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 Cristina Ramada Paiva Pimenta (Doutora)

07 de Fevereiro de 2012

Resumo

A utilização de interfaces gráficas nas aplicações informáticas é visto 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, especialmente através de teste baseado em modelos. Estas melhorias permitirão no fundo, 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. Esta dissertação pretende estudar o estado da arte na área de teste de GUI e ainda proceder a um estudo de ferramentas de modelação existentes para poder proceder à criação do ambiente de modelação necessário ao projeto mencionado anteriormente.

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.

This dissertation aims to study the state of the art in the GUI testing area and conduct a study on existing modeling tools so that it will be possible to make a modelling environment that answers all the needs of the project mentioned earlier.

Conteúdo

CONTEÚDO

Lista de Figuras

LISTA DE FIGURAS

Lista de Tabelas

LISTA DE TABELAS

Abreviaturas e Símbolos

DSL	*Domain Specific Language*
GMF	*Graphical Modeling Framework*
GUI	*Graphical User Interface*
MTB	*Model-Based Testing*
OCL	*Object Constraint Language*
UML	*Unified Modeling Language*

Capítulo 1

Introdução

Este primeiro capítulo apresenta o contexto da dissertação, o projecto do qual faz parte, por que motivo surge e quais os objectivos 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], cotando-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 esta dissertação faz parte.

1.2 Projecto

O projeto "Pattern-Based GUI Testing" [CPFA10] é um projeto criado em conjunto pela Faculdade de Engenharia da Universidade do Porto (FEUP), pela Universidade do Minho (UM) e pela 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 GUI's, através da técnica de teste baseado em modelos.

Um dos aspetos menos positivos de teste baseado em modelos prende-se com o esforço necssário para construir os modelo 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 superior através de um ambiente de modelação e de uma linguagem específica do domínio com base nos padrões de comportamento dos componentes da própria GUI, procurando assim minimizar o esforço de construção do modelo.

1.3 Descrição do Problema

Sendo uma das componentes do projeto permitir a criação de modelos através de uma linguagem específica de domínio (DSL) criada numa fase anterior do projeto e que permite modelar interfaces gráficas com base em padrões comportamentais dos seus componentes, o problema que se pretende ver respondido nesta dissertação passa pela criação de um ambiente de modelação capaz de suportar a DSL referida.

Este ambiente de modelação em conjunto com a ferramenta de engenharia reversa permitiria a criação e manutenção de modelos de interfaces gráficas, que por sua vez levariam 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. Inserida neste projeto esta dissertação 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 a DSL especificada;
- Verificar a integridade do modelo construído;
- Alertar o utilizador caso as propriedades de algum dos elementos não permita a geração de todos os casos de teste teoricamente possíveis;
- Gerar casos de teste a partir dos modelos criados.

No final desta dissertação pretende-se obter um ambiente de modelação que permita a criação de um modelo como o exemplificado na figura 1.1.

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?

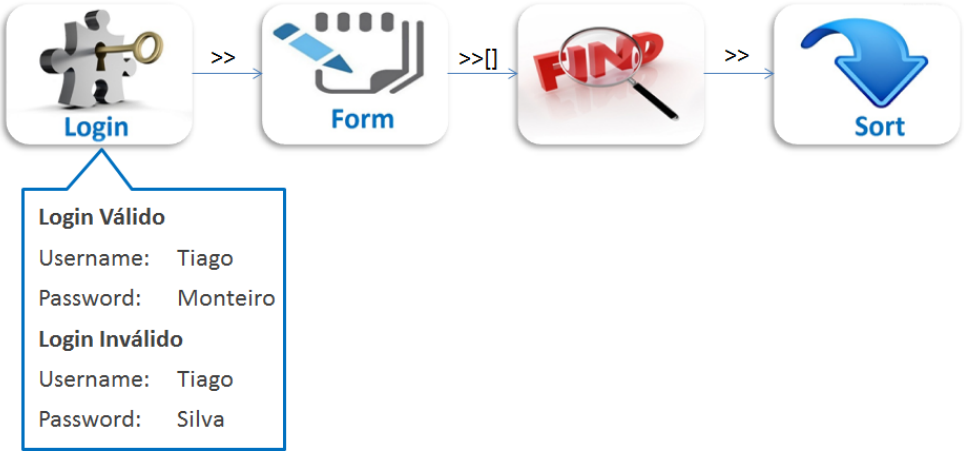

Figura 1.1: Exemplo de modelo

- Que ambientes de modelação existem atualmente?
- Dos ambientes que existem qual seria mais adequado para suportar a DSL para teste de GUIs?

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 possibilite atingir os objetivos traçados.

1.5 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais 2 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 será escolhida uma para a fase de implementação.

No capítulo 3 são apresentadas as conclusões do estudo efectuado e é apontado o trabalho a efectuar no futuro.

Capítulo 2

Revisão Bibliográfica

Este capítulo apresenta o estudo teórico de base, começando por uma breve análise sobre DSLs, segue-se uma secção sobre teste baseado em modelos, especialmente na área de 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]. O que isto significa é que uma DSL 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 utilizadores.

Para além da divisão em DSL internas ou externas, pode também considerar-se uma divisão pelo tipo de notação: visual ou textual. Como será fácil de perceber 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. Como se pode perceber por estes trabalhos 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. 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 de DSL ou uma ferramenta de modelação que permita aumentar as suas funcionalidades. 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 numa 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.

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.

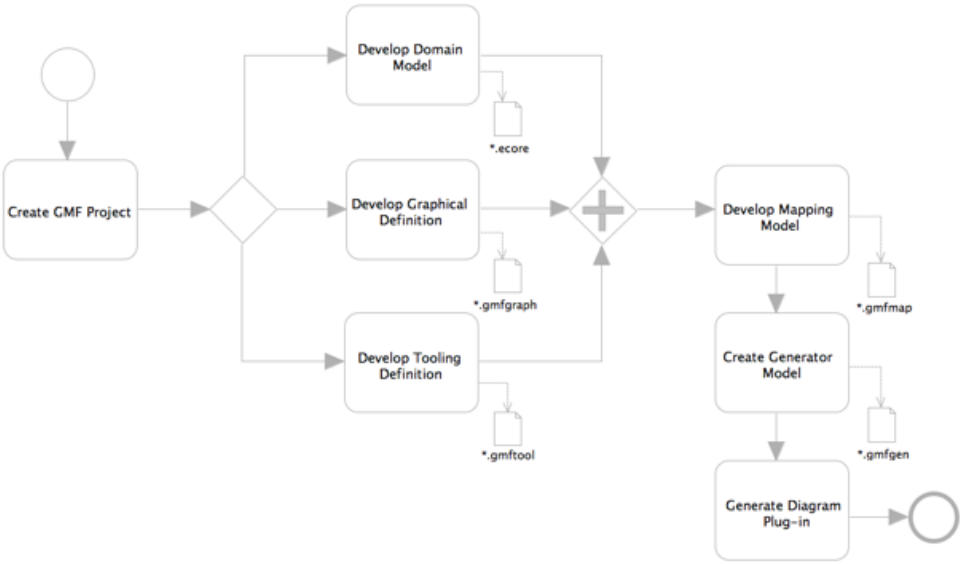

Figura 2.1: GMF Overview [GT06]

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	3
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].

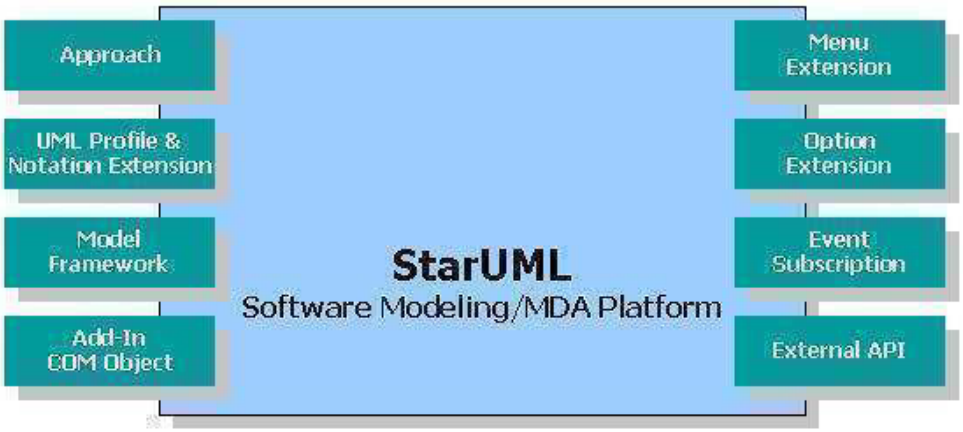

Figura 2.2: Arquitetura do ambiente de modelação StarUML [LKKL05]

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	2
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 parece cotar-se pelas 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,

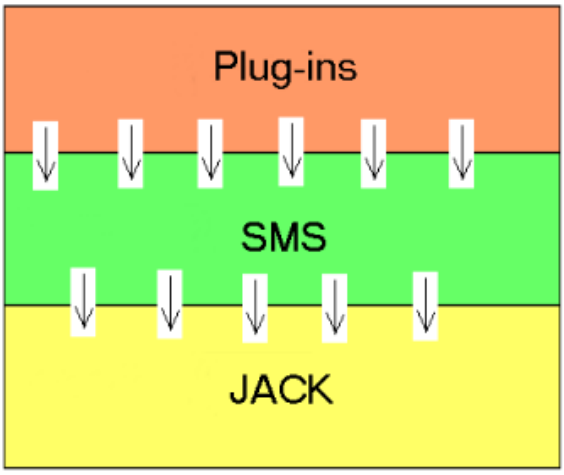

Figura 2.3: Arquitetura do ambiente de modelação Open Modelshpere [Gra08]

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	2
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

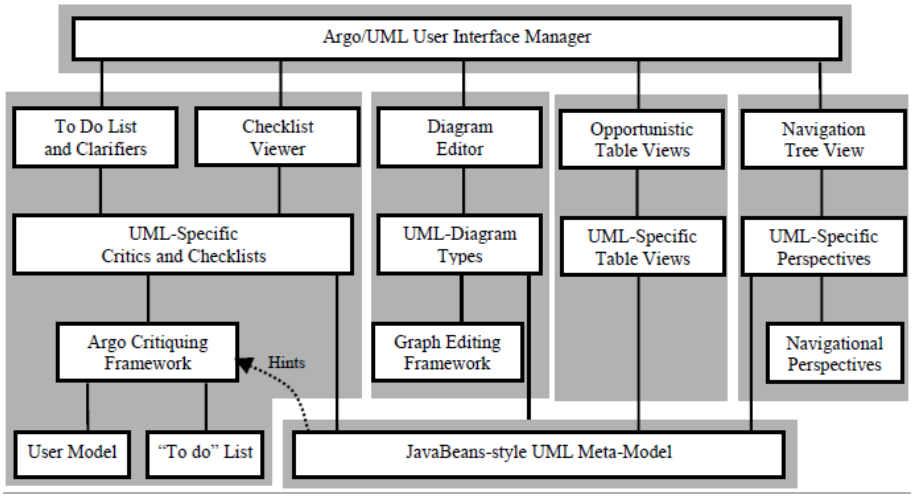

Figura 2.4: Arquitetura do ambiente de modelação ArgoUML [RR00]

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

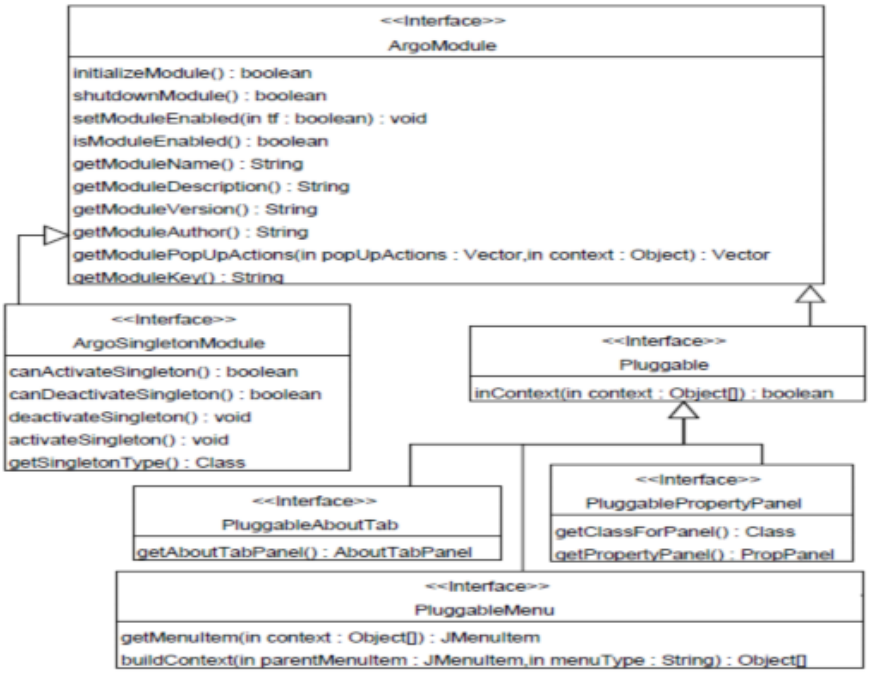

Figura 2.5: Interface de extensão do ArgoUML [LSTM04]

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	2
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	3	2	2	2
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 é mais fácil de costumizar aos 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, no entanto, apresenta também os problemas que existem nos trabalhos nesta área.

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 exaustivo na deteção de 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 permite 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

Conclusões e Trabalho Futuro

Após a apresentação do projeto a desenvolver e do estudo realizado no âmbito do mesmo, apresentam-se as conclusões do trabalho desenvolvido até ao momento e o trabalho a realizar durante a fase de implementação.

3.1 Conclusões

O estudo realizado permitiu encontrar alguns problemas no que já existe na área de teste de GUI com uso de teste baseado em modelos.

Estes pontos menos positivos nos trabalhos estudados abrem espaço à procura de soluções que ataquem estes problemas e é nesse sentido que surge o projeto agora a desenvolver, tentando atacar a questão de apanhar diversos tipos de erros e não só os que representam avarias do sistema, mas também procurar diminuir o esforço de construção dos modelos para que a construção e manutenção de casos de teste se torne mais simples, permitindo em última análise que se crie software com maior qualidade.

Neste âmbito de diminuição do esforço de criação de modelos, procura-se criar um ambiente de modelação específico para a DSL existente que permita construir modelos de forma mais rápida e eficiente. O estudo dos ambientes de modelação existentes permitiu perceber o que já existe atualmente e perceber o que é necessário para se obter o ambiente que se pretende.

Os resultados esperados passam por um ambiente simples que permita a configuração de todos os nós e arestas dos modelos. Espera-se também que o ambiente permita verificar a integridade dos modelos e conferir se as propriedades específicas de cada nó e aresta estão corretamente preenchidos. Finalmente é ainda expectável que seja possível gerar casos de teste a partir dos modelos criados.

3.2 Trabalho Futuro

Para atingir os resultados pretendidos procedeu-se à elaboração de um plano de trabalhos apresentado na figura 3.1.

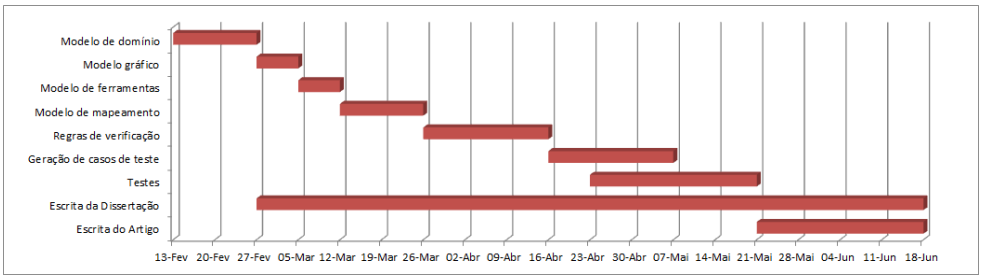

Figura 3.1: Planeamento do trabalho futuro

- **Modelo do domínio** (2 semanas) - esta fase centra-se na criação do modelo do domínio, na definição das entidades da linguagem e das suas propriedades.
- **Modelo gráfico** (1 semana) - criação do modelo gráfico, como indicado pelo nome, permite a definição das imagens a ser apresentadas durante a criação dos modelos, definindo também que entidades são representadas como nós e que entidades são representadas como arestas.
- **Modelo de ferramentas** (1 semana) - criação do modelo de ferramentas, que permite a definição das acções de criação, dos elementos a aparecer no painel de criação de modelos do editor, bem como outras ações ligadas à interface do ambiente de modelação.
- **Modelo de mapeamento** (2 semanas) - criação do modelo de mapeamento, que é o modelo que faz a ligação entre os três modelos anteriores.
- **Regras de verificação** (3 semanas) - criação das regras que permitirão executar a verificação da integridade dos modelos.
- **Geração de casos de teste** (3 semanas) - fase para aplicação das técnicas de geração dos casos de teste para cada tipo de nó, tendo em consideração as sequências possíveis permitidas pelo modelo.
- **Testes** (4 semanas) - fase de execução de testes de sistema e eventuais modificações, caso seja necessário.
- **Escrita da dissertação** (16 semanas) - em paralelo com a atividade de implementação, será escrita a dissertação.
- **Escrita do artigo** (4 semanas) - nas 4 últimas semanas, e em paralelo com a escrita da dissertação será escrito um artigo científico de apresentação do trabalho efetuado.

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.

REFERÊNCIAS

[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.

[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.