diagrama.
6. Deve ser possível ser alertado para erros de integridade no modelo.
7. Deve ser possível guardar o modelo em formato XML.

3.3 Cenários de utilização

Nesta secção apresenta-se uma brevíssima descrição de um cenário de utilização do sistema, seguido do detalhe dos diferentes casos de uso encontrados.

O utilizador começa pela criação de um projeto.

Quando é apresentado o diagrama, começa-se então a modelação com a colocação dos diferentes nós e relações a ligá-los. De lembrar que entre os nós adicionados terão que estar um nó do tipo Start e um do tipo End.

Faz-se a configuração dos diferentes nós, acrescentando todos os dados de teste que se achem pertinentes.

Finalmente, procede-se à geração dos caminhos de teste. Para a execução desta geração é necessário que não existam erros de integridade.

Trata-se de um cenário de utilização muito simples, mas que permite compreender o que é esperado que o ambiente faça.

Após uma percepção global do sistema que se pretende através dos requisitos e do cenário acima descrito, passa-se a um grau de detalhe ainda maior através dos casos de uso. Na figura 3.2 podem ver-se os casos de uso existentes.

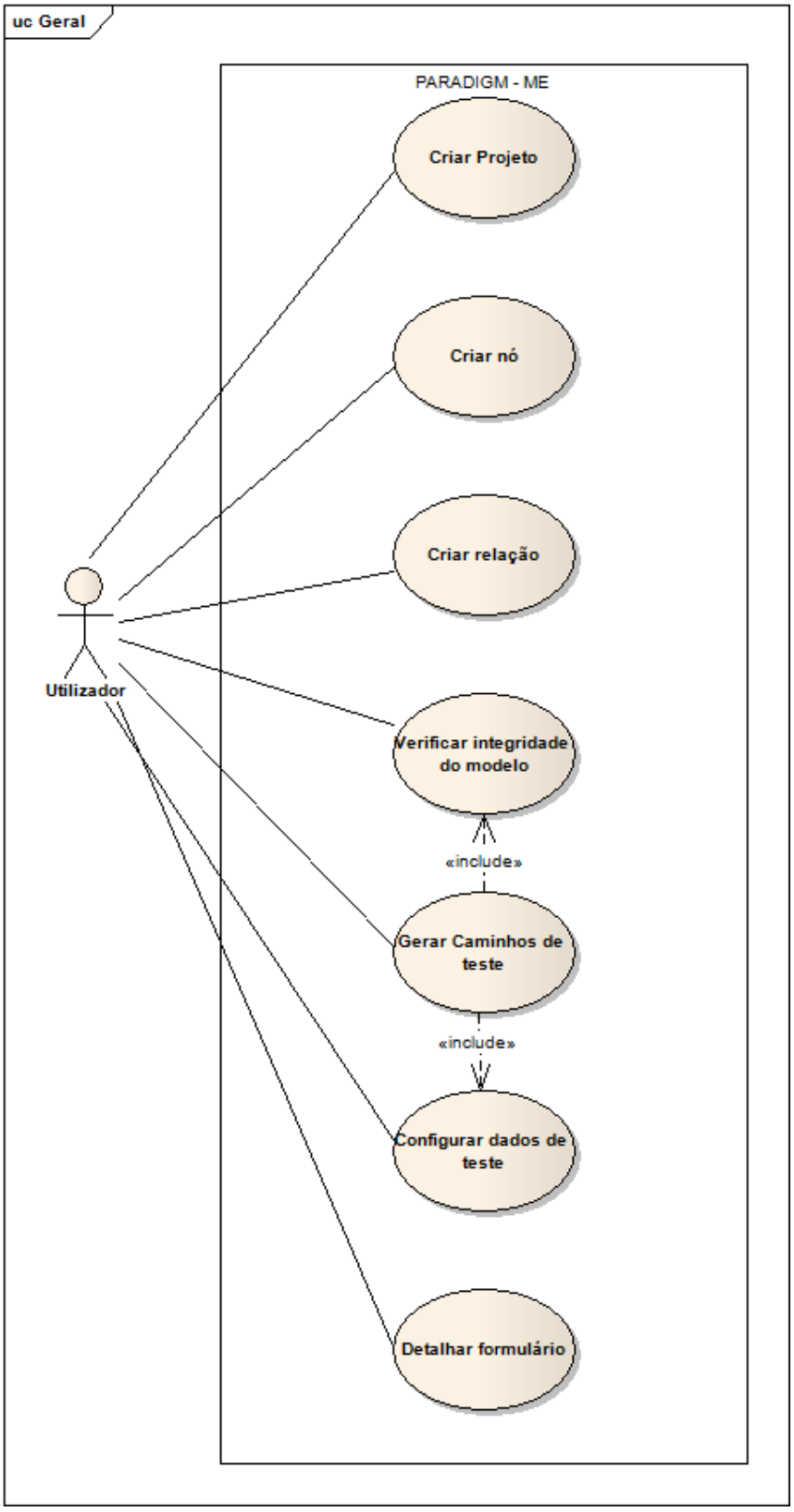


Figura 3.2: Casos de uso

De seguida são detalhados, um a um, os casos de uso existentes, sendo apresentada uma descrição e um fluxo de eventos para cada um.

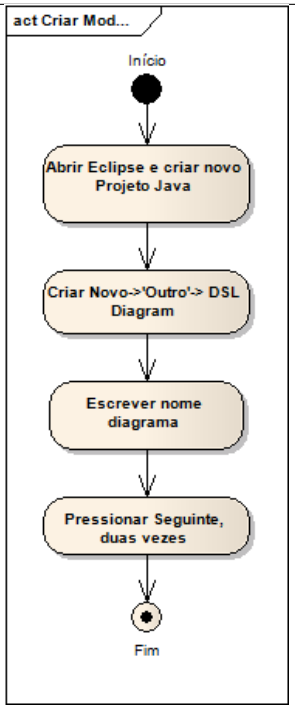
ID	UC01
Nome	Criar projeto
Descrição	Criação do projeto
Fluxo de Eventos	
Pré-condições	-
Pós-condições	Projeto foi criado e o diagrama é apresentado pela aplicação.

Tabela 3.1: Caso de uso "Criar projeto"

Este caso de uso apresenta a forma de criação de um projeto de modelação de interface gráfica. Inicialmente, trata a criação de um projeto como qualquer outro, dentro do ambiente Eclipse, no entanto, para a criação dos ficheiros de modelação existem algumas diferenças. É ainda importante explicar a necessidade de pressionar o botão seguinte duas vezes pelo facto de que só assim é possível ter o ficheiro do diagrama e o ficheiro com a descrição do modelo com o mesmo nome. Ainda de referir que este caso de uso é o ponto de entrada, ou seja, é o primeiro passo, para a utilização do ambiente de modelação.

O caso de estudo "Criar nó"apresenta uma das funcionalidades básicas do ambiente de modelação. Como o próprio nome indica este permite a criação de um dos 8 diferentes tipos de nós presentes na linguagem. Esta é uma das ações que o utilizador mais executará durante a utilização do ambiente.

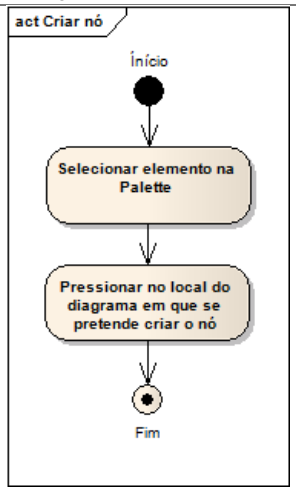
ID	UC02
Nome	Criar nó
Descrição	Criação de um nó
Fluxo de Eventos	
Pré-condições	Diagrama do modelo aberto.
Pós-condições	Existe um novo nó do tipo selecionado no diagrama.

Tabela 3.2: Caso de uso "Criar nó"

Tal como no caso de uso "Criar nó", o caso de uso "Criar relação"apresenta uma das funcionalidades básicas e mais utilizadas do ambiente. A criação de relações pressupõe sempre a existência de dois nós, um de origem e um de destino.

Tanto no caso de uso anterior como neste existem outras formas para além da apresentada no fluxo de eventos de criar os nós ou relações, que fazem uso de outras ferramentas disponibilizadas pelo ambiente.

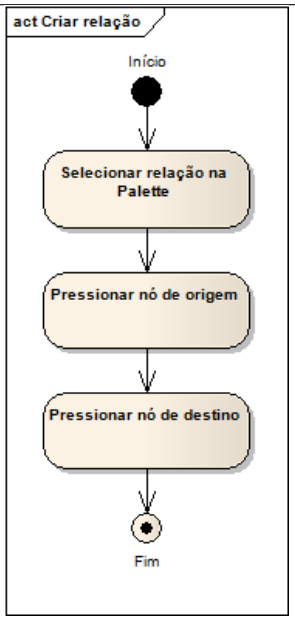
ID	UC03
Nome	Criar relação
Descrição	Criação de uma relação
Fluxo de Eventos	
Pré-condições	Diagrama do modelo aberto. Existe um nó de origem.
Pós-condições	Existe uma nova relação do tipo selecionado no diagrama, a ligar o nó de origem e o nó de destinos escolhidos.

Tabela 3.3: Caso de uso "Criar relação"

O caso de uso seguinte, "Verificar integridade do modelo", permite ao utilizador perceber se o modelo que está a construir viola alguma das regras de integridade da linguagem. Este caso de uso é extremamente importante, no sentido em que, é através desta ação que é possível que o utilizador perceba se existe algum erro no modelo que impeça a geração de casos de teste. De notar que esta verificação é também feita de forma automática imediatamente antes da geração de caminhos de teste.

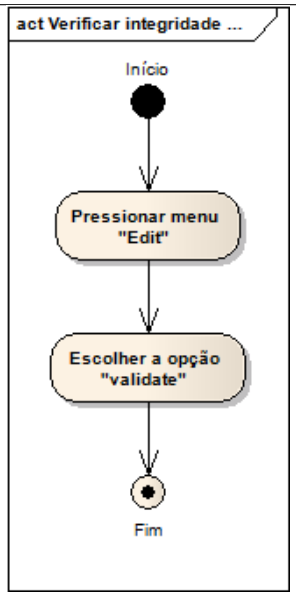
ID	UC04
Nome	Verificar integridade do modelo
Descrição	Verificação da integridade do modelo
Fluxo de Eventos	
Pré-condições	Diagrama do modelo aberto.
Pós-condições	Caso existam problemas, estes são indicados no separador respetivo.

Tabela 3.4: Caso de uso "Verificar integridade do modelo"

O caso de uso "Configurar dados de teste"apresenta a ação de introduzir os dados que o utilizador pretenda que sejam utilizados aquando da exercitação da GUI. Estes dados deverão apresentar congruência entre si, para além de que devem ter sentido dentro da interface a testar. As configurações estarão extremamente dependentes do tipo de nó de que fazem parte, uma vez que necessariamente, as configurações para um nó do tipo *"Find"* não poderão ser iguais às configurações do nó do tipo *"Sort""*.

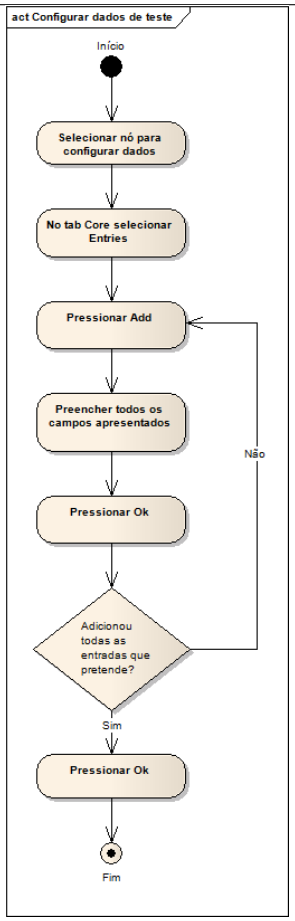
ID	UC05
Nome	Configurar dados de teste
Descrição	Configuração de dados para serem usados na fase de teste
Fluxo de Eventos	
Fluxo de Eventos alternativo	Neste caso de uso o fluxo de eventos pode ser ligeiramente diferente consoante o tipo de nó que estamos a configurar. Nomeadamente no caso do Master/Detail deve adicionar-se cada uma das entradas "master"e para cada uma delas selecioná-la na lista e adicionar as entradas "detail"respetivas.
Pré-condições	Diagrama do modelo aberto. Existe (pelo menos) um nó descendente de Behavioural.
Pós-condições	É possível verificar nas propriedades do nó, a existência das entries criadas.

Tabela 3.5: Caso de uso "Configurar dados de teste"

No caso de uso "Gerar caminhos de teste"apresenta-se a ação de criação dos casos que serão utilizados para testar a GUI. Esta geração é feita após uma verificação automática de que não existe violação de qualquer das regras de integridade da linguagem. Esta ação só deverá ser efetuada após a construção completa do modelo, bem como a configuração de todos os dados de teste a utilizar.

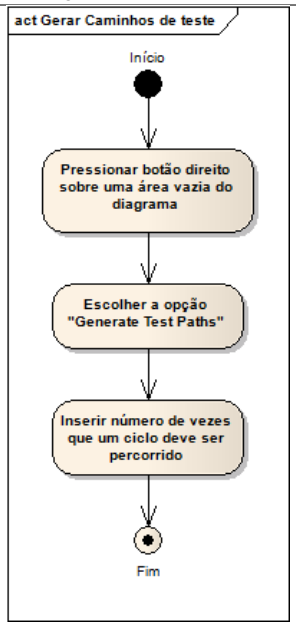
ID	UC06
Nome	Gerar caminhos de teste
Descrição	Geração de todos os caminhos de teste possíveis
Fluxo de Eventos	
Pré-condições	Diagrama do modelo aberto. Existe um nó Start, um nó End e pelo menos um caminho entre esses dois nós. O modelo não viola qualquer regra de integridade da linguagem.
Pós-condições	É criado um ficheiro com todos os caminhos entre o nó Start e o nó End do diagrama.

Tabela 3.6: Caso de uso "Gerar caminhos de teste"

Por último, o caso de teste "Detalhar formulário"apresenta as ações que levam à criação de um sob-modelo em que é possível detalhar os elementos existentes dentro de um nó do tipo formulário. Os passos apresentados no fluxo de eventos podem ser utilizados para criar um sub-modelo para o formulário em causa, mas também para abrir um sub-modelo que já tenha sido criado anteriormente.

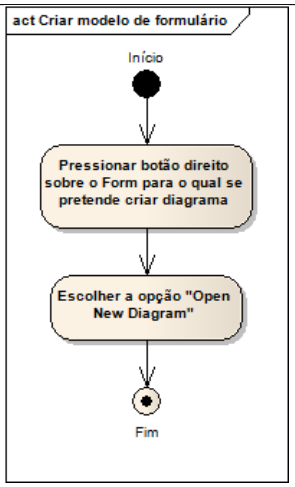
ID	UC07
Nome	Detalhar formulário
Descrição	Criação do modelo de um formulário, utilizando um novo diagrama
Fluxo de Eventos	
Pré-condições	Diagrama do modelo aberto. Existe (pelo menos) um nó do tipo Form.
Pós-condições	É criado um novo diagrama e é apresentado pela aplicação.

Tabela 3.7: Caso de uso Detalhar formulário"

3.4 Geração de caminhos de teste

Após o levantamento de requisitos é necessário destacar a fase de geração de caminhos de teste, uma vez que, esta fase acarreta a explicação de alguns aspetos importantes.

Desde logo, a forma como os modelos serão vistos na fase de geração de caminhos. Os modelos serão utilizados sob o ponto de vista de uma máquina de estados, e como tal a geração de caminhos será realizada através da travessia dessa máquina de estados que é o modelo. No entanto, esta travessia apresenta alguns pontos chave.

Em primeiro lugar para que se possa realizar esta geração é necessário que os modelos não apresentem qualquer erro de integridade.

De seguida é preciso chamar a atenção para o tratamento das relações *Optional* e *Any Order*, que ao contrário das restantes, apresentam comportamentos que não se limitam à mera passagem de um nó para o seguinte.

No caso de uma relação do primeiro tipo, imagine-se um nó A ligado a um B por uma relação *Optional* e o nó B ligado a um nó C por uma outra relação. Na geração de caminhos de teste o que

acontece é a geração de um caminho em que se passa de A para B e deste para C e a geração de outro caminho em que se passa do nó A diretamente para o nó C.

Quanto ao segundo tipo de relação referido, deverão ser criados caminhos para todas as combinações possíveis de nós. Por exemplo, se existir um nó A ligado a um nó B por uma relação *Any Order*, o gerador gera os caminhos:

- A B;
- B A.

Para além destes dois tipos de relações é ainda necessário explicar o que acontece durante a geração de caminhos quando se depara com um nó do tipo *Form*. Nestes casos o gerador deverá verificar a existência de um modelo associado a este nó e gerar todos os caminhos desse modelo. Finda essa operação, o gerador voltará ao modelo inicial e continuará a geração normalmente.

3.5 Modelos de Criação

Após a descrição em detalhe de tudo o que se pretendia do ambiente, chega o momento de criar o ambiente propriamente dito, por forma a responder às necessidades existentes.

Como referido nas conclusões do estudo às ferramentas, constantes do capítulo 2, a ferramenta a utilizar é o Eclipse Graphical Modelling Framework.

Tal como indicado no estudo, esta ferramenta faz uso de quatro modelos para a criação dos ambientes de modelação.

Estes quatro modelos, que serão vistos com maior pormenor nas próximas secções, tratam o domínio da aplicação, isto é, quais os elementos constantes dos diagramas, o aspeto gráfico dos elementos, ou seja como é que são representados no diagrama, e ainda as ferramentas para a criação dos elementos, ou seja, os menus correspondentes. Para fazer a ligação entre estes três modelos existe um quarto modelo, o modelo de mapeamento, que permite mapear um elemento do domínio para um elemento gráfico e um elemento de menus.

3.5.1 Modelo do domínio

No modelo do domínio são então definidos todos os elementos que devem constar dos diagramas, bem como todas as suas propriedades. Na figura 3.3 pode ver-se um diagrama representativo dos elementos do domínio, as suas relações e as suas propriedades.

Se se comparar este diagrama com o diagrama de descrição da linguagem, na figura 3.1, é possível perceber que este serviu de base à criação do modelo de domínio, sendo depois acrescentados elementos que têm que ver com a configuração dos dados de teste de cada nó.

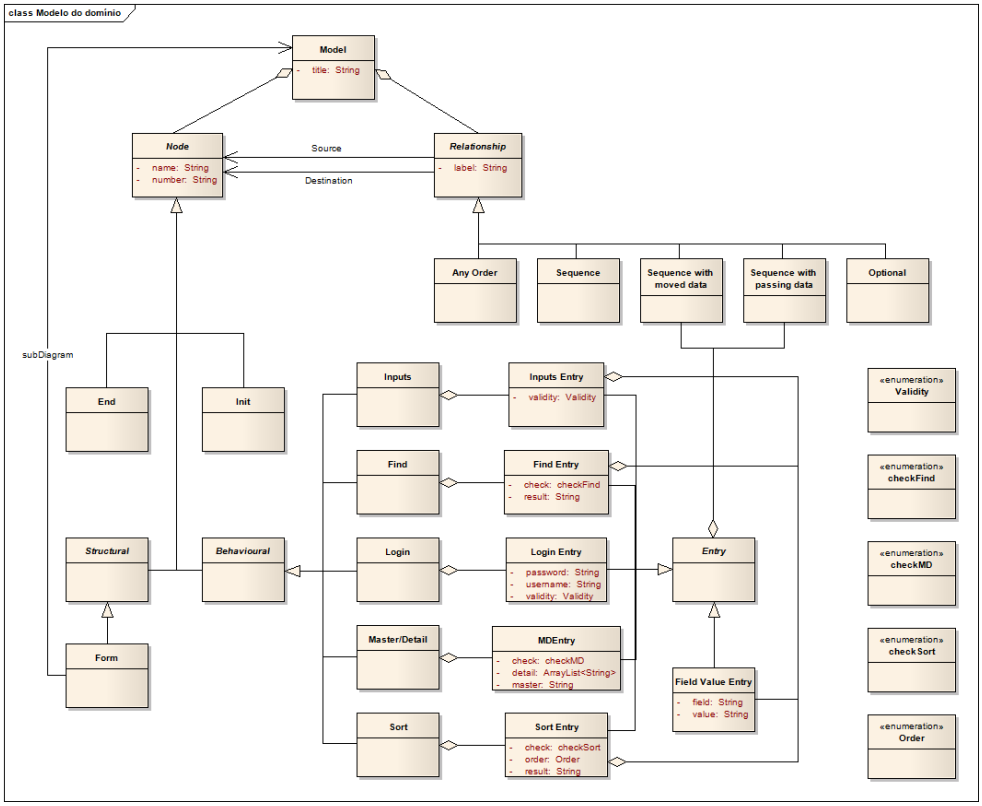


Figura 3.3: Diagrama do domínio

3.5.2 Modelo gráfico

Este modelo trata o aspeto gráfico dos elementos presentes nos diagramas a criar com o ambiente.

Na figura 3.4 pode ver-se que neste modelo são descritos todos os elementos que devem aparecer no diagrama: os nós, as relações e as etiquetas, isto é nomes ou números relativos a cada nó ou relação.

A figura 3.5 permite ver alguns detalhes, nomeadamente que o nó de Login (assim como todos os outros) é apenas desenhado como um retângulo, uma vez que a imagem que depois aparece é carregada a partir de um plug-in criado especialmente para a utilização de imagens como nós do diagrama. É também possível ver que as relações são criadas com uma simples linha e dispõem de uma label onde estará o símbolo representativo da relação.

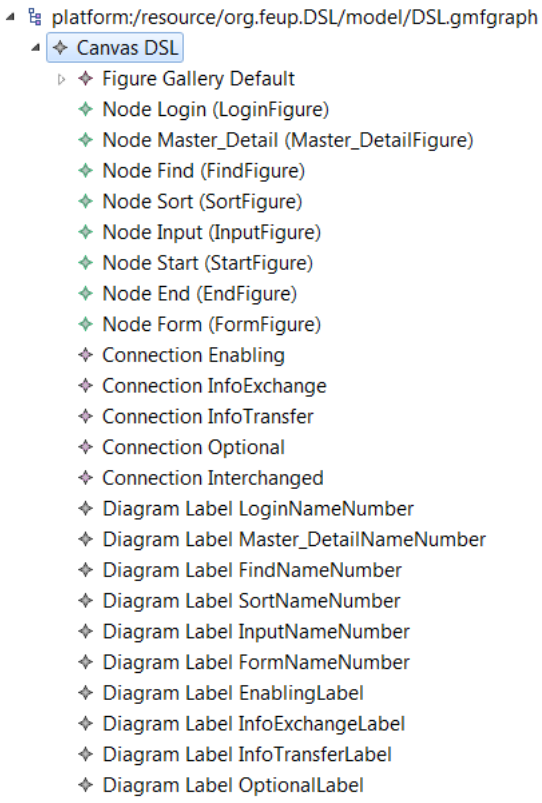


Figura 3.4: Modelo gráfico

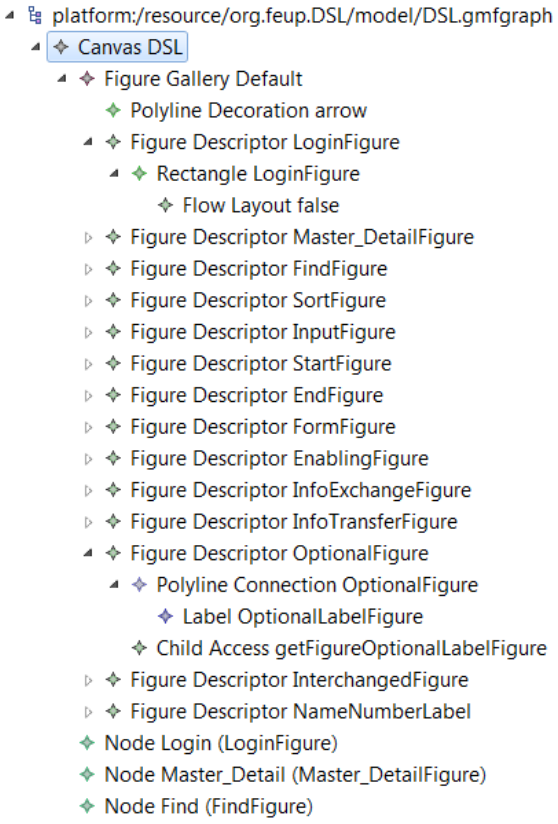


Figura 3.5: Modelo gráfico (detalhe)

3.5.3 Modelo de ferramentas

No caso do modelo de ferramentas este trata dos elementos de menus.

Na figura 3.6 é possível ver a criação da palette de elementos. Para cada nó e para cada relação é criada uma entrada, sendo também definida a imagem a usar para indicar o comando.

O modelo de ferramentas ainda não está completamente desenvolvido pelo Eclipse e como tal apresenta algumas limitações, o que significa que alguns dos menus não podem ser criados a partir deste modelo, tendo que ser criados em código.

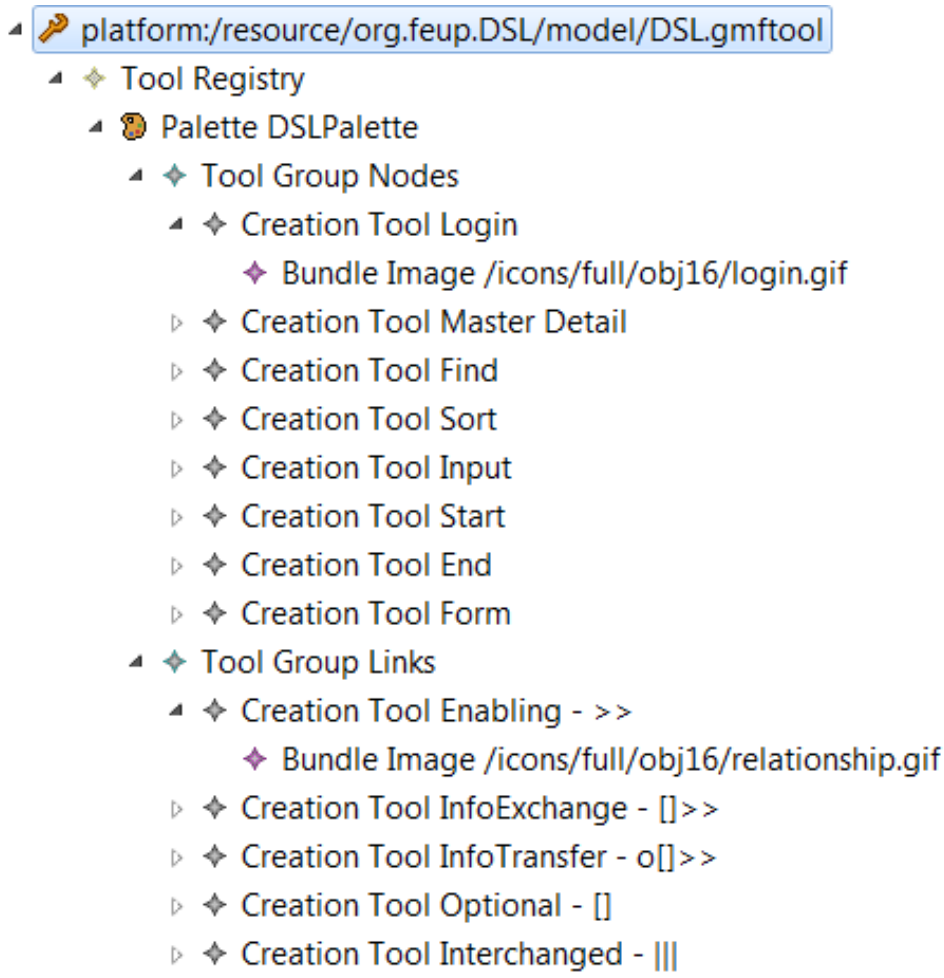


Figura 3.6: Modelo de ferramentas

3.5.4 Modelo de mapeamento

Depois da criação dos três modelos acima resta a criação do modelo de mapeamento, que permite fazer a ligação entre os outros três modelos. O modelo de mapeamento apresenta uma série de opções, desde as mais simples que têm então que ver com especificar qual a ferramenta e o aspeto gráfico que determinado elemento do domínio tem, até à criação de regras de integridade do modelo.

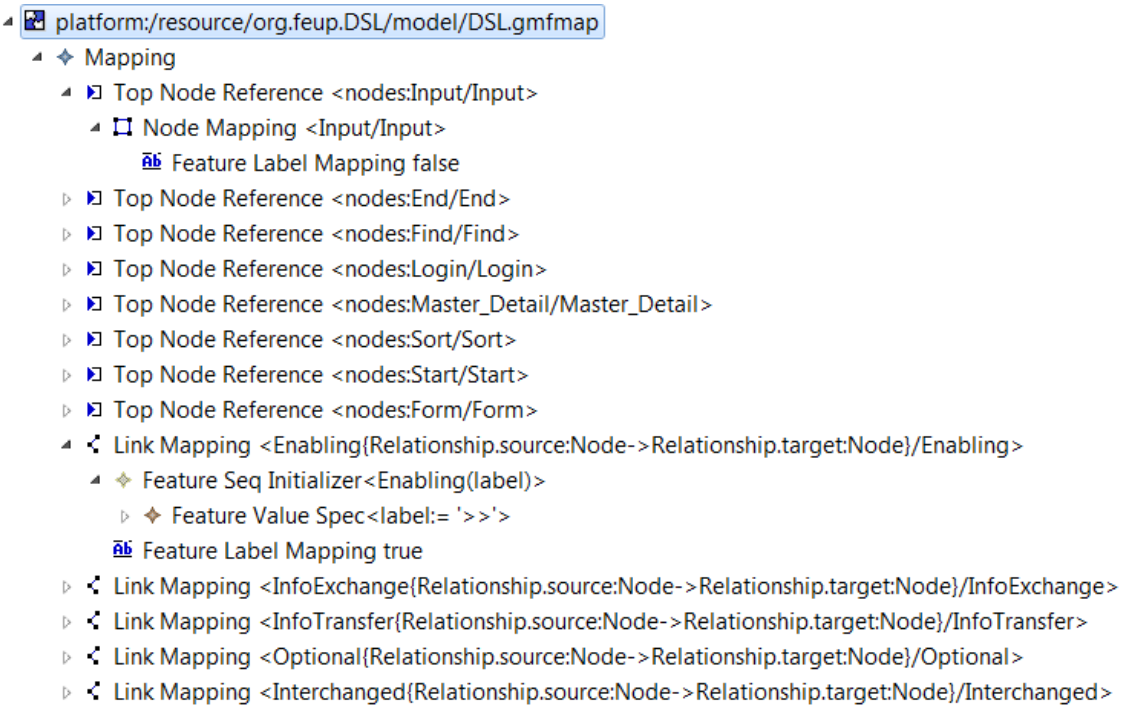


Figura 3.7: Modelo de mapeamento

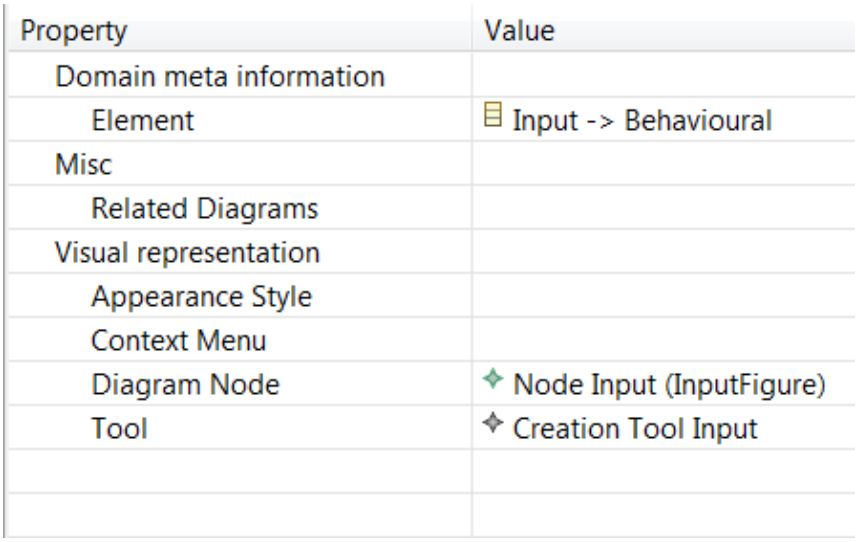


Figura 3.8: Propriedades do mapeamento do nó Input

Na figura 3.7 é possível ver cada um dos nós e relações a serem mapeados e na figura 3.8 é possível ver um exemplo das propriedades de mapeamento do nó Input.

Quanto à parte de criação das regras de integridade, na figura 3.9 pode ver-se a lista de regras implementadas.

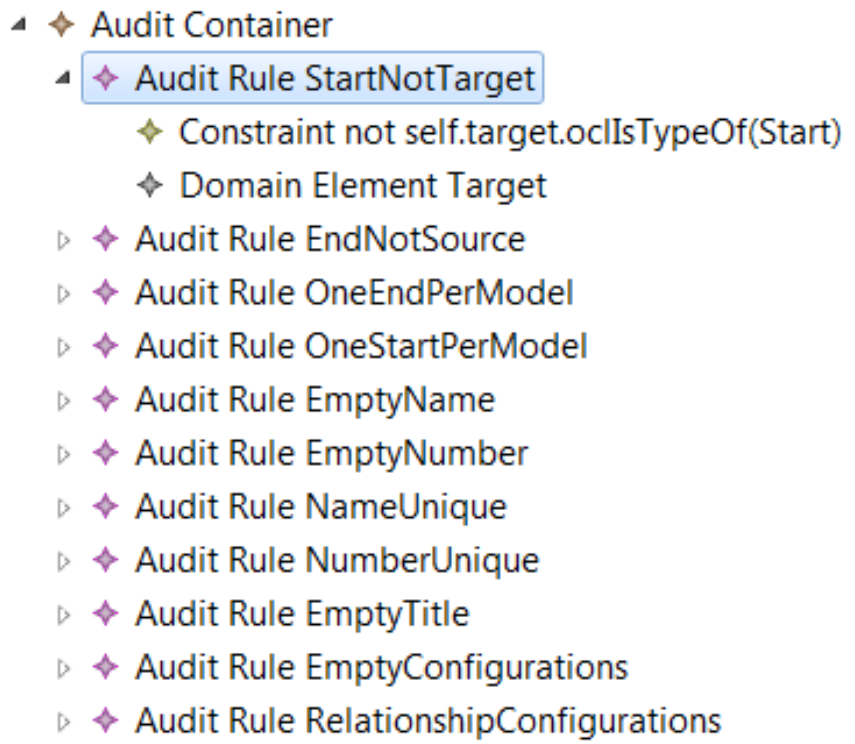


Figura 3.9: Listagem de regras

Algumas regras foram implementadas utilizando OCL, como é o caso da regra "StartNotTarget"onde se postula que o destino de uma relação não pode ser do tipo *Start*. De notar, que esta regra tem como alvo o elemento do domínio *Relação*.

A regra "EndNotSource"é semelhante à anterior com a diferença de que neste caso o que a regra diz é que a origem de uma relação não pode ser um nó do tipo *End*.

A regra "EmptyName"não permite a existência de nós sem nome. Esta regra verifica o tamanho do atributo nome do nó, garantindo que este não é 0.

A regra "EmptyNumber"é semelhante a "EmptyName", mas para o atributo "number".

A regra "EmptyTitle"é semelhante às anteriores, mas para o atributo "title"do modelo.

As regras "NameUnique"e "NumberUnique"garantem que não existem dois nós com o mesmo nome ou dois nós com o mesmo número, respetivamente.

Por outro lado, existem regras implementadas utilizando a linguagem Java, como é o caso da regra "OneEndPerModel"que não permite a criação de mais do que um nó do tipo *End* em cada modelo.

A regra "OneStartPerModel"é semelhante à anterior, mas para nós do tipo *Start*.

A regra "EmptyConfigurations"garante que não existe nenhum nó cujas configurações estejam vazias.

A regra "RelationshipConfigurations"garante a existência de integridade na configuração de dados nas relações em que existe passagem de informação.

3.6 Ambiente

A partir dos quatro modelos apresentados anteriormente é então possível gerar o código do ambiente de modelação. O ambiente de modelação gerado permitirá efetuar as ações mais simples, como criação de nós e relações, no entanto, ações mais complexas como a configuração dos dados de teste de cada nó, ou a geração de caminhos de teste não estão contemplados nesta geração automática de código.

Neste capítulo será exposto o ambiente em si, o seu aspeto e as suas funcionalidades, bem como alguns dos detalhes de implementação das ações mais complexas.

Pressupõe-se que se dispõe do ambiente de modelação instalado no Eclipse.

Criar projeto e modelo

Para criar um projeto de modelação do PARADIGM - ME basta criar um projeto Java no ambiente Eclipse. De seguida, deve criar-se o modelo procurando na listagem de "outros"por "DSL Diagram", dá-se o nome ao modelo e temos o ambiente de modelação pronto a ser utilizado, tal como se pode ver na figura 3.10,

Pode então ver-se do lado esquerdo o projeto e o modelo criados. Do lado direito está a palette com as ferramentas de criação de todos os nós e relações. Ao centro o espaço de modelação que será utilizado para criarmos os modelos, como será visto já de seguida.

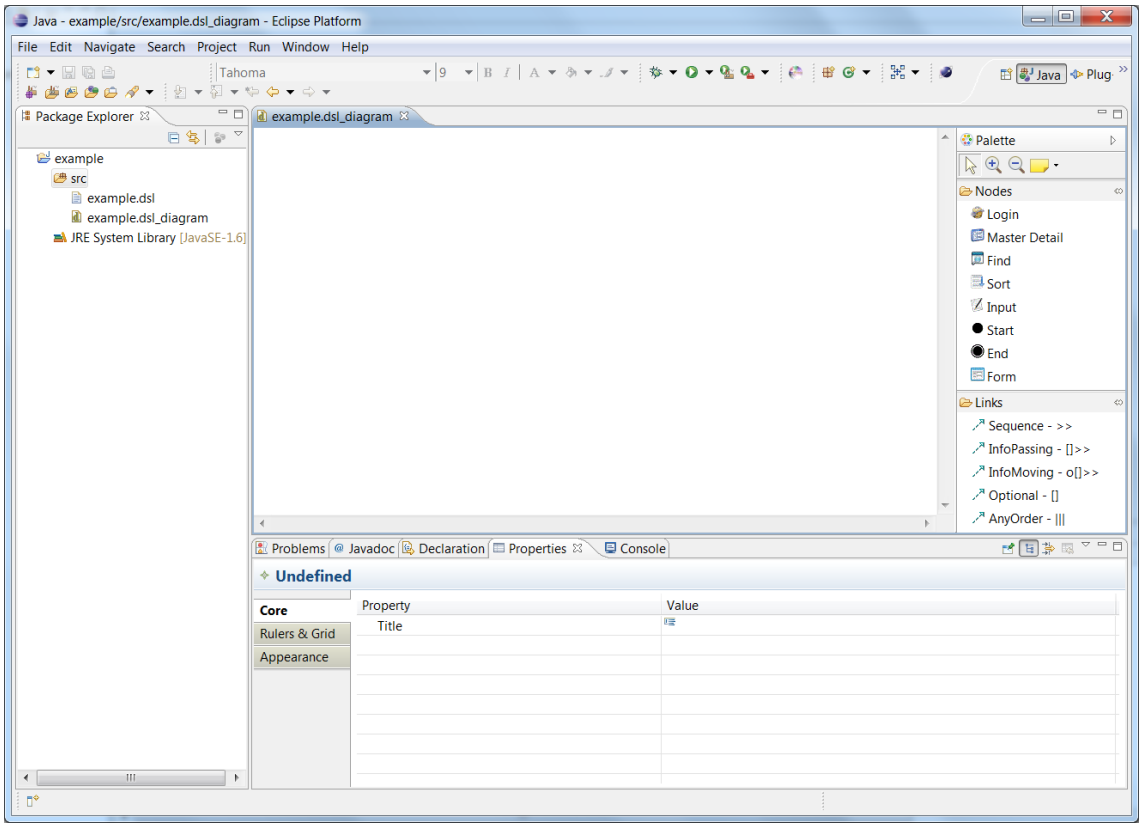


Figura 3.10: Ambiente de modelação

Modelação - criação de nós e relações

Para a criação de um nó basta pressionar na palette o tipo de nó pretendido e pressionar no espaço de modelação onde se pretende colocar o nó. De seguida escreve-se o nome a dar ao nó e entre [] o número identificador que o nó deverá ter. Este número serve para a criação dos caminhos de teste. De lembrar que todos os números e nomes deverão ser diferentes.

A criação de uma relação não é muito diferente. Deve pressionar-se na palette o tipo de relação que se pretende criar pressionar o nó de origem e manter pressionado arrastando até ao nó de destino. No caso da relação não é necessário a introdução de nome ou número.

Configuração de dados de teste

Outro aspeto importante do ambiente passa pela possibilidade de configuração de dados de teste para cada nó representativo de um padrão de comportamento (relembrando: *Inputs*, *Find*, *Login*, *Master/Detail* e *Sort*).

Para demonstrar a configuração de dados de teste será utilizado o exemplo da figura 3.11 e será usado o nó "Redef Pesquisa [2.1]".

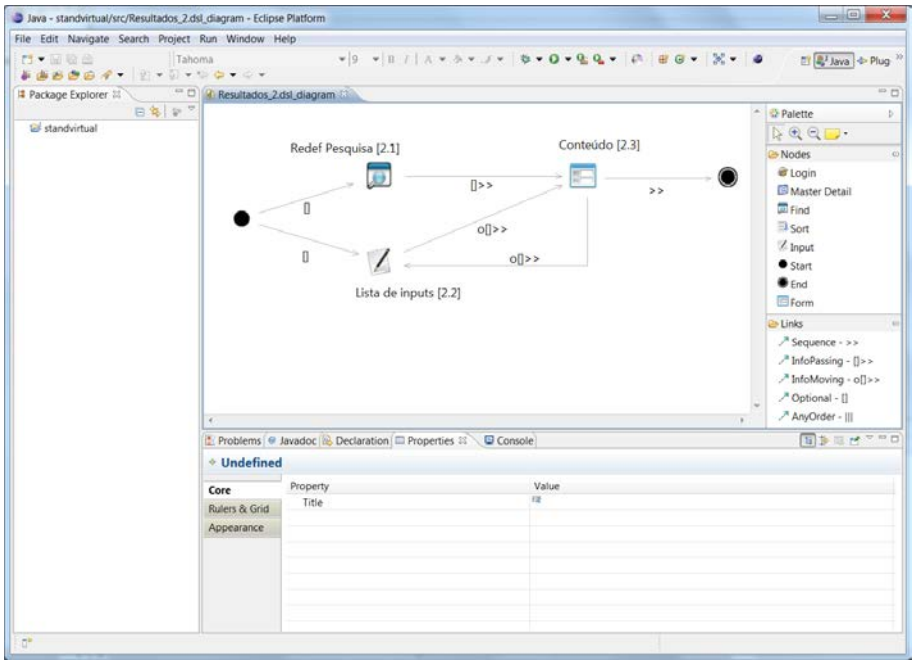


Figura 3.11: Modelo Exemplo

Após selecionar este nó, no separador de propriedades, por baixo da área de modelação, escolhe-se a propriedade *"Find entries"*. Será apresentado um diálogo semelhante ao da figura 3.12. Neste diálogo serão apresentadas as *entries* que já existem (no caso não existe nenhuma).

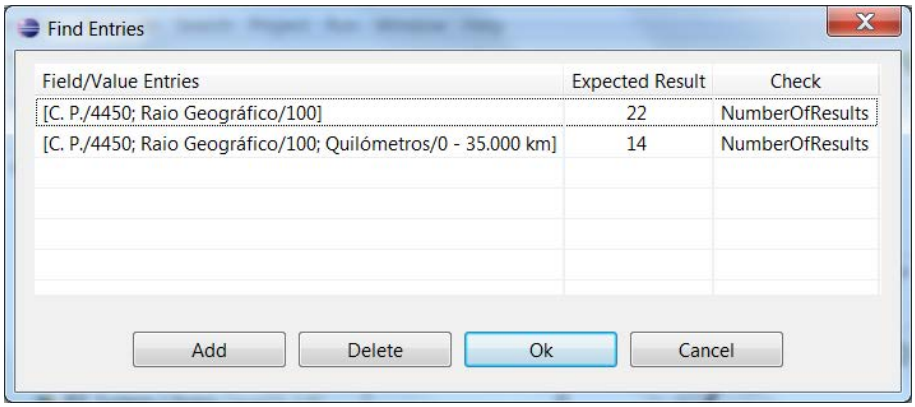


Figura 3.12: Diálogo *"Find entries"*

Para adicionar basta pressionar o botão *Add* e surgirá o diálogo apresentado na figura 3.13. Neste diálogo podemos introduzir o resultado esperado, qual o tipo de check que deve ser feito e os valores de pesquisa e em que campos estes devem ser introduzidos. Para a inserção dos valores a pesquisar e respetivos campos basta pressionar *Add* e será apresentado o diálogo presente na figura 3.14. Neste diálogo bastará escrever o nome do campo e o valor que se pretende introduzir no mesmo.

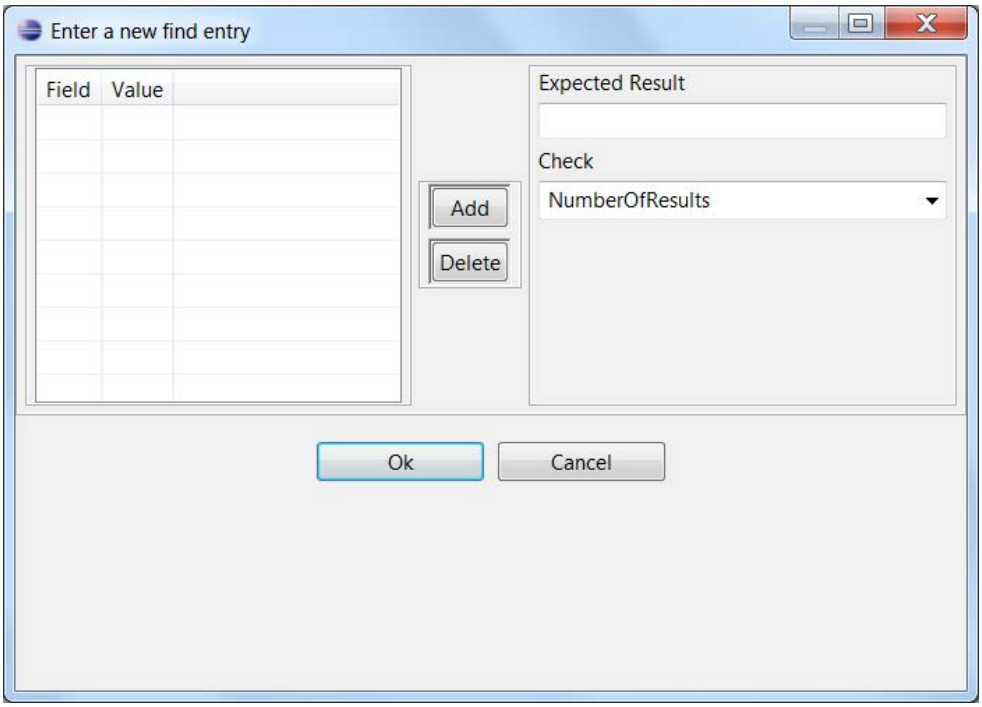


Figura 3.13: Diálogo para adicionar um *"find entry"*

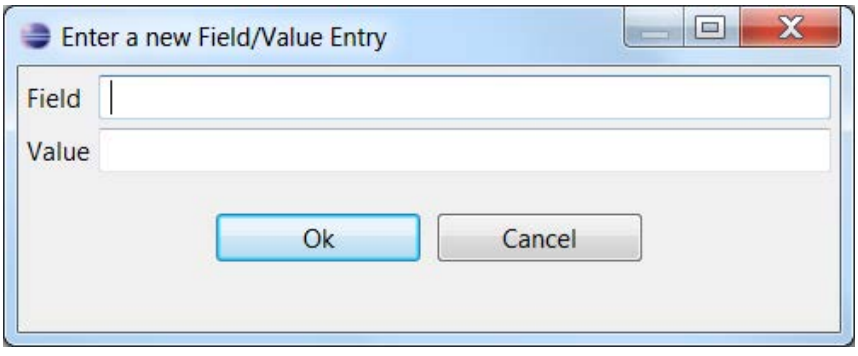


Figura 3.14: Diálogo para adicionar *"Field/Value entries"*

Para possibilitar a criação destas configurações foi necessário criar diálogos adaptados a cada um os dos tipos de nó. Para tal foi necessário, criar um pequeno plug-in que permitisse identificar qual a propriedade que estava a ser chamada e caso fosse a de configuração, direcionar para a criação do diálogo respetivo.

Na figura 3.15 pode ver-se o código que identifica a propriedade e chama o respetivo descriptor de propriedade que seguidamente chama o diálogo apropriado.

```
@Override
protected IPropertyDescriptor createPropertyDescriptor(
        IItemPropertyDescriptor itemPropertyDescriptor) {
    DSLPackage pkg = DSLPackage.eINSTANCE;
    Object feature = itemPropertyDescriptor.getFeature(object);
    if (pkg.getLogin_LoginEntries().equals(feature)) {
        return new LoginEntryPropertyDescriptor(object, itemPropertyDescriptor);
    }
    else if(pkg.getMaster_Detail_MasterDetail().equals(feature)){
        return new Master_DetailPropertyDescriptor(object, itemPropertyDescriptor);
    }
    else if(pkg.getFind_FindEntries().equals(feature)){
        return new FindEntryPropertyDescriptor(object, itemPropertyDescriptor);
    }
    else if(pkg.getSort_SortEntries().equals(feature)){
        return new SortEntryPropertyDescriptor(object, itemPropertyDescriptor);
    }
    else if(pkg.getInput_InputEntries().equals(feature)){
        return new InputEntryPropertyDescriptor(object, itemPropertyDescriptor);
    }
    else if(pkg.getInfoExchange_InfoToExchange().equals(feature)){
        return new InfoToExchangePropertyDescriptor(object, itemPropertyDescriptor);
    }
    else if(pkg.getInfoTransfer_InfoToTransfer().equals(feature)){
        return new InfoToTransferPropertyDescriptor(object, itemPropertyDescriptor);
    }
    else {
        return super.createPropertyDescriptor(itemPropertyDescriptor);
    }
}
```

Figura 3.15: Código de identificação da propriedade a ser chamada

Os diálogos foram construídos com recurso ao SWT (Standard Widget Toolkit).

De notar ainda que vai sendo manipulada uma lista de *Find Entries* enquanto o primeiro diálogo está aberto. Quando finalmente este diálogo é fechado, só então todas as alterações efetuadas são guardadas.

Geração de caminhos de teste

Outro objetivo passava pela geração dos caminhos de teste. Para gerar os caminhos de teste de um modelo bastará pressionar o botão direito do rato e no menu de contexto escolher a opção *"Generate Test Paths"*, como se pode ver na figura 3.16.

O ambiente perguntará qual o número de vezes que se deve passar num ciclo, caso existam ciclos no modelo.

A geração faz-se através de um algoritmo recursivo começando no nó do tipo *Start*, que é obrigatório estar presente em cada modelo para que seja possível fazer a geração, e navega a partir

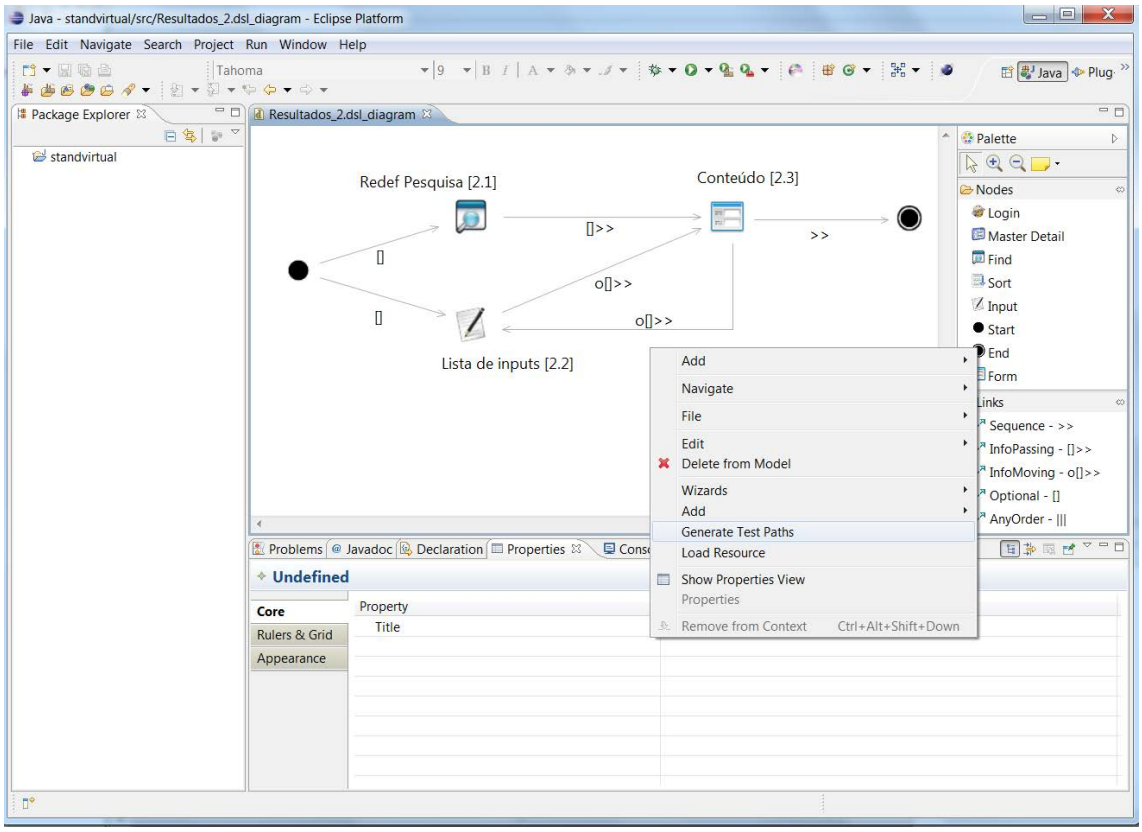


Figura 3.16: Menu para geração de caminhos de teste

daí obtendo os vizinhos do nó respetivo até que chega ao nó do tipo *End*. Aí chegado faz rollback passando para um caminho diferente.

Tal como requerido (e descrito em 3.4) a geração tem em conta as relações do tipo *Optional* e do tipo *Any Order* e nós do tipo *Form*. De ressalvar que no caso destes nós o gerador verifica a existência de um modelo associado a este nó e gera os caminhos para esse modelo guardando esse conjunto de caminhos gerados. No final da geração de caminhos do modelo de primeiro nível o gerador trata então de "injetar"os caminhos gerados para os modelos de cada nó do tipo *Form* no local respetivo.

Quanto à questão dos ciclos utiliza-se uma estratégia semelhante à utilizada para os nós do tipo *Form*, isto é, quando um ciclo é identificado, todos os nós que pertencem ao ciclo são guardados numa nova lista. No final da geração, cada uma destas listas de ciclos é injetada nos caminhos finais.

3.7 Resumo e Conclusões

Este capítulo permite perceber o levantamento de requisitos, os casos de uso existentes e de que forma o ambiente foi implementado.

Em primeiro lugar permitiu conhecer a linguagem e perceber a que diz respeito cada um dos elementos.

Através dos requisitos entendeu-se as funcionalidades que devem estar presentes e os casos de uso trouxeram um fluxo detalhado do que se espera que deva acontecer a cada momento de interação com o sistema.

Quanto à implementação do ambiente de modelação é importante relembrar a criação dos 4 modelos (domínio, gráfico, ferramentas e mapeamento) que a framework utiliza para a geração de um ambiente básico. Após esta fase, passou-se a uma fase de refinamento para atingir um ambiente mais completo e com as funcionalidades pretendidas.

Capítulo 4

Casos de estudo

Neste capítulo são apresentados alguns casos de estudo utilizados durante a realização de testes ao ambiente criado.

Com estes casos de estudo foi possível perceber a aplicabilidade da ferramenta a casos reais, permitindo também perceber se o ambiente é capaz de atingir todos os objetivos que tinham sido propostos inicialmente.

As aplicações selecionadas foram Standvirtual e ERA Portugal. De notar que a escolha destas aplicações não foi inocente, uma vez que se procuraram aplicações que fossem capazes de ilustrar os padrões que já estão disponíveis na linguagem.

Nas secções seguintes, são apresentados os modelos criados, algumas configurações de nós e os caminhos de teste gerados.

4.1 Standvirtual

O Standvirtual (http://www.standvirtual.com) é um sítio Web de venda e compra de veículos, cuja página inicial pode ser vista na figura 4.1.

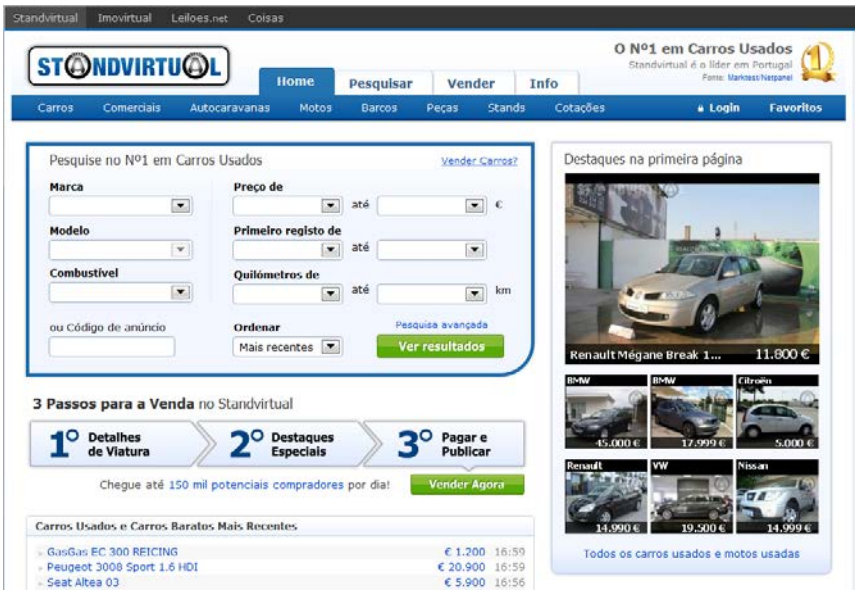


Figura 4.1: Página inicial do sítio standvirtual.com

Devido às características deste sítio, com a possibilidade de pesquisar veículos e ordenar e filtrar os resultados obtidos, apresenta-se como um caso interessante para utilizar no teste do ambiente de modelação.

Na figura 4.2, pode ver-se o modelo da aplicação. Como se pode perceber, este modelo apresenta um nó do tipo *"Find"* representando a pesquisa inicial dos elementos, cuja interface pode ser vista na figura 4.1, e um formulário que apresenta os resultados da pesquisa efetuada, estando o aspeto deste formulário na interface presente na figura 4.3.

De reparar que a ligação entre a pesquisa e o formulário de resultados é feita através de uma ligação do tipo "Sequence with passing data". É utilizada esta ligação dado que as informações que são utilizadas na pesquisa são passadas ao formulário de resultados.

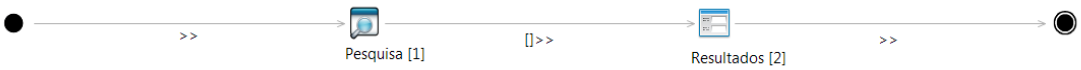


Figura 4.2: Modelo da aplicação standvirtual

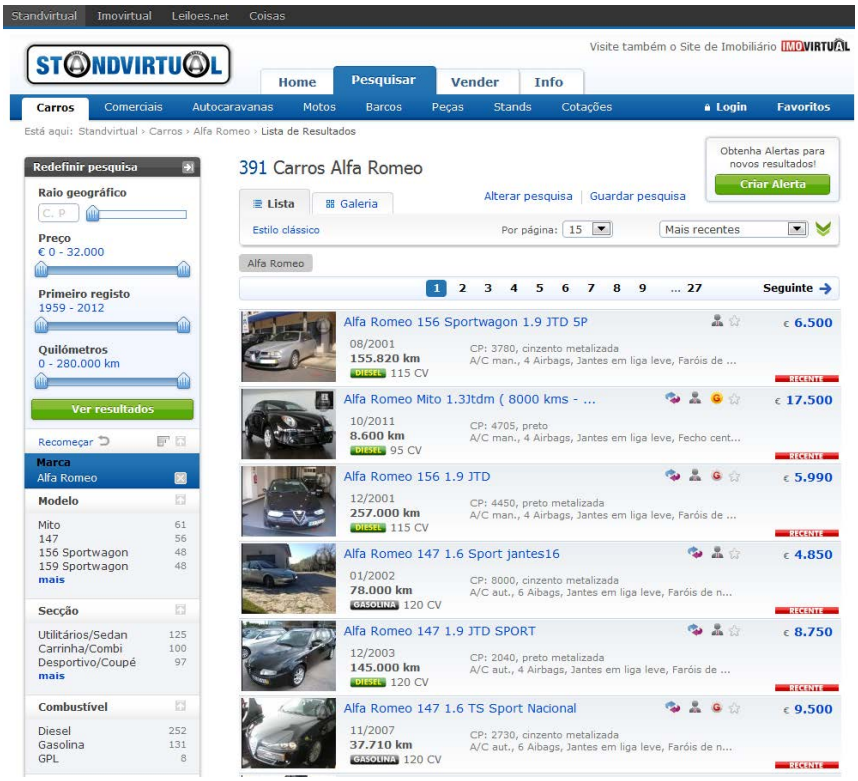


Figura 4.3: Ecrã de apresentação de resultados da pesquisa da aplicação standvirtual

Segue-se então a fase de detalhe do formulário de Resultados. O modelo deste formulário pode ser visto na figura 4.4.

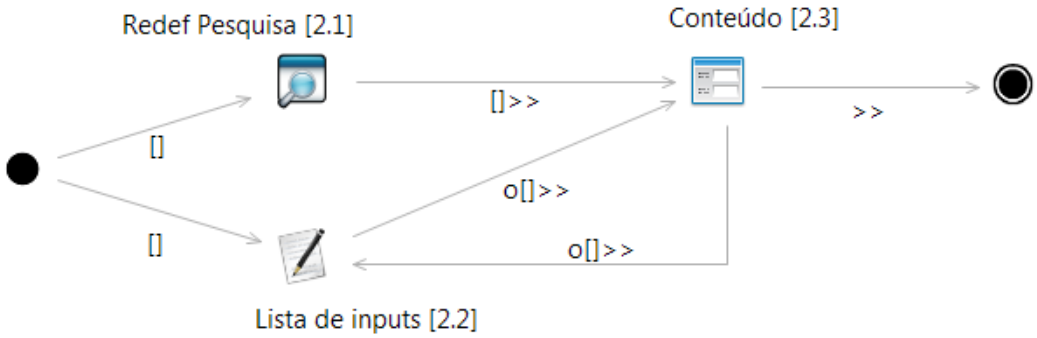


Figura 4.4: Modelo do formulário Resultados

O ecrã de resultados pode ser decomposto em três partes distintas: redefinição da pesquisa, refinamento de pesquisa e resultados propriamente ditos (conteúdo).

Pode começar-se pela redefinição de pesquisa que mais não é do que uma nova pesquisa daí que seja utilizado um nó do tipo *"Find"*. Quanto ao refinamento da pesquisa, pode ver-se como uma série de inputs que vão sendo feitos, daí que se utilize o nó do tipo *"Input"*. Finalmente

o conteúdo apresenta um formato um pouco diferente e portanto será utilizado um nó do tipo *"Form"*, que permitirá detalhar esta parte da interface um pouco melhor.

Quanto às ligações utilizadas para ligar estes elementos, é de notar que feita a pesquisa inicial são apresentados resultados (caso existam) e, portanto, tanto a questão da redefinição da pesquisa como a questão do refinamento da pesquisa são meramente opcionais, visto que podemos interagir com o conteúdo sem que seja necessário passar por qualquer uma das duas partes anteriormente referidas.

Já no que concerne à ligação da redefinição da pesquisa para o conteúdo, como já foi dito anteriormente, redefinir a pesquisa é semelhante a fazer uma nova pesquisa, logo será tratada como a pesquisa inicial, isto é, existe uma passagem de informação da redefinição da pesquisa para o conteúdo.

A ligação do refinamento da pesquisa ao conteúdo é feito através de uma ligação de movimento de informação uma vez que quando se seleciona um dado input na lista de refinamentos este desaparece da lista para que não possa ser escolhido novamente. O mesmo acontece se pretendermos retirar uma das opções de refinamento. Nesse caso a opção de refinamento é movida do conteúdo, novamente, para a lista de refinamentos possíveis.

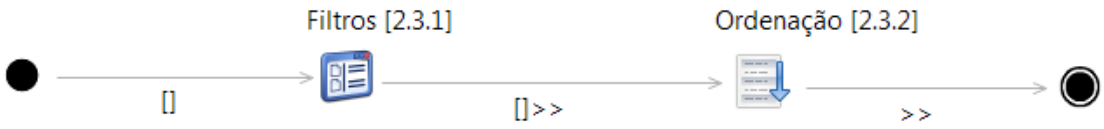


Figura 4.5: Modelo do formulário Conteúdo

Após a criação de todos os elementos que devem estar presentes no modelo, deve-se configurar os dados de teste a ser utilizados.

Nas figuras 4.6 e 4.7 podem ver-se algumas das configurações, no caso para os nós "Pesquisa [1]"e "Lista de Inputs [2.2]".

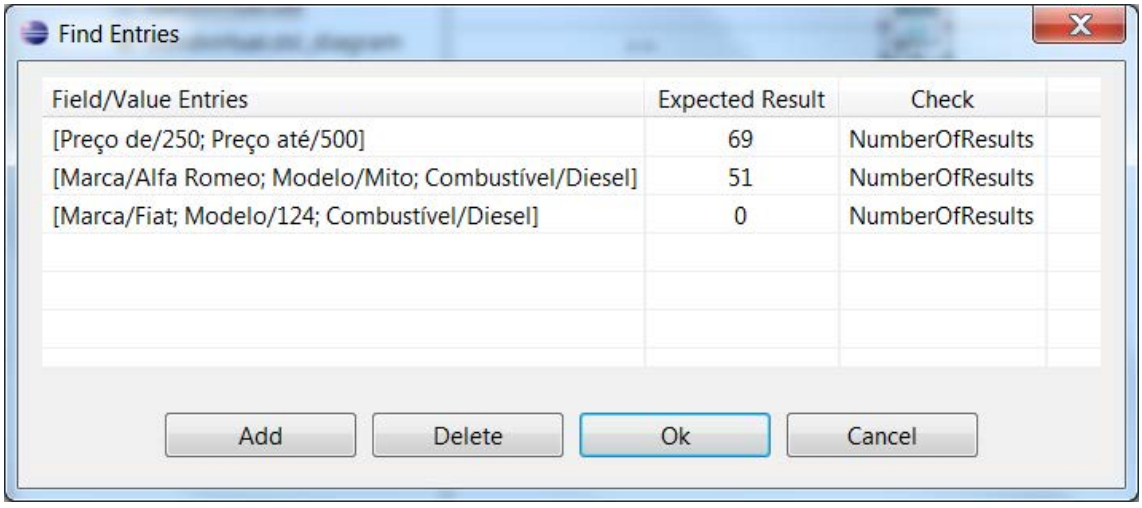


Figura 4.6: Configurações do nó "Pesquisa [1]"

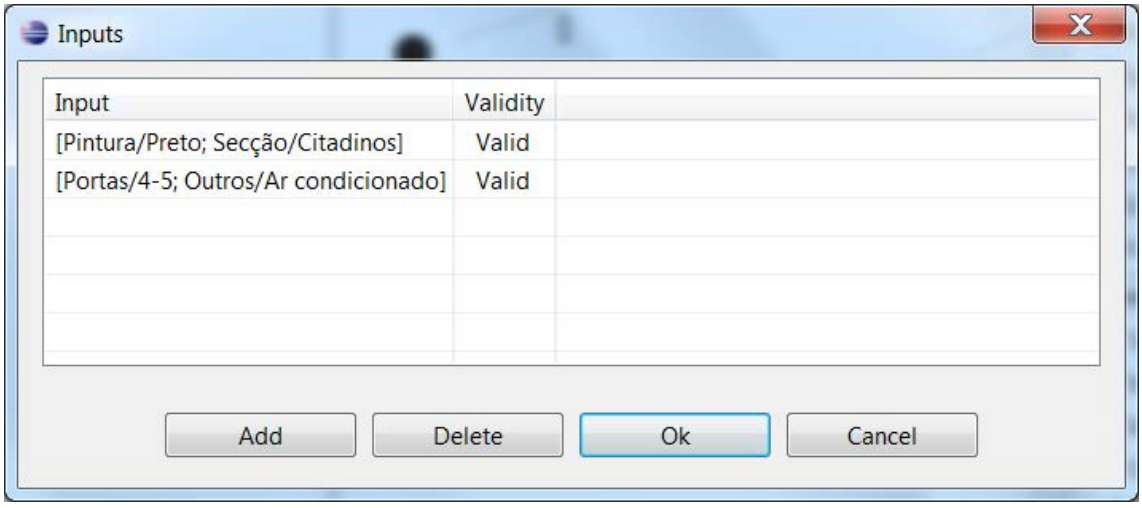


Figura 4.7: Configurações do nó "Lista de Inputs [2.2]"

Após a criação de configurações para todos os nós, pode então passar-se à geração dos caminhos de teste. Seguindo os passos descritos no capítulo anterior, escolhe-se a opção "Generate Test Paths"do menu de contexto, insere-se o número de vezes que os ciclos devem ser percorridos (caso estes existam no modelo) e o ambiente gera os caminhos.

Para este caso de estudo os caminhos gerados foram:

1. (Start, 1, 2, 2.3, 2.3.2, End)
2. (Start, 1, 2, 2.1, 2.3, 2.3.2, End)
3. (Start, 1, 2, 2.2, 2.3, 2.3.2, End)
4. (Start, 1, 2, 2.3, 2.3.1, 2.3.2, End)
5. (Start, 1, 2, 2.1, 2.3, 2.3.1, 2.3.2, End)
6. (Start, 1, 2, 2.2, 2.3, 2.3.1, 2.3.2, End)

O caminho 1 é a situação em que se faz uma pesquisa e depois no ecrã de resultados se faz uma ordenação, sem redefinir ou refinar a pesquisa. No caminho 2 faz-se a primeira pesquisa e no ecrã de resultados redefine-se a pesquisa seguindo-se a ordenação. O caminho 3 executa a pesquisa inicial, refina essa mesma pesquisa e depois parte para a ordenação. Os caminhos 4, 5 e 6 são semelhantes a 1, 2 e 3, respetivamente, com a pequena diferença de que antes da ordenação se fazem algumas alterações aos dados passados ao conteúdo, alterando os resultados da pesquisa.

De ressalvar que caso existam erros de integridade ou falta de configurações, o sistema não permite prosseguir para a geração de caminhos e alerta o utilizador de que deverá resolver os erros reportados no separador de erros do ambiente (ver figura 4.8).

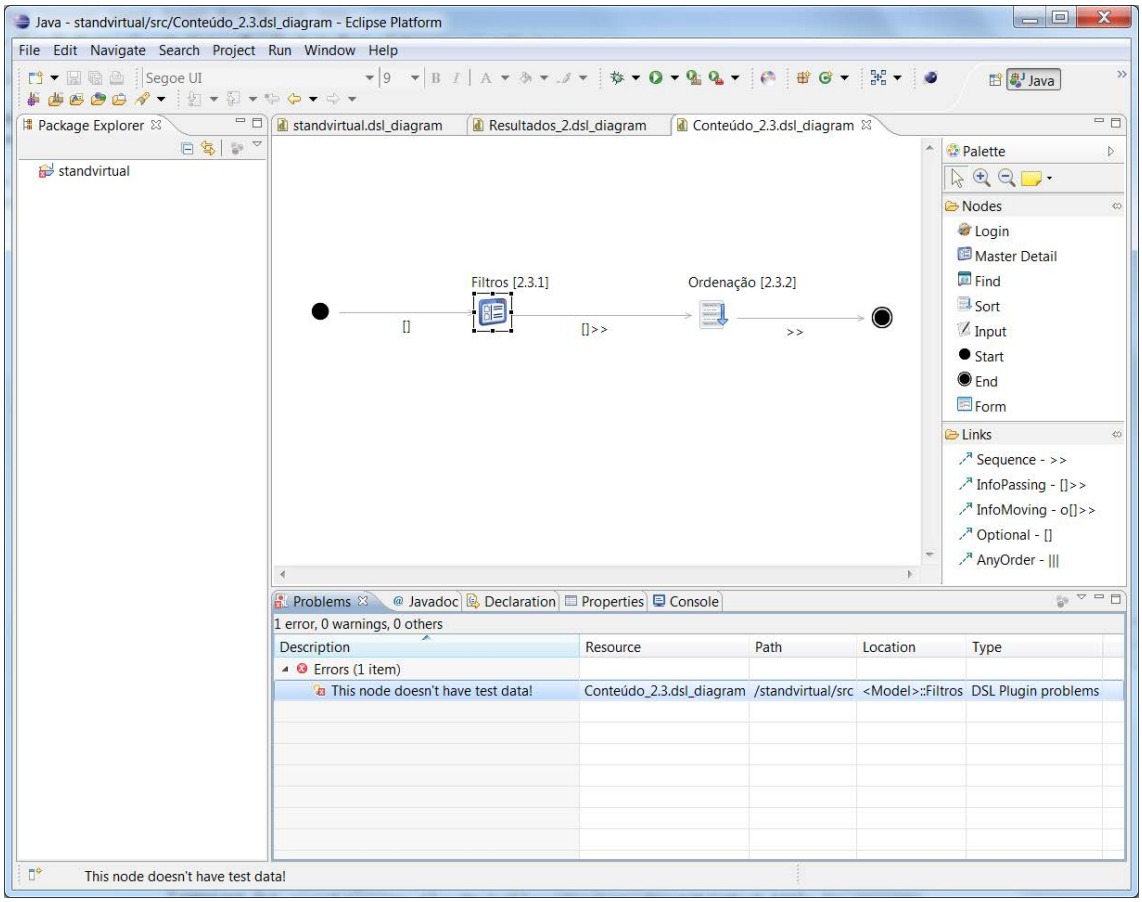


Figura 4.8: Alerta de erros no modelo

4.2 ERA

A ERA é uma imobiliária que tem um sítio na Web (www.ERA.pt) onde se pode pesquisar diversos tipos de imóveis. Existe uma série de critérios de entre os quais o utilizador pode escolher para efetuar a pesquisa. Na figura 4.9 pode ver-se o aspeto da página inicial.

Figura 4.9: Página inicial do sítio ERA.pt

Pode perceber-se pelo aspeto da página que tem várias semelhanças com o caso de estudo anterior, no entanto, o facto de a linguagem ainda ser um pouco limitada no número de padrões que suporta leva a que as aplicações sejam sempre semelhantes.

Mesmo tendo em conta essas potenciais semelhanças, ainda assim é possível modelar de forma completamente diferente, como se pode perceber pelo modelo criado para esta aplicação presente na figura 4.10.

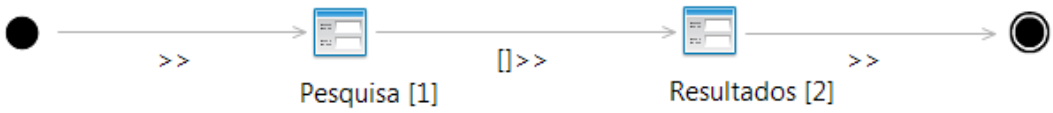


Figura 4.10: Modelo da aplicação ERA.pt

Comparando com o caso de estudo anterior em vez de utilizar o elemento *"Find"* optou-se por uma modelação um pouco diferente, fazendo uso do formulário e dentro desse formulário criaram-se diversos nós cada um representando cada um dos inputs presentes na página. O detalhe do formulário pode ser visto na figura 4.11.

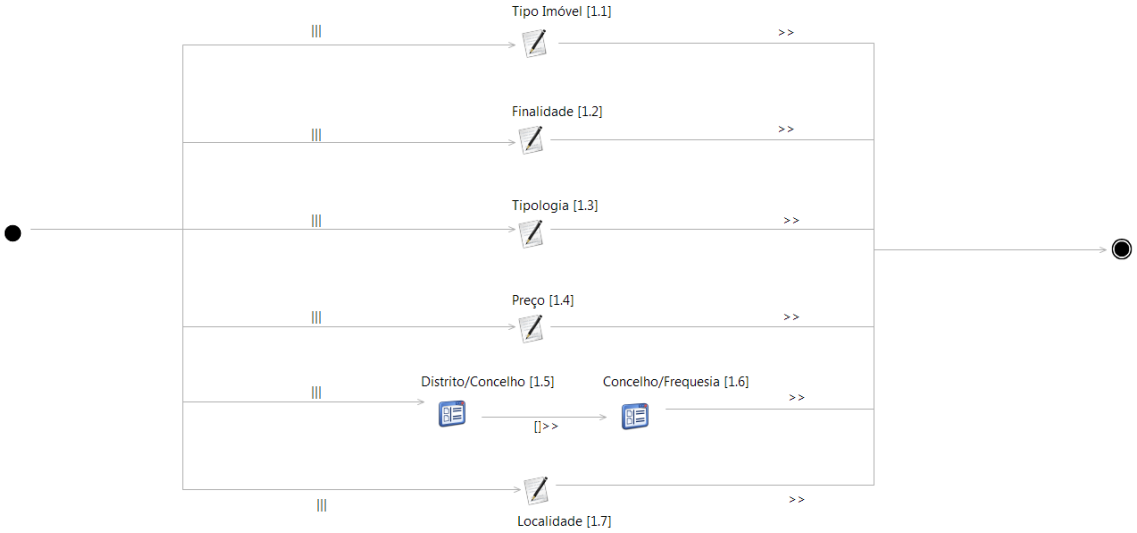


Figura 4.11: Detalhe do formulário de pesquisa da aplicação ERA

De notar a utilização de relações do tipo "Any Order", já que as informações podem ser preenchidas em qualquer ordem, ou seja, é indiferente se fazemos primeiro input do tipo de imóvel ou da tipologia, por exemplo. De notar também a passagem de informação entre os dois *"Master/Detail"* uma vez que os dados no nó "Distrito/Concelho"influenciam os dados no nó "Concelho/Distrito".

Para terminar a modelação falta ainda modelar o formulário que representa a página de apresentação de resultados. Esta página de apresentação de resultados (ver figura 4.12), é constituída por uma listagem de todos os imóveis que obedecem aos parâmetros de pesquisa inseridos anteriormente e possibilita ainda a ordenação através de uma série de critérios.

Perante esta GUI e os elementos existentes, a modelação cinge-se à criação de um nó de ordenação (ver figura 4.13).

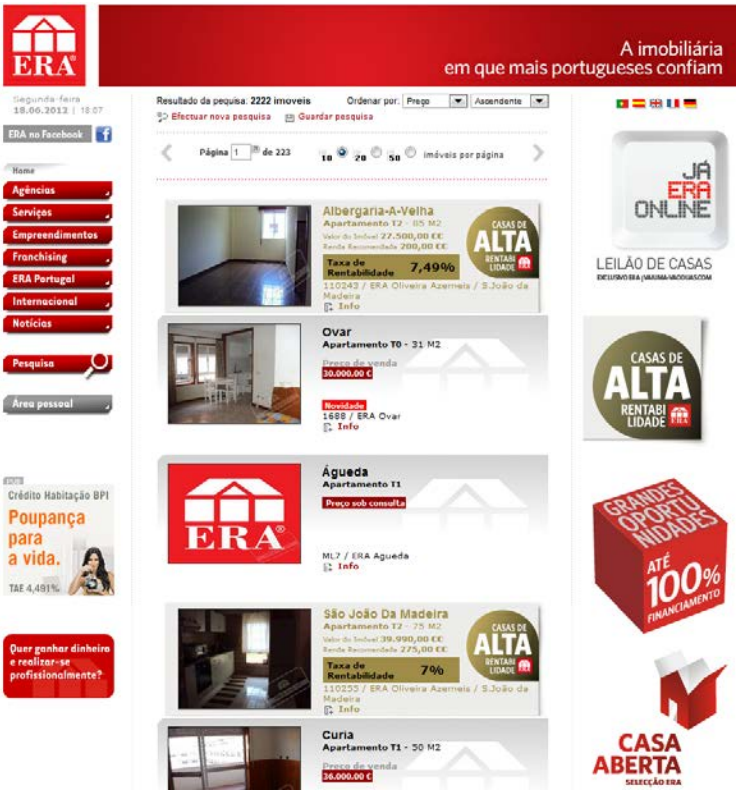


Figura 4.12: Página de resultados da aplicação ERA.pt

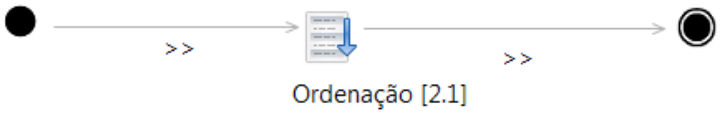


Figura 4.13: Detalhe do formulário de resultados da aplicação ERA

Após a modelação, o passo seguinte é a configuração de todos os elementos. Nas figuras 4.14, 4.15, 4.16 e 4.17 podem ver-se alguns dos dados utilizados nas configurações.

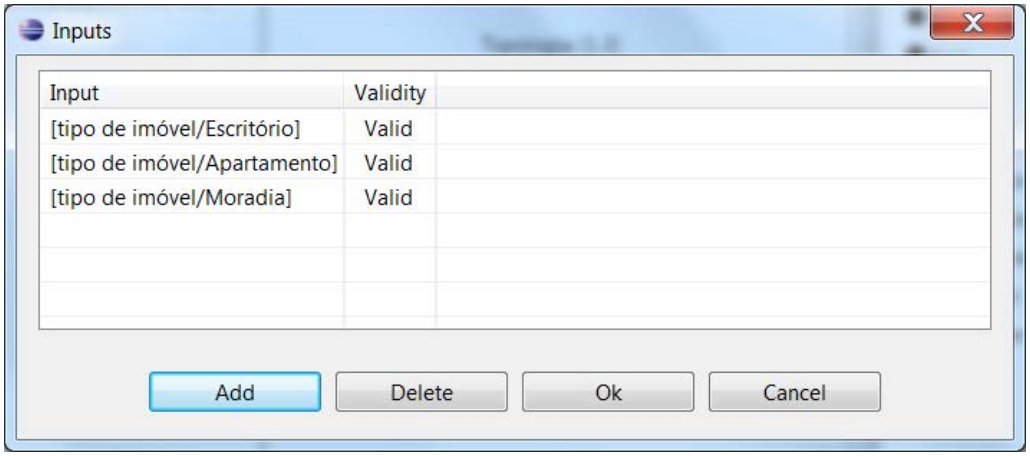


Figura 4.14: Configurações do nó "Tipo Imóvel [1.1]"

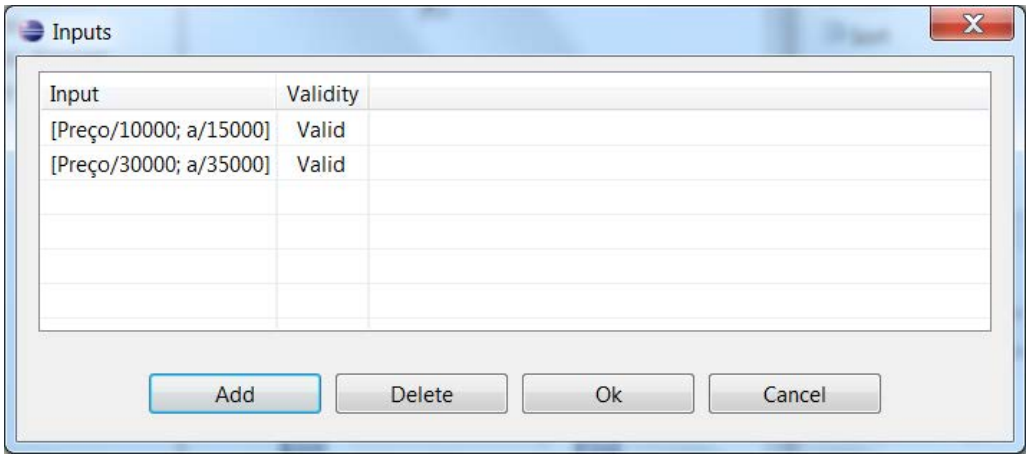


Figura 4.15: Configurações do nó "Preço [1.4]"

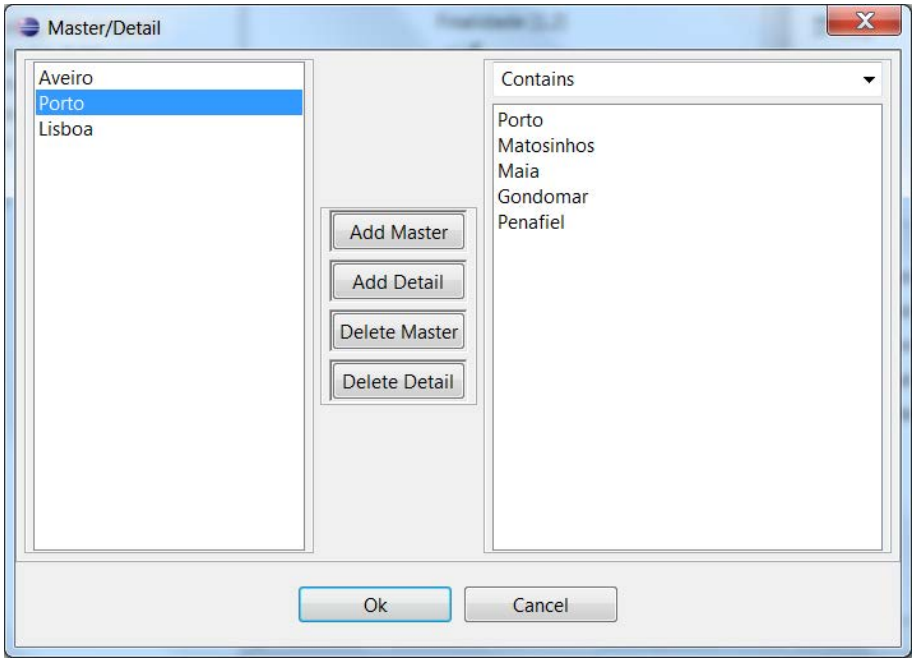


Figura 4.16: Configurações do nó "Distrito/Concelho [1.5]"

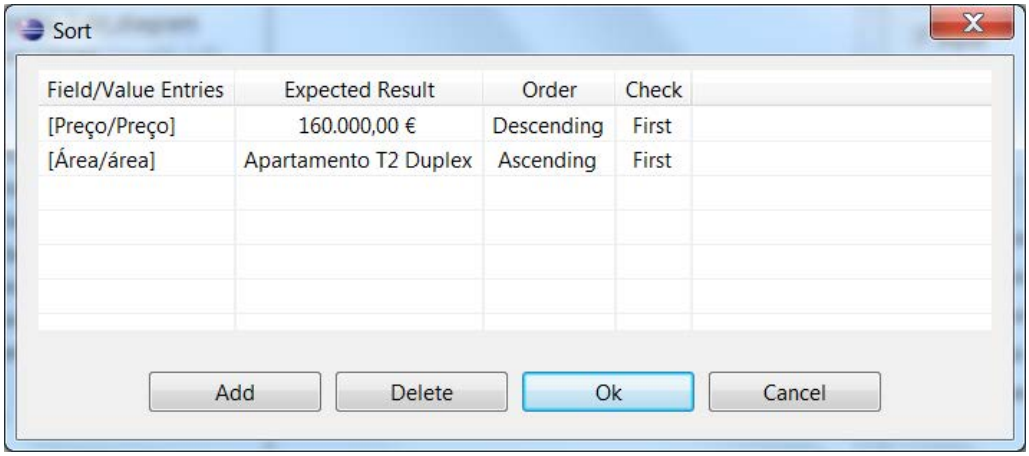


Figura 4.17: Configurações do nó "Ordenação [2.1]"

De seguida, é efetuada a geração de caminhos.

Apenas para comprovar, mais uma vez, a capacidade de deteção de problemas do ambiente, retirou-se o nó *"Start"* e tentou-se então gerar caminhos de teste. A resposta do ambiente foi imediata reportando um erro por falta do nó *"Start"*, como se pode ver na figura 4.18.

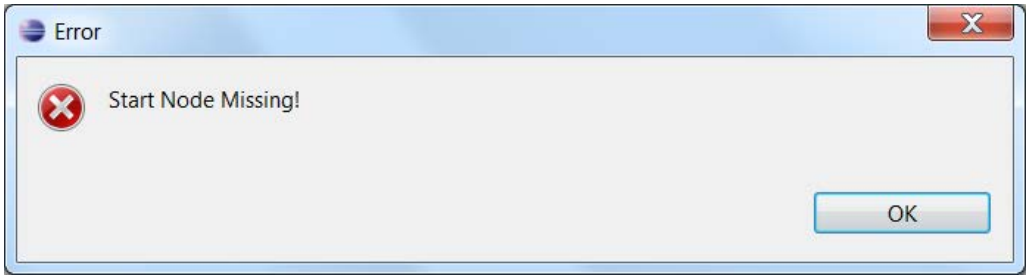


Figura 4.18: Diálogo de erro por falta do nó *"Start"*

Após a restituição do nó *"Start"*, volta-se a tentar a geração de caminhos de teste. Neste caso, devido à existência de relações do tipo "Any Order", serão gerados vários caminhos em tudo idênticos exceto na ordem dos diferents inputs. Devido ao elevado número de caminhos criados e à sua similaridade estes não serão listados exaustivamente, no entanto, a título informativo pode dizer-se que o número total de caminhos é de 720 caminhos.

4.3 Resumo

A utilização de casos de estudo para confirmar se uma aplicação faz o que se pretende é das formas mais utilizadas atualmente. O princípio estipula que se determinada aplicação cumpre os objetivos dentro de alguns exemplos então é provável que os cumpra para outros casos semelhantes.

Como os casos acima permitiram demonstrar, o ambiente de modelação faz tudo o que dele se esperava, permitindo criar modelos, configurar dados de teste e gerar os caminhos de teste. É também capaz de detetar uma série de problemas, como por exemplo, a falta de nós *"Start"* e *"End"*, ou a falta de dados de configuração.

Quanto ao tempo de geração de casos de teste, este não é muito elevado, no entanto, a utilização de modelos extremamente complexos ou a possibilidade de muitas combinações (como no segundo caso de estudo), torna o processo um pouco mais lento.

É também necessário relembrar que o tipo de aplicações escolhidas para teste é muito semelhante, o que em muito se deve à linguagem, uma vez que neste momento esta apresenta apenas alguns dos padrões de comportamento que existem nas GUIs.

Capítulo 5

Conclusões e Trabalho Futuro

Atualmente a existência de uma interface gráfica para uma aplicação informática é (quase) absolutamente necessária, devido à facilidade de interação utilizador-aplicação que esta componente adiciona. Apesar disso, o teste desta componente tão importante continua a ser feito de forma essencialmente manual.

No sentido de contrariar esta tendência urge encontrar formas de automatizar e sistematizar as tarefas de teste de GUIs. Uma das melhores formas de o fazer será, porventura, a utilização de teste baseado em modelos devido aos bons resultados apresentados noutras implementações já existentes. O problema desta abordagem prende-se com a construção dos modelos que nem sempre é simples.

Este projeto visa então aproveitar as vantagens de teste baseado em modelos procurando simultaneamente mitigar o problema da construção dos modelos através da utilização de padrões de comportamento dos componentes das interfaces gráficas.

5.1 Satisfação dos Objetivos

O objetivo deste projeto passa então pela criação de um ambiente de modelação que permita a criação dos modelos e a verificação da sua integridade, a configuração dos nós do modelo e a geração de caminhos de teste.

Para a criação do ambiente estudou-se o que já existe nesta área de ambientes de modelação para perceber qual seria o mais interessante de utilizar. Decidiu-se entao pela escolha do Eclipse Graphical Modelling Tools pela sua flexibilidade e por estar associado ao um reconhecido software de desenvolvimento como é o Eclipse.

O ambiente criado apresenta todas as funcionalidades pretendidas permitindo:

- Criar modelos com todos os elementos da linguagem de modelação;
- Verificar se o modelo está bem construído, nomeadamente no que concerne à existência de nós como o *Start* e o *End*;
- Configurar dados de teste para cada nó de tipo *Behavioural*;

- Geração dos caminhos de teste.

O ambiente de modelação permite criar modelos fazendo uso de todos os tipos de nós e arestas presentes na linguagem. Durante e após a construção do modelo o utilizador pode, em qualquer momento, validar o modelo, percebendo se alguma das regras de integridade da linguagem está a ser quebrada.

Também relativamente à configuração de dados de teste para cada nó do tipo *Behavioural*, o ambiente responde positivamente, permitindo efetuar diversas configurações de dados de teste.

Finalmente, é também possível a geração de caminhos de teste de forma simples. Esta geração de caminhos tem em conta os diferentes tipos de nós e arestas presentes no modelo, respondendo corretamente conforme os elementos presentes.

Em todas as aplicações informáticas existem pormenores que podem ficar um pouco melhor, no entanto, como se pode constatar pelo documento apresentado, todos os objetivos foram cumpridos.

5.2 Trabalho Futuro

Em termos do que pode ser feito no futuro, a prioridade passará por permitir o mapeamento dos nós presentes no modelo para os controlos existentes na interface gráfica. Esta fase não será fácil devido à dificuldade de identificação dos elementos em interfaces gráficas, um problema bem reconhecido nesta área.

Outro ponto que deverá ser trabalhado no futuro prende-se com a geração dos testes propriamente ditos a partir das configurações de dados de teste e dos caminhos gerados. Com estes dois pontos será então possível correr efetivamente os testes nas GUIs.

Como último ponto, será necessário também trabalhar no sentido de identificar mais padrões de comportamento de elementos das GUIs, permitindo assim estender a linguagem, e consequentemente o ambiente, tornando-a cada vez mais dotada e capaz de testar um maior número de GUIs.

Referências

[BBN04] Mark Blackburn, Robert Busser e Aaron Nauman. Why model-based test automation is different and what you should know to get started. In *International Conference on Practical Software Quality and Testing*, pages 212–232, 2004.

[BM07] P.A. Brooks e A.M. Memon. Automated gui testing guided by usage profiles. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 333–342. ACM, 2007.

[CPFA10] Marco Cunha, A.C.R. Paiva, H.S. Ferreira e Rui Abreu. PETTool: A pattern-based GUI testing tool. In *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*, volume 1, pages V1–202. IEEE, 2010.

[Cun08] H.C. Cunningham. A little language for surveys: Constructing an internal DSL in Ruby. In *Proceedings of the 46th Annual Southeast Regional Conference on XX*, pages 282–287. ACM, 2008.

[EEHT05] Karsten Ehrig, Claudia Ermel, Stefan Hänsgen e Gabriele Taentzer. Generation of visual editors as eclipse plug-ins. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering - ASE '05*, pages 134–143. ACM, 2005.

[EFW01] I.K. El-Far e J.A. Whittaker. Model-Based Software Testing. *Encyclopedia of Software Engineering*, pages 1–22, 2001.

[EJ01] Robert Esser e J.W. Janneck. A framework for defining domain-specific visual languages. In *Workshop on Domain Specific Visual Languages, ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA-2001)*, 2001.

[FP10] Martin Fowler e R. Parsons. *Domain-specific languages*. Addison-Wesley Professional, 2010.

[Gra08] Grandite. *Open ModelSphere 3.0 - Developer Guide*. 2008.

[Gra09] Grandite. *Open ModelSphere - User Guide*. 2009.

[Gro09] Richard C Gronback. *ECLIPSE MODELING PROJECT - A Domain-Specific Language Toolkit*. Addison-Wesley, 2009.

[GT06] Richard C Gronback e Artem Tikhomirov. Developing a Domain-Specific Modeler with the Eclipse Graphical Modeling Framework (GMF). In *a workshop of the 20th European Conference on Object-Oriented Programming (ECOOP 2006)*, 2006.

[Hud98] Paul Hudak. Modular domain specific languages and tools. In *Software Reuse, 1998. Proceedings. Fifth International Conference on*, pages 134–142. IEEE, 1998.

[LKKL05] Minkyu Lee, Hyunsoo Kim, Jeongil Kim e Jangwoo Lee. *StarUML 5 . 0 - Developer Guide*. 2005.

[LSTM04] S.F. Lopes, Carlos Silva, Adriano Tavares e J.L. Monteiro. Extending argoUML for real-time UML. In *Proceedings of the IASTED International Conference Advances in Computer Science and Technology*. IASTED/ACTA Press, 2004.

[MBN03] Atif Memon, I. Banerjee e A. Nagarajan. GUI ripping: Reverse engineering of graphical user interfaces for testing. In *Proceedings of the 10th Working Conference on Reverse Engineering*, pages 260–269, 2003.

[Mem02] A.M. Memon. GUI testing: Pitfalls and process. *IEEE Computer*, 35(8):87–88, 2002.

[PFTV05] A. Paiva, J. C. P. Faria, Nikolai Tillmann e R. F. A. M. Vidal. A model-to-implementation mapping tool for automated model-based GUI testing. In *Proceedings of the 7th international conference on Formal Methods and Software Engineering (ICFEM'05)*, pages 450–464, 2005.

[RR00] Jason E Robbins e David F Redmiles. Cognitive Support , UML Adherence , and XMI Interchange in ArgoUML. *Information and Software Technology*, 42(2):79–89, 2000.

[Som06] Ian Sommerville. *Software Engineering: 8th Edition*. Addison Wesley, 2006.

[UL07] Mark Utting e Bruno Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann, 2007.

[VKV00] A. Van Deursen, P. Klint e Joost Visser. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, 35(6):26–36, 2000.

[vW12] M van Welie. Welie.com - Patterns in interaction design, Accessed January 2012.