FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Geração de Casos de Teste a partir de Modelos de Tarefas

Ana Sofia Barros Barbosa

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Ana Cristina Ramada Paiva Pimenta (Professora, FEUP)

28 de Junho de 2010

Geração de Casos de Teste a partir de Modelos de Tarefas

Ana Sofia Barros Barbosa

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Raul Fernando de Almeida Moreira Vidal (Professor Associado)

Vogal Externo: José Francisco Creissac Freitas Campos (Professor Auxiliar)

Orientador: Ana Cristina Ramada Paiva Pimenta (Professora Auxiliar)

---

22 de Julho de 2010

Resumo

Hoje em dia, a forma mais comum de interacção entre o software e o utilizador é realizada recorrendo a interfaces gráficas com o utilizador (GUIs). A qualidade da GUI é um factor bastante importante que irá influenciar o utilizador a utilizá-la ou não. Desta forma é necessário testá-la convenientemente. Este tipo de testes enfrenta diversas dificuldades devido às características da GUI o que acarreta grandes despesas em termos de custo e tempo. Actualmente, existem diversas ferramentas de teste de GUIs. No entanto, muitas dessas ferramentas não automatizam a geração de casos de teste. Para automatizar a geração de casos de teste, é possível recorrer aos métodos de teste baseado em modelos formais. No entanto, estes métodos apresentam diversas dificuldades, principalmente a nível de construção dos modelos. Uma forma de ultrapassar essas dificuldades consiste em aumentar o nível de abstracção dos modelos recorrendo para isso a modelos de tarefas. Contudo, estes modelos apresentam apenas situações previstas e correctas de funcionamento não descrevendo os erros comuns que o utilizador comete na sua utilização.

Esta dissertação tem como objectivo analisar a viabilidade de utilização de modelos de tarefas para testar GUIs face a erros do utilizador. Desta forma, é apresentada uma abordagem que contribui para a introdução desse aspecto nas abordagens existentes.

Numa primeira etapa foi realizado um estudo sobre notações de modelação de tarefas e de ferramentas de teste baseado em modelos existentes, analisando as características e vantagens e desvantagens das mesmas. Numa segunda etapa foi definida uma abordagem para solucionar o problema em questão.

A abordagem apresentada nesta dissertação propõe a modelação das GUIs recorrendo à notação CTT e apresenta um algoritmo de realização de transformações aos modelos originais introduzindo erros típicos do utilizador. Para a definição deste algoritmo foi realizado um estudo mais aprofundado dos erros típicos que os utilizadores cometem ao interagir com uma GUI. De seguida foram analisadas diversas aplicações existentes de forma a detectar padrões na construção dos seus modelos. O algoritmo proposto apresenta uma identificação desses padrões e apresenta uma estratégia de alteração do modelo original de forma a criar modelos de teste para cada padrão detectado.

Para validar a abordagem definida foi desenvolvida a ferramenta CMT Tool. Esta ferramenta recebe um modelo de tarefas e aplica o algoritmo definido na abordagem, gerando diversos modelos de teste. Para avaliar a qualidade dos modelos de teste gerados foram seleccionadas diversas aplicações com GUIs como caso de estudo.

Os resultados obtidos pela análise dos casos de estudo permitem concluir que a abordagem definida permite testar as GUIs e detectar as falhas face a situações de comportamento inesperado do utilizador. Desta forma, pode-se concluir que a abordagem apresentada é válida e que os modelos de tarefas podem ser utilizados para testar erros típicos de comportamento do utilizador.

Abstract

Nowadays most software applications provide a graphical user interface (GUI). The quality of the GUI leads the user to use it or not, so it is important to test it. This kind of test faces too many difficulties that increase both effort and time spent. There are many tools for GUI testing, however they do not automate the generation of test cases. Model based testing allows the automation of test case generation. Though, it also faces many difficulties, mostly in what concerns to model specification. One way to surpass these difficulties is to increase the model's abstraction, using task models. Task models allow us to specify GUI behavior in terms of tasks. However, they only specify normal situations of correct user behavior which do not allow the GUI testing in situations of common user error.

The main goal of this dissertation is to analyze the feasibility of GUI testing against typical user errors using task models in the context of model based testing. So, an approach is presented that introduces this aspect.

Initially, it a study was conducted of many existent task notations and model based GUI testing tools. In a second phase an approach was proposed to solve the problem mentioned above.

The presented approach proposes the use of the CTT notation for GUI modeling, and presents an algorithm that transforms the original model introducing common user errors. To do so, typical user GUI interaction errors were studied. Then, many applications were analyzed in order to detect patterns in their task models. The proposed algorithm identifies these patterns and presents a strategy to introduce user errors in the original model in order to create test models for each detected pattern.

To validate the approach a tool was developed, the CMT Tool. This tool applies the proposed algorithm strategy to a task model generating many test models. In order to evaluate the quality of the test models generated many GUI applications were selected to be used as case studies.

The results observed in the case studies suggest that this approach really allows the detection of GUI faults under situations of unexpected user behavior. Overall, it is possible to conclude that this is a valid approach and that task models can be used to test typical user errors.

Agradecimentos

Agradeço à minha orientadora, a professora Ana Cristina Ramada Paiva Pimenta, da Faculdade de Engenharia da Universidade do Porto, por todo o apoio e orientação prestada que tornaram possível a realização desta dissertação.

Agradeço também ao professor José Creissac Campos, do departamento de Informática da Universidade do Minho, por me ter disponibilizado a ferramenta TOM e pela disponibilidade demonstrada no esclarecimento das minhas dúvidas.

Um agradecimento especial aos meus colegas Catarina Saraiva, Marco Cunha e Pedro Sousa, do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto, por todo apoio prestado durante o desenvolvimento desta dissertação, nos bons e maus momentos. Agradeço também ao Marco Cunha pelos conhecimentos transmitidos sobre padrões de interfaces gráficas com o utilizador e Framework UI Automation.

Por fim, agradeço aos meus pais, Armanda e António Barbosa, irmã, Joana, e namorado, João Rei, pelo encorajamento, suporte, paciência e atenção.

Muito Obrigada, sem vocês este desafio teria sido muito mais difícil!

Ana Sofia Barros Barbosa

Conteúdo

Lista de figuras

Lista de tabelas

Abreviaturas

API	Application Program Interface
CMT Tool	CTT Model Transformation Tool
CTT	ConcurTaskTrees
CTTE	ConcurTaskTrees Environment
FIT	Framework for Integrated Tests
FSM	Finite State Machine
GOMS	Goals, Operations, Methods, Selection Rules
GTA	Groupware Task Analysis
GUI	Graphical User Interface
GUITAR	GUI Testing framework
IDATG	Integrating Design and Automated Test Case Generation
PATHS	Planning Assisted Tester for graphical user interface Systems
PTS	Presentation Task Set
TKS	Task Knowledge Structure
UAN	User Action Notation
XML	eXtensible Markup Language
XPath	XML Path Language

Capítulo 1

Introdução

Actualmente, o teste de software é uma medida bastante importante a ter em conta no decorrer do desenvolvimento de software. Esta medida visa aumentar a confiança quer no correcto funcionamento do software, quer na sua qualidade. Desta forma, o teste de software pode ser definido como sendo uma técnica de investigação empírica que procura detectar os erros de um produto ou serviço de software, tendo como principal objectivo fornecer a todos os intervenientes (*stakeholders*) informações sobre a qualidade do mesmo.

O processo de detecção de erros do teste de software consiste na definição de casos de teste, na execução do software que se pretende testar com esses casos de teste e na comparação dos resultados observados nos testes com os esperados de forma a detectar os erros existentes. Neste sentido, o teste de software consiste numa técnica do processo de verificação e validação (V&V) que permite determinar se o produto está a ser bem construído e se funciona bem (verificação), e também determinar se o produto que está a ser construído é o correcto, ou seja, se o produto corresponde às necessidades e requisitos dos seus utilizadores (validação). [Tea02]

Devido à elevada complexidade do software é geralmente impossível testar todos os seus traços de execução, ou seja, todas as suas acções possíveis. Desta forma, de acordo com Dijkstra [Dij72], o teste de software pode ser utilizado para revelar a presença de erros, mas nunca a sua ausência.

1.1 Teste de Interfaces Gráficas

Hoje em dia, a forma mais comum de interacção entre o software e o utilizador é realizada através da utilização de interfaces gráficas com o utilizador, também conhecidas como GUIs (*Graphical User Interfaces*). Assim sendo, para avaliar o correcto funcionamento do software é

necessário avaliar também o correcto funcionamento da interface gráfica. Neste sentido, é necessária a realização de testes para testar o seu correcto funcionamento e verificar se estas se encontram definidas de acordo com as suas especificações.

O teste de interfaces gráficas permite aumentar a confiança no seu correcto funcionamento e qualidade. A qualidade da interface gráfica é um factor muito importante que irá influenciar bastante o utilizador a utilizá-la ou não sendo, por isso, importante a realização de testes. Contudo, a realização e desenvolvimento destes testes acarreta grandes despesas devido aos enormes custos e tempo requeridos. [PFV07]

As interfaces gráficas possuem diferentes características das do software tradicional (interfaces de linha de comandos) o que origina diversos problemas no teste do seu correcto funcionamento. Um dos seus principais problemas é o vasto espaço de interacções possíveis com a interface gráfica. Numa interface gráfica, cada sequência de eventos pode conduzir a diferentes estados, podendo levar a uma "explosão" de estados, o que faz com que o teste manual de todas as possibilidades de interacção não seja viável. Contudo, a automatização do teste de interfaces gráficas também não é trivial uma vez que a "explosão" de estados conduz também à "explosão" dos casos de teste necessários para testar todos os estados possíveis. Outro dos problemas da automatização dos testes é a necessidade e dificuldade de simular as acções do utilizador. Para simular as acções do utilizador surge a necessidade de mapear as acções semânticas (por exemplo, clicar num botão ou seleccionar um item) para acções a nível da programação, sendo necessária a escrita de código adaptador. Existe também a necessidade de transformar de alguma maneira os objectos visuais existentes nas interfaces em objectos de código de forma a possibilitar o teste das mesmas. Outro aspecto negativo que dificulta a automatização dos testes é o facto de que as interfaces gráficas respondem mais lentamente do que as APIs (*Application Program Interfaces*), pois têm que realizar o *rendering* dos seus objectos, entre outras acções. Este aspecto pode dificultar a distinção entre uma interface lenta ou bloqueada. [PFV07, Pim07, SCP08, Xie06]

Actualmente, existem algumas ferramentas de auxílio à geração de casos de teste para as interfaces gráficas e de avaliação da cobertura desses testes. No entanto, essas escassas ferramentas não automatizam a geração de casos de teste. As ferramentas do tipo Capture/Replay facilitam a construção de casos de teste, gravando as interacções do utilizador com a interface em scripts de teste que podem ser reproduzidas mais tarde, no entanto, requerem ainda muito esforço manual. [PFV07]

1.2 Teste de Interfaces Gráficas baseado em Modelos

Como foi referido, um dos maiores problemas do teste de interfaces gráficas com o utilizador (GUIs) consiste na automatização da geração dos casos de teste, que é uma tarefa

difícil. Para tentar combater esta dificuldade existe a possibilidade de utilizar teste baseado em modelos para gerar casos de teste de forma automática para testar as GUIs.

O teste baseado em modelos consiste no teste da implementação de um sistema de software através da comparação do estado e comportamento dessa implementação com um modelo de como este se deve comportar, o oráculo. Como já foi referido, este tipo de teste permite a geração automática dos casos de teste. Para testar um sistema, o processo de teste baseado em modelos pode ser dividido em cinco etapas principais, que se encontram representadas na Figura 1.1.

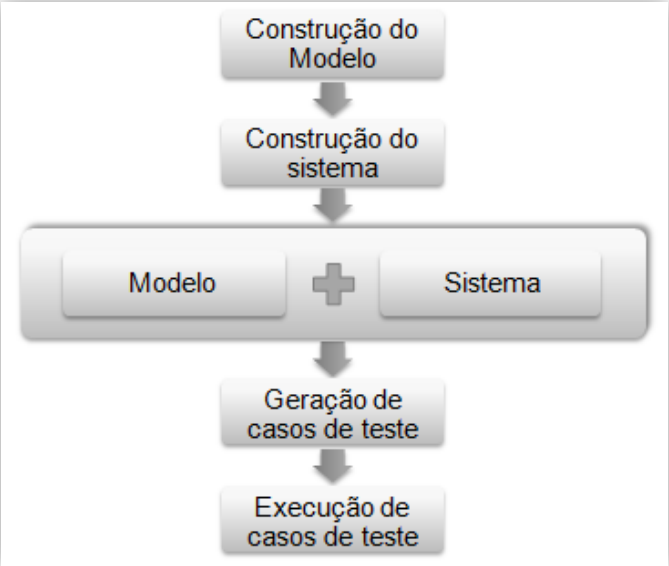


Figura 1.1: Processo do teste baseado em modelos

Como se pode observar na Figura 1.1, o primeiro passo do processo de teste baseado em modelos consiste na elaboração do modelo que irá representar o sistema a testar. Este modelo deve descrever o comportamento do sistema que se pretende testar. Seguidamente pode-se proceder então à implementação do sistema a testar. Com o modelo e o sistema criados gera-se automaticamente um conjunto de casos de teste que irão verificar o correcto funcionamento do sistema, comparando-o com o modelo. De seguida são executados esses casos de teste. Cada teste verifica a consistência entre o modelo e a implementação. Caso se verifique alguma inconsistência então significa que existe um erro na implementação ou até no próprio modelo que é necessário corrigir.

Existem algumas ferramentas de teste baseado em modelos que permitem testar interfaces gráficas. Contudo, apesar da existência dessas ferramentas, a utilização do teste baseado em modelos para testar GUIs apresenta diversas dificuldades. Um dos principais problemas deste tipo de teste reside na dificuldade de construção dos modelos que, por vezes, é muito dispendiosa em tempo e esforço. Como descrito em [Mem07], alguns dos modelos possuem

uma limitada extensibilidade, sendo incapazes de responder a todos os problemas do teste de GUIs. Outro dos seus problemas é a necessidade da utilização de notações formais para definir o oráculo de testes. Algumas das ferramentas exigem a aprendizagem de novas notações de modelação o que recai de novo num dos problemas do teste de GUIs. Por outro lado, existe um grande esforço de configuração do processo de geração dos casos de teste. Apesar da geração dos testes ser efectuada de forma automática existe ainda um grande processo manual para definição do modelo e configuração da geração de testes. Para a geração dos casos de teste surge a necessidade de simular as acções da GUI para que a implementação possa ser testada sem a intervenção do utilizador e também a necessidade de mapear essas acções para acções abstractas para execução dos testes. Desta forma, os modelos devem ter um nível de abstracção adequado de forma a captarem acções do utilizador sobre a GUI e os seus efeitos. Estas ferramentas têm ainda que lidar com a "explosão" de estados que é uma das dificuldades do teste de GUIs. [Mem07, PFV07, SCP08]

1.3 Identificação do Problema

Como foi referido anteriormente, o teste de interfaces gráficas com o utilizador (GUIs) é muito importante para assegurar o correcto funcionamento do software. Contudo, apesar da sua importância, esta tarefa não é trivial. Uma das suas grandes dificuldades consiste na automatização deste tipo de testes. Existem várias ferramentas de teste de GUIs, no entanto, não automatizam a geração de casos de teste. Para automatizar a geração de casos de teste, é possível recorrer aos métodos de teste baseado em modelos formais. Contudo, as abordagens existentes apresentam ainda algumas limitações que impedem a sua aplicação em contextos industriais. O desenvolvimento dos modelos formais das GUIs requer um significativo esforço e tempo. Existe também um grande esforço requerido no processo de configuração de geração dos casos de teste para garantir a qualidade dos testes gerados. Outro dos problemas é que por vezes os profissionais envolvidos no desenvolvimento dos testes apresentam relutância em escrever especificações formais, preferindo utilizar notações em que se encontram mais habituados.

Devido a estas dificuldades, surge a necessidade de encontrar uma forma de reduzir o esforço de construção dos modelos. Uma possibilidade para tentar ultrapassar esta barreira consiste em aumentar o nível de abstracção dos modelos recorrendo, para isso, a modelos de tarefas. Na fase de desenho de sistemas interactivos, estes modelos são muito utilizados para representar as tarefas que um utilizador pode realizar no sistema. Contudo, estes modelos apenas contemplam situações previstas e correctas de comportamento dos utilizadores. Por exemplo, para realizar uma tarefa A, o utilizador tem que realizar uma sequência de acções, mas o que acontece se o utilizador realizar essa sequência de acções por outra ordem não é

especificado. Estes erros típicos de comportamento do utilizador não são contemplados na fase de desenho destes modelos, pelo que os testes realizados sobre os modelos não iriam testar essas situações.

1.4 Motivação e objectivos

O principal objectivo deste trabalho de investigação consiste no teste de interfaces gráficas com o utilizador (GUIs) baseado em modelos, utilizando como base modelos de tarefas que depois se adaptam para testar comportamentos não previstos do utilizador.

Para alcançar este objectivo pretende-se avaliar a viabilidade da utilização de modelos de tarefas no contexto de teste baseado em modelos. Pretende-se ainda analisar de que forma a modelação de tarefas pode ou não ser utilizada para modelar situações ou comportamentos de erro dos utilizadores e como fazer alterações ao modelo para testar as GUIs. Por fim, pretende-se desenvolver uma ferramenta capaz de explorar automaticamente um modelo de tarefas e, a partir dele, gerar casos de teste. No final, será avaliada a qualidade dos casos de teste gerados por essa ferramenta. Ou seja, pretende-se avaliar se o conjunto de testes gerados pela ferramenta é realmente eficaz para testar situações inesperadas de comportamento do utilizador que ocorrem tipicamente na utilização de interfaces gráficas.

1.5 Estrutura

Esta dissertação de mestrado encontra-se dividida em seis capítulos principais. O primeiro capítulo corresponde a esta introdução que apresenta o contexto, o problema e os objectivos desta dissertação.

No segundo capítulo é apresentado o levantamento do estado da arte relativo a notações de modelação de tarefas e ferramentas de teste de interfaces gráficas com o utilizador baseadas em modelos existentes. Neste capítulo é apresentada uma descrição das diversas notações e ferramentas analisadas.

O terceiro capítulo apresenta a abordagem utilizada na realização da dissertação. Neste capítulo são descritas todas as decisões efectuadas ao longo da dissertação e a metodologia seguida para pôr em prática a abordagem definida.

O quarto capítulo apresenta os detalhes mais tecnológicos relativos às partes de desenvolvimento identificadas no capítulo da abordagem.

No quinto capítulo são apresentados diversos casos de estudo utilizados para validação da abordagem definida.

Por fim, no sexto capítulo são apresentadas as conclusões obtidas no decorrer do desenvolvimento da dissertação, sendo apresentada uma análise do alcance dos objectivos propostos. Por fim são também apresentadas propostas de trabalho futuro.

Capítulo 2

Estado da Arte

Neste capítulo será apresentado o estado da arte relativo a modelação de interfaces gráficas a partir de modelos de tarefas, incluindo notações de modelação de tarefas, assim como ferramentas de teste de interfaces gráficas baseado em modelos.

2.1 Modelação de Tarefas

Durante o processo de desenvolvimento de sistemas interactivos, mais concretamente na fase de desenho, os modelos de tarefas são os modelos mais comuns utilizados pelos profissionais. Uma tarefa consiste numa actividade que é realizada a fim de atingir um determinado objectivo. A modelação de tarefas consiste na identificação das interacções do utilizador com o sistema interactivo, realçando os principais problemas que devem ser considerados no processo de desenvolvimento de interfaces. Os modelos de tarefas permitem uma representação abstracta da interacção do utilizador com o sistema. [MPS02, SCP08]

Existem várias notações e métodos de modelação de tarefas. De seguida, serão apresentados alguns.

2.1.1 CTT

A notação CTT (*ConcurTaskTrees*) consiste numa notação de modelação e de análise hierárquica de tarefas. O principal objectivo é ser uma notação fácil de utilizar que permita o suporte ao desenho de aplicações industriais, ou seja, de aplicações de médias e grandes dimensões. A estrutura da notação CTT consiste numa estrutura hierárquica de tarefas em árvore, o que a torna bastante intuitiva, permitindo a decomposição de tarefas em tarefas mais pequenas. Outra das suas características é que esta notação se foca nas actividades, que são o

aspecto mais importante no desenvolvimento de interfaces interactivas. A notação CTT possui uma sintaxe gráfica que facilita a interpretação e permite a definição de tarefas concorrentes, permitindo o estabelecimento de relações temporais entre tarefas. [KK05, MPS02, SCP08]

Com a notação CTT é possível representar diferentes tipos de tarefas, que se encontram apresentados na Figura 2.1.

Figura 2.1: Tipos de tarefas dos CTT

Como se pode observar pela análise da Figura 2.1, existem 4 tipos de tarefas:

- **Tarefas do utilizador:** tarefas realizadas exclusivamente pelo utilizador;
- **Tarefas de interacção:** representam os inputs que o utilizador insere na aplicação;
- **Tarefas da aplicação:** tarefas realizadas exclusivamente pelo sistema que representam os outputs que a aplicação transmite ao utilizador;
- **Tarefas abstractas:** tarefas que requerem acções complexas e que não recaem completamente em nenhuma das situações anteriores.

Com excepção das tarefas abstractas, todas as tarefas devem aparecer no modelo como folhas da árvore. As tarefas abstractas aparecem como nós internos e são utilizadas para estruturar o modelo. [PMM97, SCP08]

Como foi referido, a notação CTT permite também definir relações temporais entre as tarefas. Para definir essas relações existe um conjunto de operadores que as permitem representar. Na Tabela 2.1, é possível observar alguns dos operadores existentes. Em [MPS02], é apresentada uma descrição mais detalhada de cada um dos operadores assim como a sua ordem de prioridade.

Tabela 2.1: Operadores da notação CTT

Operador	Nome Original
[]	Choice
|=|	Order Independency
|||	Concurrent

|[]|	Concurrent with Info Exchange
[>	Disabling
|>	Suspend/Resume
>>	Enabling
[]>>	Enabling with Info Exchange
T	Task
T*	Iterative Task
[T]	Optional Task

Com os vários tipos de tarefas e com os operadores, é então possível construir um modelo de tarefas de um sistema interactivo representando a decomposição de tarefas e as suas relações. A Figura 2.2 apresenta um exemplo de um modelo de tarefas em CTT.

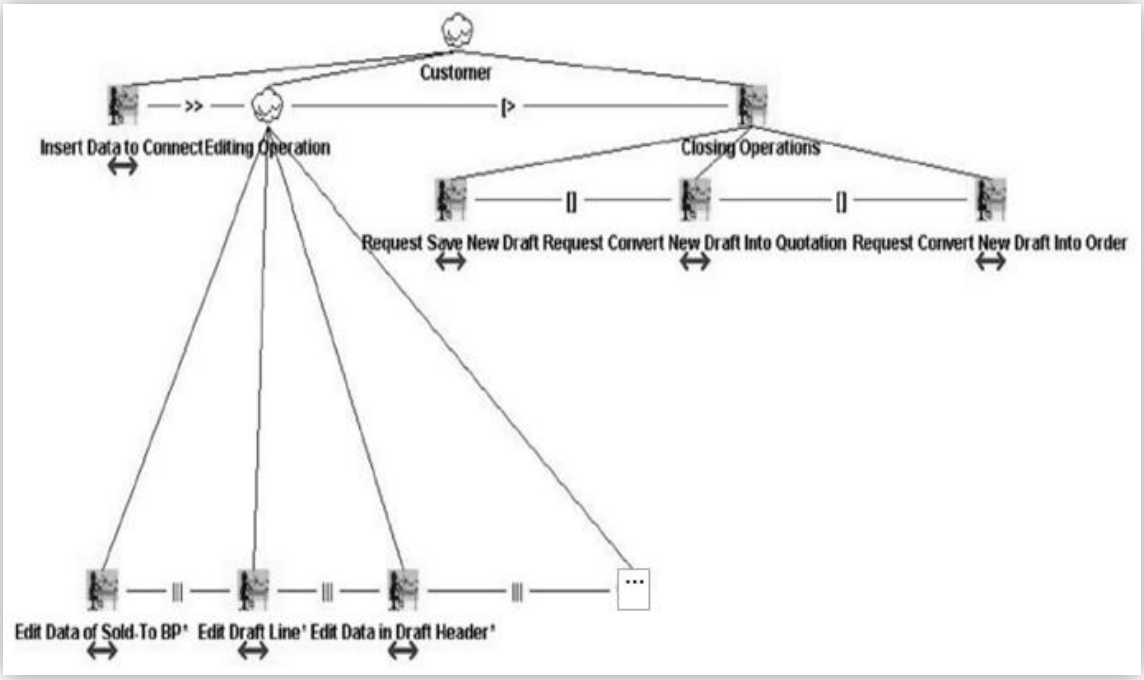


Figura 2.2: Exemplo de um modelo de tarefas em CTT [MPS02]

O exemplo da Figura 2.2 refere-se às funcionalidades de uma aplicação ERP a que um cliente tem acesso. Em primeiro lugar, o utilizador pode inserir os dados para conexão. Quando esta tarefa termina, ficam disponíveis as operações de edição, como indicado pelo operador "**>>**". As operações de edição podem ser realizadas ao mesmo tempo (operador "|||") e repetidamente (operador "*"). Durante a realização das operações de edição, estas podem, a qualquer momento, ser interrompidas pela realização das operações de fecho, conforme

indicado pelo operador "[>". Nesta altura o utilizador pode escolher a operação de fecho pretendida de entre as três existentes (operador "[]").

O processo de construção do modelo compreende três fases principais. Na primeira fase, são identificadas as tarefas e é realizada uma decomposição hierárquica das mesmas, representando-as numa estrutura em forma de árvore. A segunda fase consiste na identificação das relações temporais existentes entre tarefas do mesmo nível hierárquico, como foi referido, estas relações são representadas recorrendo a operadores. Por fim, a terceira fase corresponde à identificação dos objectos que serão manipulados e das suas acções. Cada objecto é associado a cada tarefa de forma apropriada. Este processo de identificação é realizado camada a camada. [PMM97]

Para ajudar em todo este processo de construção do modelo de tarefas, o grupo de Interacção Pessoa Computador - ISTI (Pisa) desenvolveu uma ferramenta de suporte denominada CTTE [PMM10] (*Concur Task Trees Environment*). Esta ferramenta consiste num editor de modelos de tarefas que permite a construção de modelos de tarefas para um único utilizador ou de modelos de tarefas cooperativos. Na Figura 2.3, é possível observar um exemplo de um modelo de tarefas cooperativo na plataforma CTTE versão 2.4.3.

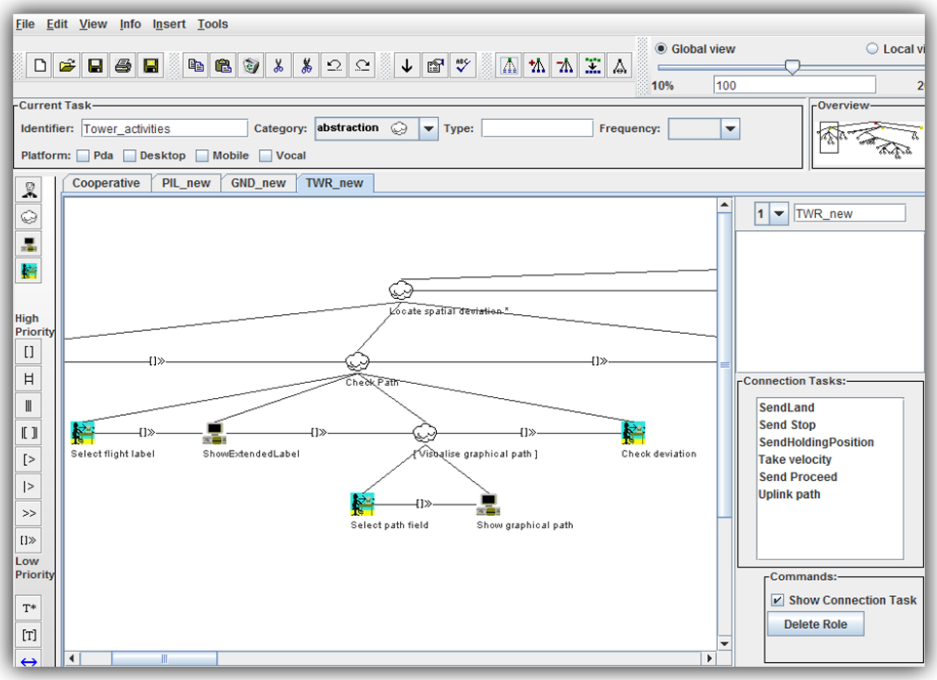


Figura 2.3: Exemplo de modelo de tarefas no CTTE

O CTTE permite ainda guardar os modelos de tarefas especificados ou partes deles em diversos formatos. É possível guardar as especificações como imagens no formato JPEG, ou então exportar para formato XML ou HTML. [MPS02]

Para além do CTTE, existe ainda outra ferramenta de suporte à edição de modelos CTT, denominada TERESA [MPS10]. Esta ferramenta suporta o desenho de interfaces multimodais, ou seja, interfaces que permitem diversas formas de interacção, como por exemplo, por teclado, rato e voz. Desta foram, a ferramenta TERESA [MPS10], fornece um ambiente semi-automático de suporte a diversas transformações úteis para os profissionais avaliarem o seu desenho a diferentes níveis de abstracção e gerar interfaces do utilizador para uma plataforma específica. O principal objectivo desta ferramenta é permitir, a partir de um único modelo, gerar diferentes interfaces para cada plataforma (por exemplo, para Web e para um telemóvel). Para esse processo, a ferramenta parte das tarefas definidas em CTTE e, após algumas transformações, gera automaticamente interfaces. [MPS04]

A ferramenta TERESA [MPS10] suporta a edição e análise de modelos CTT especificados em XML e, a partir da análise semântica dos operadores temporais permite a geração automática de PTSs (*Presentation Task Sets*) e das transições de um PTS para outro. Um PTS consiste num conjunto de tarefas que estão activas num determinado ponto de interacção, num mesmo período de tempo. A partir desses PTSs é que são depois geradas as interfaces. Esta ferramenta permite também a configuração de diversos atributos e propriedades por parte dos designers. [MPS04]

Na Figura 2.4 é possível observar a construção e edição de um modelo de tarefas na ferramenta TERESA.

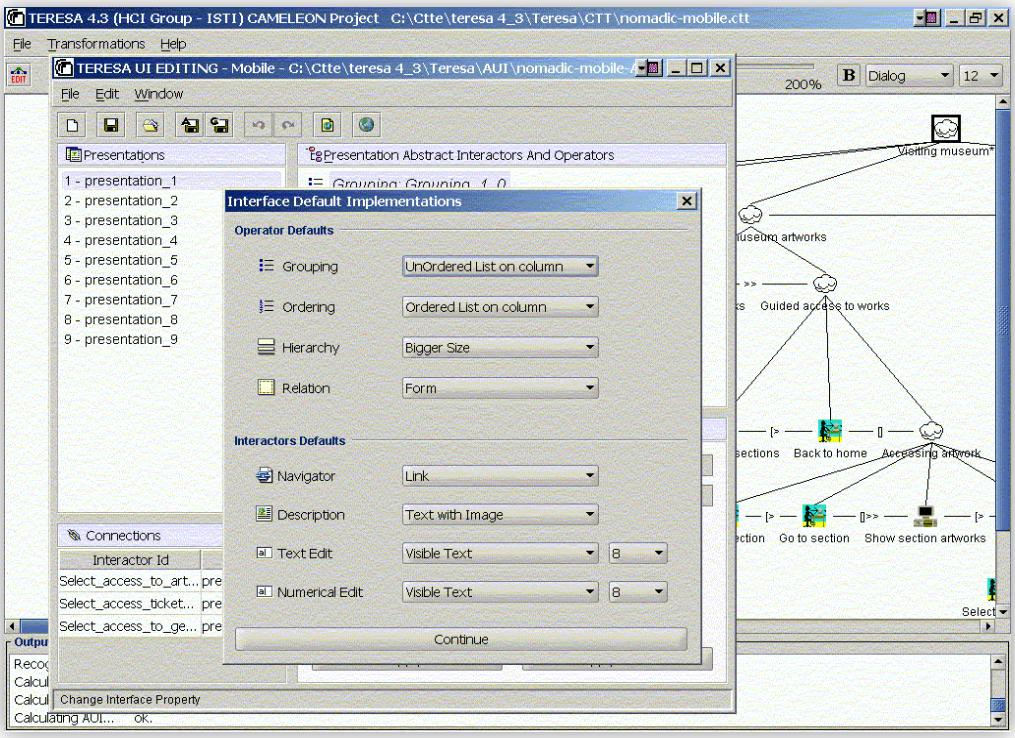


Figura 2.4: Interface da ferramenta TERESA [MPS10]

2.1.2 GTA

GTA (*Groupware Task Analysis*) consiste num método de análise de tarefas que tem como objectivo modelar ambientes complexos, combinando aspectos de diferentes métodos. Este método, em vez de se focar nas actividades de um único utilizador, foca-se no estudo das actividades de um grupo ou organização, em que muitos utilizadores interagem com os sistemas interactivos. GTA consiste assim numa ferramenta conceptual que permite especificar os aspectos mais importantes das tarefas relevantes quando se desenham sistemas multi-utilizador, focando-se nos aspectos de grupo. [BLV10, VWC02, WV03, WVE98]

Para desenhar este tipo de sistemas é necessário descrever, não só as tarefas, como também outros aspectos relevantes do mundo das tarefas. Deste modo, o GTA permite uma distinção entre o modelo de tarefas descritivo e o modelo de tarefas prescritivo. No início do processo de desenho de sistemas interactivos é muito útil compreender a situação actual. O GTA permite assim a descrição da situação actual das tarefas, o desenho de um modelo descritivo do conhecimento dos utilizadores, que terá importante relevância no desenvolvimento do novo sistema. Após a descrição desta situação actual, é necessário reestruturar o desenho da estrutura de tarefas incluindo aspectos tecnológicos. Consequentemente, o GTA salienta a necessidade de desenhar um modelo prescritivo do mundo de tarefas futuro, descrevendo como deverão ser as tarefas quando o sistema for desenvolvido e utilizado. Este modelo prescritivo deve conter as funcionalidades e artefactos que estão a ser desenvolvidos, assim como as suas relações com os utilizadores, objectivos e tarefas e contexto de utilização. [VWC02, WV03]

No modelo GTA, os modelos de tarefas são constituídos por agentes, trabalho e situação. Cada um destes aspectos apresenta uma diferente perspectiva do mundo de tarefas. Os agentes representam as pessoas e os sistemas. Os agentes que desempenham um conjunto comum de tarefas formam os papéis. Os papéis indicam classes de agentes aos quais são alocados conjuntos de tarefas. Quanto ao trabalho, o GTA considera os aspectos estruturais e dinâmicos do trabalho, tendo como base as tarefas. Desta forma, existem dois tipos de tarefas: as tarefas unitárias e as tarefas básicas. As unitárias correspondem às tarefas mais simples que um utilizador quer realizar. As básicas correspondem a tarefas para as quais o sistema fornece uma única função, decomposta em acções do utilizador e operações do sistema. No desempenho das suas tarefas, os utilizadores manipulam objectos. O desempenho de uma tarefa é desencadeado por eventos ou por outra tarefa. A descrição dos objectos do ambiente, incluindo a sua estrutura e atributos, consiste na análise do mundo de tarefas pela perspectiva da situação. [Tar04, WV03, WVE98]

O GTA permite que um objectivo possa ser alcançado de diversas maneiras sendo que cada tarefa tem um objectivo e os objectivos podem ser alcançados por várias tarefas. Contudo, é difícil estabelecer uma relação temporal entre as tarefas. O GTA apenas apresenta suporte

para uma descrição sequencial de tarefas, sendo que a descrição de relações temporais entre tarefas não pode ser precisamente especificada. Na Figura 2.5, podem-se observar duas formas de representar o fluxo de tarefas em GTA. [Tar04, WV03]

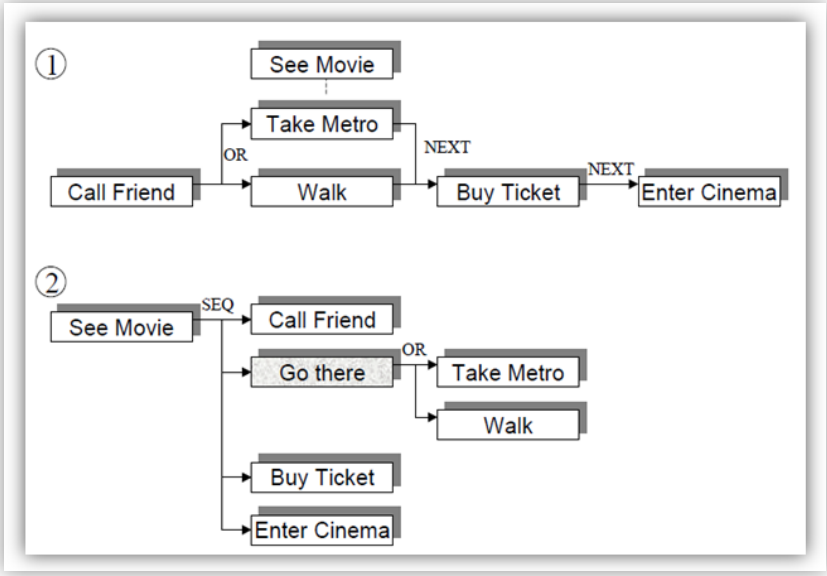


Figura 2.5: Exemplos do fluxo de tarefas em GTA [WV03]

Como se pode observar na Figura 2.5, a primeira representação apresenta a linha temporal num único eixo, no eixo das abcissas. A segunda representação apresenta uma decomposição de tarefas com recurso a construtores, existindo uma mistura entre tempo e decomposição de tarefas. Nesta segunda representação foi ainda necessário inserir uma nova tarefa "Go there", semelhante às tarefas abstractas do CTT. As duas representações apresentam a mesma descrição do mesmo fluxo de tarefas e em ambas se notam dificuldades em estabelecer relações temporais. [WV03]

Para suportar este método de análise de tarefas, o GTA, foi desenvolvido o Euterpe [BLV10] que oferece um conjunto básico de primitivas de análise. O Euterpe consiste numa ferramenta que ajuda na construção de árvores de tarefas de forma gráfica, hierarquias de objectos, papéis e desencadeamento de tarefas e eventos para cada tarefa. Esta ferramenta também permite a geração do fluxo de trabalho, baseando-se no desencadeamento de tarefas e eventos que cada tarefa proporciona. Contudo, as relações temporais entre as tarefas são ambíguas devido à limitação do GTA, uma vez que este não permite a representação formal das relações temporais entre as tarefas. Esta ferramenta permite também a geração de documentação como páginas HTML. A Figura 2.6 apresenta um exemplo de um modelo de tarefas na ferramenta Euterpe.

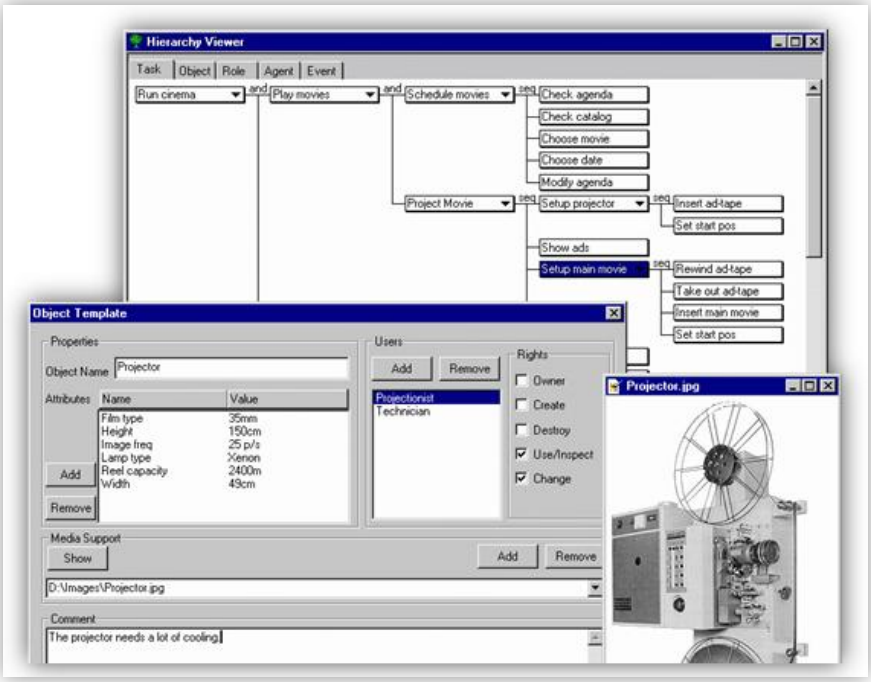


Figura 2.6: Exemplo de modelo de tarefas em Euterpe [BLV10]

2.1.3 TKS

TKS (*Task Knowledge Structure*) consiste numa abordagem teórica de analisar e modelar tarefas, baseando-se no conhecimento. Como o nome indica, o seu objectivo é modelar estruturas de tarefas com base no conhecimento que utilizadores têm sobre o domínio de tarefas do sistema interactivo. Estas estruturas do conhecimento vão sendo desenvolvidas à medida que os utilizadores aprendem e realizam tarefas. [Wil10]

Esta abordagem permite a distinção entre diferentes tipos de conhecimento, por exemplo: tarefas, objectos, papéis, objectivos, acções centrais (acções importantes para o alcance do objectivo) e pré-condições de acções. [Ham96, Wil10]

Em TKS, uma tarefa consiste numa actividade realizada por um agente a fim de atingir um determinado objectivo, uma mudança de estado. Um papel consiste num conjunto de tarefas. Cada pessoa pode assumir diversos papéis. Outra suposição é que as pessoas possuem estruturas de conhecimento bem organizadas que se relacionam com as tarefas. Desta forma, cada tarefa de um papel tem uma TKS associada. [Ham96, JJ91]

Uma TKS é dividida em duas partes principais: uma "estrutura objectivo" e uma "estrutura taxonómica". A "estrutura objectivo" consiste na representação da sequência de actividades de tarefas, identificando os objectivos da TKS. Esta estrutura representa um plano que define a ordem pela qual as tarefas têm que ser realizadas para atingir um objectivo particular. A "estrutura taxonómica" consiste na representação do conhecimento sobre os objectos e acções, as suas relações e comportamentos. [Ham96, JJ91]

2.1.4 GOMS

GOMS (Goals, Operators, Methods, Selection Rules) consiste numa técnica de modelação, que permite modelar o comportamento do utilizador em termos de objectivos, operadores, métodos e regras de selecção. [Hoc02]

Em [CNM83] é apresentada uma definição para cada um dos conceitos do modelo GOMS:

- **Objectivos:** Um objectivo é definido como sendo uma estrutura simbólica que define um conjunto de assuntos a resolver e determina um conjunto de métodos possíveis para a sua resolução;
- **Operadores:** Os operadores são actos elementares e cognitivos, que são executados para mudar um aspecto mental do utilizador ou afectar o ambiente das tarefas. Um operador é definido por um efeito (output) específico e duração;
- **Métodos:** Os métodos são os procedimentos que descrevem como se pode alcançar um objectivo;
- **Regras de Selecçã**o: Quando um objectivo pode ser alcançado por vários métodos, as regras de selecção permitem seleccionar qual o método a utilizar para alcançar esse objectivo.

Com base nestes conceitos é possível descrever o modelo GOMS como sendo uma série de métodos que são utilizados pelos utilizadores para atingir objectivos. Um utilizador pode executar vários métodos para atingir os seus objectivos. Um método é uma lista sequencial de operadores. Se existir mais de um método possível para atingir um objectivo é utilizada uma regra de selecção para escolher qual o método a utilizar, dependendo do contexto. [Hoc02]

A partir do modelo original GOMS, foram desenvolvidas diversas variantes: KLM, CMN-GOMS, NGOMSL, CPM-GOMS. Estas variantes foram desenvolvidas com o objectivo de superar alguns problemas do modelo de GOMS original. Em [Hoc02] pode ser encontrada uma descrição destas variantes. Existem também algumas ferramentas que ajudam na criação de modelos GOMS, como é o caso da GOMSED [Rud10] e da CogTool [Uni09] que se baseia na abordagem KLM.

Apesar das características deste modelo, este apresenta algumas limitações. O modelo de GOMS fornece uma completa descrição dinâmica do comportamento. A partir de uma tarefa, este modelo permite instanciar a sua descrição como uma sequência de operadores, associando tempos a cada operador. Se estes tempos forem dados como distribuições, o modelo consegue fazer previsões estatísticas. No entanto, essas previsões apenas são válidas se não ocorrerem erros. Esta é portanto uma das suas limitações, que se encontra descrita em [CNM83]. Outra das suas limitações é que este modelo é limitado a descrever o comportamento do utilizador, não considerando outros aspectos das interfaces. Alguns autores, como é o caso de [MPS02, PMM97, Tar04], referem que o modelo GOMS permite uma decomposição hierárquica de

tarefas e objectivos, usando uma linguagem textual, no entanto, a maioria das suas abordagens apenas permite uma análise sequencial das suas tarefas.

2.1.5 UAN

A UAN (*User Action Notation*) consiste numa notação orientada às tarefas e ao utilizador para especificar o comportamento assíncrono de sistemas interactivos e descrever acções do utilizador. Esta notação permite descrever o comportamento cooperativo entre o utilizador e a interface enquanto estes realizam uma tarefa em conjunto, sendo as tarefas do utilizador o seu principal elemento de análise. [HSH90]

Em UAN, uma interface consiste numa estrutura hierárquica de tarefas desordenadas e assíncronas. As acções do utilizador, o correspondente *feedback* da interface, a informação relativa aos estados e possíveis transições são especificadas a um nível de abstracção mais baixo. Desta forma estes elementos podem ser ocultados por níveis de abstracção mais elevados, os níveis semânticos, onde se constrói a estrutura de tarefas da interface. Em qualquer nível de abstracção, é possível estabelecer relações temporais de sequência, intercalação e concorrência entre acções e tarefas do utilizador. [HSH90]

Os objectos e acções em UAN são representados pelos seguintes símbolos:

- **M:** botão do rato;
- **v:** baixo;
- **^:** cima;
- **[ ]:** contexto de um objecto;
- **~:** mover o cursor;
- **K:** input de caracteres (teclado);
- **K"copy":** cópia.

Existe ainda um conjunto de símbolos que permite estabelecer as relações temporais entre tarefas e acções. Em [HSH90] é possível encontrar uma descrição mais detalhada dos vários símbolos da notação UAN. Ainda em [HSH90] é possível encontrar um simples exemplo da especificação de tarefas em UAN. Esse exemplo consiste na tarefa básica de seleccionar um ficheiro que requer a execução de duas acções por parte do utilizador:

1 Mover o cursor para o ícone do ficheiro;

2 Clicar no botão do rato.

Estas acções podem ser representadas em UAN da seguinte forma:

1 ~[ícone]

2 Mv^

Na acção 1, é indicada a movimentação do rato (~) para o contexto do ícone. De seguida na acção 2, o clique do rato é representado com a compressão do botão do rato (v) e a sua

libertação (^). Para tarefas e acções mais complexas e extensas, esta notação pode ser bastante difícil de se analisar.

Cada tarefa que o utilizador pode realizar é descrita numa tabela que contém os campos ilustrados na Tabela 2.2. A leitura desta tabela de descrição de tarefas deve ser realizada da esquerda para a direita e de cima para baixo. [HSH90]

Tabela 2.2: Tabela de descrição de tarefas em UAN

Tarefa: <nome da tarefa>		
Acções do Utilizador	*Feedback* da interface	Estado da interfaces

Uma das principais limitações desta notação é a inexistência de ferramentas de suporte à edição e análise das descrições de tarefas. No entanto, existem alguns esforços para resolver esta limitação. Em [SDMS98] é apresentada uma ferramenta que se encontra em desenvolvimento, que se baseia nas tabelas de descrição de tarefas da notação UAN. As principais características desta ferramenta consistem em suportar a definição de tarefas recorrendo aos símbolos da linguagem UAN, a fácil edição das tabelas de descrição e a verificação sintáctica das especificações das definições de tarefas.

2.1.6 Análise Comparativa

Cada uma das notações de modelação de tarefas apresentada segue sua própria abordagem e possuí um conjunto diversificado de características baseando-se em certos aspectos das interfaces gráficas.

A notação CTT permite uma descrição hierárquica de tarefas sob a forma de árvore, possuindo um conjunto de operadores para definir relações temporais entre as tarefas. Esta notação permite a especificação de diferentes tipos de tarefas, tarefas do utilizador, do sistema e de interacção. A notação UAN permite descrever uma tarefa através do preenchimento de uma tabela de descrição, que apresenta a descrição do comportamento cooperativo entre o utilizador e a interface. Esta notação também possui um conjunto de símbolos para representar relações temporais entre as tarefas. Contudo, para especificações mais longas e complexas, a notação textual UAN pode ser mais difícil de se analisar do que a notação CTT, que possui uma sintaxe gráfica. A notação CTT possui ferramentas de suporte à análise e construção dos modelos, CTTE e Teresa, que permite a sua exportação para formato XML ou HTML. Ao contrário da CTT, a notação UAN não possui ferramentas de suporte que permitam a exportação das tabelas de descrição para outros formatos.

A notação GOMS é bastante eficaz na descrição do comportamento do utilizador, permitindo ainda fazer previsões estatísticas com base em tempos associados aos operadores, caso não haja erros. Contudo, esta notação apresenta uma limitação em relação às notações CTT e UAN que é o facto de apenas permitir representar o comportamento do utilizador, deixando outros aspectos importantes das interfaces de parte. Outra das suas desvantagens é que a maioria das suas abordagens apenas permite uma análise de tarefas sequencial. Contudo, algumas das suas abordagens permitem a descrição hierárquica e possuem ferramentas de suporte.

Enquanto a notação GOMS se baseia no comportamento do utilizador, a notação TKS baseia-se no conhecimento que o utilizador tem sobre o sistema interactivo, descrevendo as tarefas como estruturas de conhecimento.

Em UAN não é possível descrever tarefas de sistemas multi-utilizadores, como é o caso da notação GTA, por exemplo. Esta notação, o GTA, procura descrever as tarefas de acordo com as actividades dos vários utilizadores de um grupo ou organização, permitindo ainda o alcance dos objectivos de diversas maneiras. Contudo, esta notação não permite uma descrição específica das relações temporais entre tarefas, sendo esta ambígua, coisa que é ultrapassada pelas notações CTT e UAN. No entanto, a ferramenta Euterpe [BLV10] oferece suporte à construção de modelos em GTA e permite a geração de fluxo de trabalho baseando-se no desencadeamento de tarefas e eventos que cada tarefa proporciona. A notação CTT também possui uma ferramenta de suporte bastante eficaz, a TERESA [MPS10], que suporta a edição e análise de modelos CTT e permite a geração automática de PTSs (conjunto de tarefas activas num determinado ponto de interacção) e transições entre eles.

2.2 Ferramentas de Teste de Interfaces Gráficas Baseado em Modelos

Para auxiliar o teste de interfaces gráficas com o utilizador (GUIs) existem diversas ferramentas de teste. Algumas dessas ferramentas baseiam-se em teste baseado em modelos e permitem automatizar alguns aspectos dos testes de GUIs. Para construção do modelo, estas ferramentas recorrem a linguagens formais devido à necessidade de utilizar semânticas formais para construir modelos precisos. De seguida serão apresentadas algumas abordagens e ferramentas de teste de interfaces gráficas baseado em modelos.

2.2.1 Spec Explorer

O Spec Explorer [VCGSTN08] é uma ferramenta de teste baseado em modelos da Microsoft Research. Para construção dos modelos, esta ferramenta suporta duas linguagens de modelação: Spec# [BLS05] e Asml [GRS03].

O Spec# [BLS05] consiste numa linguagem de programação que é uma extensão à linguagem de programação C#, orientada a objectos. O Spec# acrescenta a essa linguagem tipos não nulos, funções de controlo de excepções e também pré-condições e pós-condições.

A linguagem Asml [GRS03] consiste numa linguagem abstracta que permite a especificação de máquinas de estados. Esta linguagem de especificação é executável e pode ser utilizada quer na fase de desenho, quer na fase de implementação, quer na fase de teste do processo de desenvolvimento de software. Para as suas especificações, esta linguagem usa XML e Word.

O Spec Explorer [VCGSTN08] automatiza a geração e execução de casos de teste a partir do modelo especificado. Para gerar os casos de teste é necessário identificar os métodos do modelo que permitem transições entre o sistema, esses métodos são denotados como acções. Essas acções estão activas em determinados estados do sistema, sendo estas especificações realizadas recorrendo a pré-condições. Com as acções e pré-condições especificadas, o Spec Explorer cria, a partir da exploração do modelo, uma máquina de estados finita (FSM). A partir dessa máquina de estados são gerados vários casos de teste que correspondem a sequências de acções identificadas na mesma. Esses casos de teste são posteriormente executados sobre o modelo e a implementação para detectar erros. Esta ferramenta permite realizar testes "on-the-fly", à medida que vão sendo criados, possuindo um algoritmo que combina as técnicas de geração e execução de casos de teste.

Esta ferramenta está adaptada para o teste de APIs (*Application Program Interfaces*) mas exige muito esforço para ser aplicada ao teste de GUIs. Face a estas dificuldades foram desenvolvidas algumas extensões a esta ferramenta no sentido de permitirem a sua adaptação ao teste de GUIs.

Em [PVST05] é apresentada a ferramenta GUI Mapping Tool. Esta ferramenta procura automatizar o processo de mapeamento entre os elementos da GUI e a implementação, auxiliando o utilizador a mapear as acções lógicas descritas no modelo para acções físicas sobre os objectos da GUI que se pretende testar, gerando automaticamente o código adaptador.

Existe uma outra extensão ao Spec Explorer que procura evitar o problema da explosão de estados tendo em conta a estrutura hierárquica das GUIs [PVFT]. Existe também uma abordagem que procura abstrair o nível de construção dos modelos em Spec#, através da utilização da linguagem UML para modelar a GUI [PFV07]. Esta abordagem define um conjunto de regras para traduzir o protocolo UML das máquinas de estados em pré e pós-condições da linguagem Spec#.

Existe ainda outra abordagem que utiliza a notação de modelação de tarefas CTT como base para a construção dos modelos em Spec# [SCP08]. Esta abordagem recorre à ferramenta Teresa [MPS10], de suporte ao CTT, que, a partir da modelação de tarefas em CTT, permite a geração automática de máquinas de estados finitas. A partir da máquina de estados gerada é

gerado automaticamente o modelo em Spec#, através da ferramenta TOM, proposta pelos autores.

2.2.2 GUITAR

O GUITAR [MBN03] (*GUI Testing frAmewoRk*) é uma ferramenta que permite o teste de interfaces gráficas com o utilizador (GUIs). Esta ferramenta possui um conjunto de módulos, componentes e ferramentas que permitem a automatização dos testes.

A ferramenta GUITAR baseia-se num processo dinâmico de engenharia reversa. Este processo consiste em automatizar a geração do modelo de testes a partir de GUIs já existentes. Esta automatização permite assim reduzir tempo e esforço necessários para a construção do modelo para a utilização de teste baseado em modelos. O modelo construído consiste num gráfico de fluxo de eventos para modelar o comportamento dos objectos e uma árvore de integração para modelar o comportamento entre objectos. Todos os modelos construídos podem ser visualizados graficamente, facilitando a sua análise. Contudo, não existe a possibilidade de edição dos modelos, nem de construção manual, sendo este processo totalmente automático e assegurado pela ferramenta. [MBN03]

2.2.3 IDATG

IDATG [BMS98] (*Integrating Design and Automated Test case Generation*) é uma ferramenta que permite a geração automática de casos de teste para interfaces gráficas com o utilizador (GUIs), a partir da sua especificação formal. Esta ferramenta oferece métodos de teste baseados em fluxo de trabalho e orientados a dados.

O IDATG fornece um editor que facilita a construção do modelo de especificação da GUI. Essa especificação é realizada através de especificações de acções do utilizador e cenários de teste, orientados às tarefas. Para gerar os casos de teste, esta ferramenta recorre ao método de classes equivalentes. O mapeamento dos testes para elementos da GUI (por exemplo, simular um clique do rato) é feito com o GUI Spy, que também permite capturar as janelas directamente do ecrã. A execução dos testes não é suportada por esta ferramenta, contudo, o IDATG permite a exportação dos casos de teste para XML ou outros formatos, a fim de serem executados por outras aplicações (por exemplo, o WinRunner). [BM09, BMS98]

2.2.4 PATHS

A ferramenta PATHS [MPS01] (*Planning Assisted Tester for grapHical user interface Systems*) consiste num sistema de geração automática de casos de teste baseado nas estruturas hierárquicas para modelar interfaces gráficas com o utilizador (GUIs).

Esta abordagem baseia-se em técnicas usadas em inteligência artificial. O seu principal conceito é que o *designer* apenas precisa de conhecer e especificar os objectivos que um utilizador pode realizar sobre uma GUI. A sequência dos vários eventos que o utilizador tem que realizar para alcançar os objectivos é automaticamente gerada pela PATHS. Esses eventos, também denominados de planos, serão utilizados como casos de teste da GUI. Para gerar esses planos, a ferramenta faz uma análise hierárquica da estrutura da GUI e cria operadores hierárquicos que irão ser usados no processo de geração de planos. O designer de testes tem, no entanto, que descrever as pré-condições e efeitos desses operadores. Para gerar os casos de teste é necessário fornecer ao sistema de planeamento um conjunto de estados inicias e objectivos, só depois é que são gerados múltiplos planos hierárquicos. As técnicas abordadas por PATHS têm como objectivo reduzir a complexidade e tamanho do modelo, assim como aumentar a eficiência da geração de casos de teste. [MPS01]

2.2.5 Outras Abordagens

Alguns autores seguem outra abordagem para o teste de interfaces gráficas.

Em [Bel01], Belli propõe a utilização de máquinas de estados finitas (FSMs) e expressões regulares para modelar as interfaces gráficas com o utilizador (GUIs). Essas máquinas de estados devem modelar não só o comportamento esperado e correcto, como também o comportamento errado. Assim sendo, o modelo original deve ser expandido para que contenha todas as interacções ilegais com o sistema. Os casos de teste gerados por essa expansão do modelo podem levar o sistema a estados correctos ou a estados de falha.

Shehady e Siewiorek, em [SS97], propõe a utilização de máquinas de estados finitas variáveis (VFSMs) para especificar o modelo formal da GUI. VFSMs consistem extensões das FSMs com condições associadas a cada transição. Para gerar casos de teste, o modelo VFSM é convertido numa FSM equivalente, a partir da qual são gerados os testes.

White e Almezen, em [WA00], propõe uma metodologia de modelação de GUIs que se foca nas sequências de objectos da GUI e selecções. Estas sequências de objectos e selecções são denominadas CIS (*Complete Interaction Sequences*), e podem ser utilizadas para o utilizador completar uma tarefa. Para gerar casos de teste, cada CIS é convertida para uma FSM.

2.3 Conclusões

Existem várias notações de modelação de tarefas de interfaces gráficas, cada uma com os seus objectivos e características. A notação CTT foca-se na descrição hierárquica de tarefas e relacionamentos temporais. No caso da notação GTA, esta foca-se na descrição de tarefas

realizadas em grupo. A abordagem TKS procura descrever as tarefas baseando-se em estruturas de conhecimento que o utilizador possui sobre as mesmas. A notação GOMS permite modelar tarefas com base em objectivos, operadores, métodos e regras de selecção, focando-se apenas no comportamento do utilizador. A notação UAN permite a descrição do comportamento cooperativo entre o utilizador e a interface, enquanto estes realizam uma tarefa em conjunto, baseando-se em tabelas de descrição. Algumas destas notações possuem ferramentas de suporte que facilitam a construção dos modelos e sua exportação.

Como foi referido, existem algumas ferramentas de teste de GUIs baseado em modelos que permitem automatizar alguns aspectos do teste de GUIs. O Spec Explorer [VCGSTN08] automatiza a geração e execução de casos de teste a partir de modelos especificados em duas linguagens específicas. Esta ferramenta não é indicada para teste de GUIs, no entanto existem algumas abordagens e expansões que o permitem, como a GUI Mapping Tool [PVST05]. A ferramenta GUITAR [MBN03] permite automatizar a geração do modelo a partir da GUI. A ferramenta IDATG [BMS98] fornece um editor que facilita a construção do modelo e permite exportação dos casos de teste gerados para XML ou outros formatos mas não a sua execução. A ferramenta PATHS permite a construção automática de caminhos (sequências de acções) necessários para atingir um determinado objectivo. Existem ainda algumas abordagens baseadas em FSMs (*Finite State Machines*) [Bel01], VFSMs (*Variable Finite State Machines*) [SS97] e CIS (*Complete Interaction Sequences*) [WA00].

Capítulo 3

Abordagem

Durante a fase de desenho de software, os modelos de tarefas são bastante utilizados para especificar as acções que um utilizador pode realizar sobre uma interface gráfica com o utilizador (GUI). Existe uma abordagem que utiliza estes modelos como base do teste baseado em modelos para o teste das GUIs [CSP09, SCP08]. Contudo, os modelos utilizados nesta abordagem apenas modelam situações previstas de comportamento, não permitindo assim testar a interface face a erros típicos do utilizador. Nesta dissertação de mestrado foi definida uma abordagem que pretende contribuir para a introdução deste aspecto.

Para definir uma estratégia de introdução de erros nos modelos de tarefas, em primeiro lugar, foi necessária a selecção de uma notação de modelação de tarefas das apresentadas na secção 2.1. Seguidamente foi necessário aprofundar o conhecimento da mesma e analisar os tipos de erros que um utilizador normalmente comete. Com esses conhecimentos foi então definida uma estratégia de introdução de erros. Uma vez que se pretende utilizar teste baseado em modelos para teste das GUIs, foi também necessária a selecção de uma ferramenta de teste baseado em modelos das apresentadas na secção 2.2. Após esta selecção foi necessário definir uma estratégia de alteração do modelo original, introduzindo erros típicos do utilizador, e de um algoritmo de realização dessas transformações.

Em termos gerais, a abordagem seguida consiste em partir do modelo de tarefas original, aplicar alterações ao modelo, introduzindo erros do utilizador, e, a partir desse modelo alterado, gerar o oráculo de testes a utilizar pela ferramenta de teste baseado em modelos. A Figura 3.1 apresenta, de uma forma geral, um esquema desta abordagem.

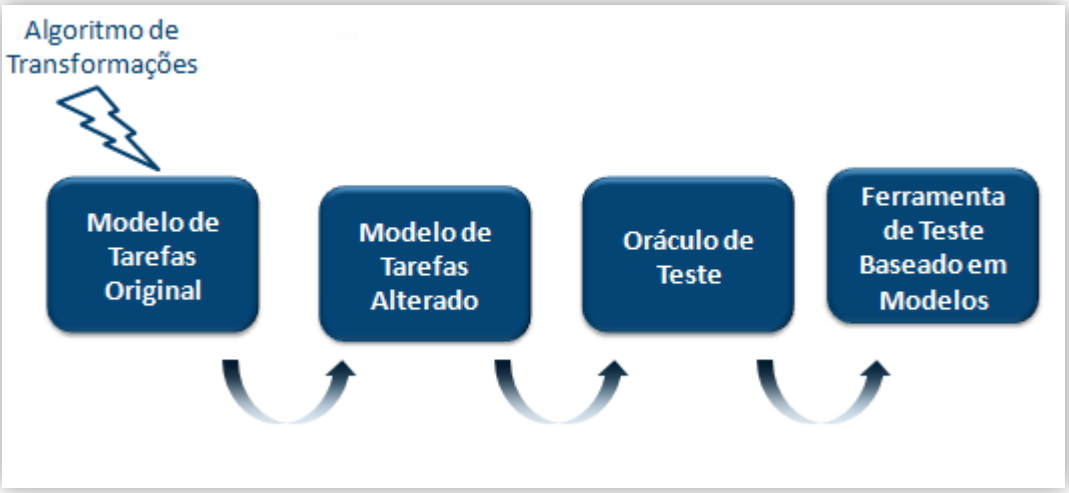


Figura 3.1: Abordagem geral

As secções seguintes apresentam uma descrição mais detalhada desta abordagem, incluindo a descrição da notação de modelação escolhida, dos erros típicos do utilizador, da metodologia adoptada e de estratégias de alteração do modelo de tarefas original.

3.1 Notação de modelação de tarefas

Para seleccionar a notação de modelação de tarefas a utilizar por esta abordagem foram analisadas cuidadosamente as diversas notações apresentadas no capítulo anterior (secção 2.1). Como critérios de escolha da notação foram considerados a facilidade de representação de modelos de tarefas de GUIs, assim como a existência de ferramentas de auxílio à construção dos modelos de tarefas.

Após esta análise, a notação de modelação de tarefas escolhida foi notação CTT (*ConcurTaskTrees*). Esta notação foi escolhida não só devido às suas características como também à existência de um conjunto de ferramentas de suporte bastante úteis.

A sua sintaxe gráfica facilita bastante a sua análise e interpretação, permitindo estruturar as tarefas de forma hierárquica, distinguir entre diferentes tipos de tarefas e o estabelecer relações temporais entre as mesmas recorrendo, para isso, a um conjunto de operadores. Estas características são factores importantes para especificação de modelos de tarefas de interfaces gráficas.

Outra das vantagens desta notação é a existência da ferramenta Teresa que presta suporte à construção dos modelos de tarefas. Esta ferramenta apresenta um conjunto de funcionalidades que permitem construir e manipular facilmente modelos de tarefas especificados em CTT. Uma das suas funcionalidades é a possibilidade de exportação desses modelos para XML, o que contribuiu para a elaboração desta abordagem. Outra vantagem da Teresa é que esta permite a

geração automática de máquinas de estados finitas que representam as transições entre conjuntos de tarefas que representam os estados (PTSs) e sua exportação para XML.

De seguida, apresenta-se uma descrição mais detalhada dos tipos de tarefas e operadores da notação CTT, assim como da ferramenta Teresa.

3.1.1 Tipos de tarefas da notação CTT

A notação CTT permite a distinção entre diferentes tipos de tarefas. Como já foi referido, existe a possibilidade de definir tarefas do utilizador (), tarefas de interacção (), tarefas de aplicação () e tarefas abstractas ().

As tarefas do utilizador permitem especificar tarefas que representam apenas pensamentos e intenções do utilizador, não contribuindo para alteração do estado da aplicação. Ao contrário do tipo de tarefas anterior, as tarefas de interacção permitem especificar acções que o utilizador realiza sobre a aplicação, que conduzem à alteração do estado da mesma. As tarefas de aplicação especificam tarefas que representam a resposta da aplicação face às acções realizadas pelo utilizador. Estas tarefas não alteram o estado da interface, apenas representam a sua alteração. Por fim, as tarefas abstractas permitem a especificação de tarefas mais complexas que são compostas por diversas sub-tarefas. Estas tarefas aparecem como nós internos da árvore e nunca como folhas. Este tipo de tarefas permite assim a estruturação da árvore de tarefas, facilitando a sua leitura.

3.1.2 Operadores da notação CTT

Como já foi referido, os operadores da notação CTT, possibilitam a especificação de relações temporais entre tarefas. Para compreender melhor a hierarquia de tarefas da notação CTT, apresenta-se, na Tabela 3.1, uma descrição mais detalhada dos operadores desta notação.

Tabela 3.1: Descrição dos operadores CTT

Exemplo	Designação	Descrição
T	Tarefa	Tarefa de realização obrigatória.
[T]	Tarefa opcional	Tarefa de realização não obrigatória.
T*	Tarefa iterativa	No final da tarefa T esta pode voltar a ser repetida as vezes que se pretender.

T1 [] T2	T1 ou T2 (*Choice*)	Só uma das tarefas é realizada, ou T1 ou T2.
T1 |=| T2	T1 e T2 por qualquer ordem (*Order Independency*)	As tarefas T1 e T2 podem ser realizadas por qualquer ordem.
T1 ||| T2	T1 ao mesmo tempo que T2 (*Concurrent*)	As tarefas T1 e T2 são realizadas concorrentemente.
T1 |[]| T2	T1 ao mesmo tempo que T2 e passa informação (*Concurrent with Info exchange*)	As tarefas T1 e T2 são realizadas concorrentemente, havendo passagem de informação de T1 para T2.
T1 |> T2	T2 desactiva T1 (*Disabling*)	As tarefas T1 e T2 encontram-se activas ao mesmo tempo, no entanto, T1 só se pode realizar até T2 ser realizada. A qualquer momento da tarefa T1 é possível realizar T2, deixando T1 de estar disponível.
T1 |> T2	T2 suspende T1 (*Suspend/Resume*)	A tarefa T2 interrompe T1, mas T1 continua do ponto onde estava quando T2 termina.
T1 >> T2	T1 activa T2 (*Enabling*)	T2 precede T1, ficando activa apenas quando T1 termina, as tarefas têm que se realizar pela ordem definida, T1 e só depois T2.
T1 []>>T2	T1 activa T2 e passa informação (*Enabling with Info exchange*)	T2 precede T1, ficando activa apenas quando T1 termina, havendo passagem de informação de T1 para T2, as tarefas têm que se realizar pela ordem definida, T1 e só depois T2.

3.1.3 Ferramenta Teresa

A ferramenta Teresa [MPS10], como já foi referido, permite a edição e construção de modelos de tarefas CTT, possuindo diversas funcionalidades de manipulação dos modelos. Em Teresa, a construção dos modelos é realizada recorrendo à sintaxe gráfica do CTT, permitindo a visualização dos modelos sob a forma de árvores hierárquicas de tarefas. Devido à complexidade de alguns modelos de maiores dimensões, existe a possibilidade de ocultar a informação relativa a sub-árvores do modelo. A Figura 3.2 apresenta um exemplo de um modelo de tarefas em Teresa.

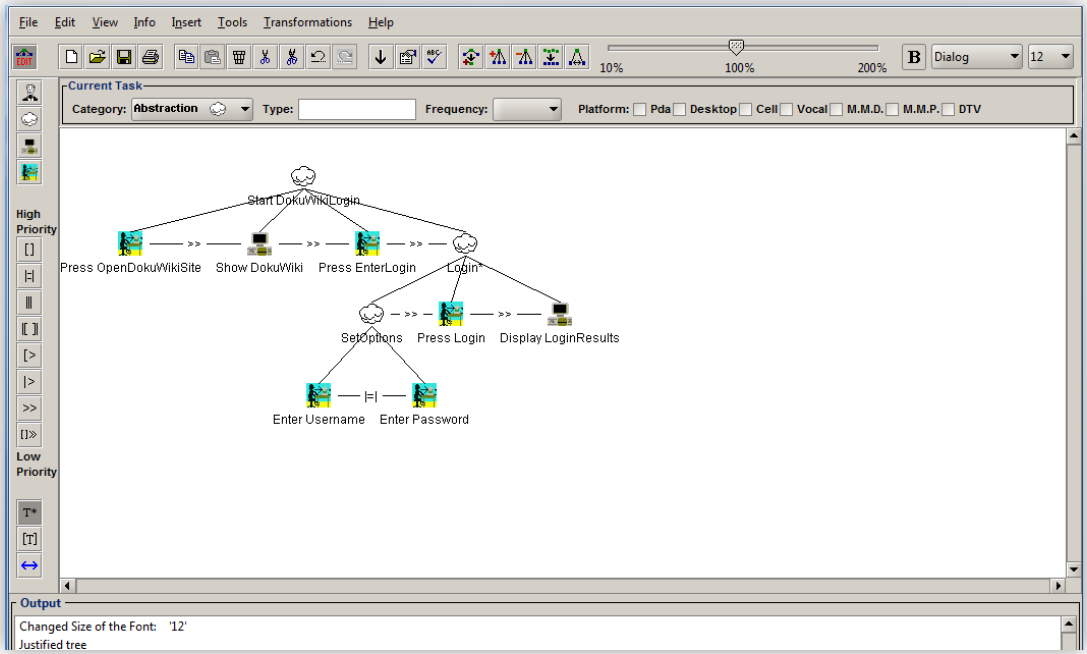


Figura 3.2: Exemplo de modelo de tarefas na ferramenta Teresa

Como se pode observar na figura anterior, o modelo de tarefas construído encontra-se na parte central da interface e o tipo de tarefas e operadores a utilizar encontra-se na lateral esquerda. A ferramenta Teresa permite ainda a especificação de determinadas características para cada tarefa. Estas características consistem na descrição, nome, identificador, pré-condição, objectos manipulados e tempos de execução, entre outras, como se pode observar na Figura 3.3.

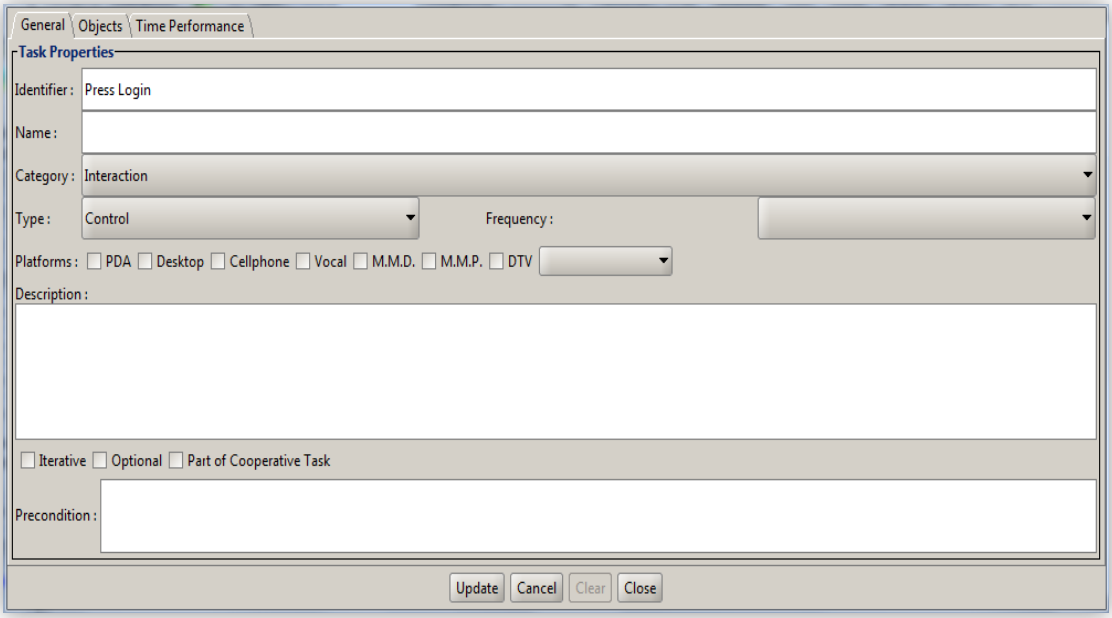


Figura 3.3: Características das tarefas na ferramenta Teresa

Os modelos criados podem ser guardados em ficheiros com formato CTT mas também existe a possibilidade da sua exportação para XML. Existe também a possibilidade de guardar os modelos para ficheiro na sua totalidade ou então guardar apenas sub-árvores do modelo original.

Uma vez que é permitida a exportação de modelos para XML, é também possível realizar a importação desses modelos. Existe ainda uma funcionalidade que permite a inclusão de uma sub-árvore, previamente exportada para ficheiro, no modelo de tarefas que se encontra em edição.

Após criação ou importação de um modelo de tarefas na ferramenta Teresa, existe a possibilidade de geração de um PTS, que consiste numa máquina de estados finita que indica as tarefas activas em cada estado assim como as transições entre as mesmas. Como as tarefas do utilizador não especificam nenhuma acção sobre a aplicação são excluídas do PTS.

A Figura 3.4 apresenta um exemplo de um PTS gerado na ferramenta Teresa.

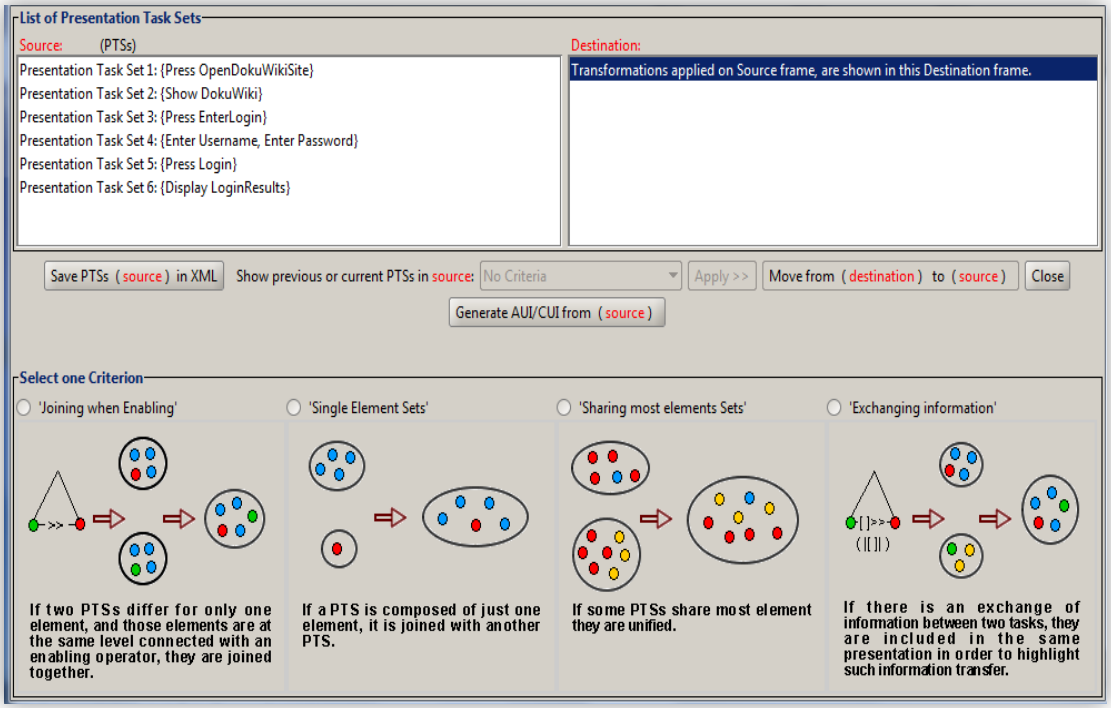


Figura 3.4: Exemplo de PTS na ferramenta Teresa

3.2 Erros típicos do utilizador

Segundo Reason, em [Rea90], o processo cognitivo de realização de tarefas de uma pessoa divide-se em três etapas. A primeira etapa consiste no planeamento, em que é identificado o objectivo da tarefa e a sequência de acções para atingir o objectivo, o plano. A segunda etapa consiste no armazenamento do plano na memória, até que este seja executado. A terceira fase consiste na execução do plano, realização das acções estipuladas.

Durante este processo cognitivo podem surgir erros, associados a cada uma das etapas. Em [Rea90] são identificados três tipos de erros do utilizador: deslizamentos (*slips*), lapsos (*lapses*) e enganos (*mistakes*).

Os erros do tipo deslizamentos correspondem à etapa de execução do processo cognitivo, e consistem na execução incorrecta de uma acção. Um exemplo típico deste erro consiste na execução de acções de uma sequência pela ordem errada, ou seja, o utilizador executa as acções contidas no plano pela ordem errada.

Os lapsos são erros que ocorrem na fase de armazenamento e consistem na incorrecta omissão de acção. Um exemplo típico deste erro consiste na omissão de uma acção planeada, ou seja, o utilizador esquece-se de realizar a acção.

Os erros do tipo enganos ocorrem na fase de planeamento e consistem na definição de um plano errado para alcance do objectivo, ou seja, o plano escolhido para alcançar o objectivo não é adequado. Um exemplo deste tipo de erros consiste na execução da tarefa errada para alcance do objectivo.

Em termos de tarefas, os dois primeiros tipos de erros podem ser representados no modelo de tarefas através da omissão, troca de operadores ou troca de ordem das tarefas do modelo ou por combinações destas acções.

O terceiro tipo de erros pode ser representando, recorrendo à elaboração de diferentes estratégias para alcance do objectivo. Cada estratégia corresponderia a um modelo de tarefas diferente, permitindo assim testar a aplicação com as diversas estratégias.

3.3 Metodologia

Para alcançar os objectivos propostos, seguindo a abordagem proposta anteriormente, foi definida a metodologia que se encontra representada no esquema da Figura 3.5.

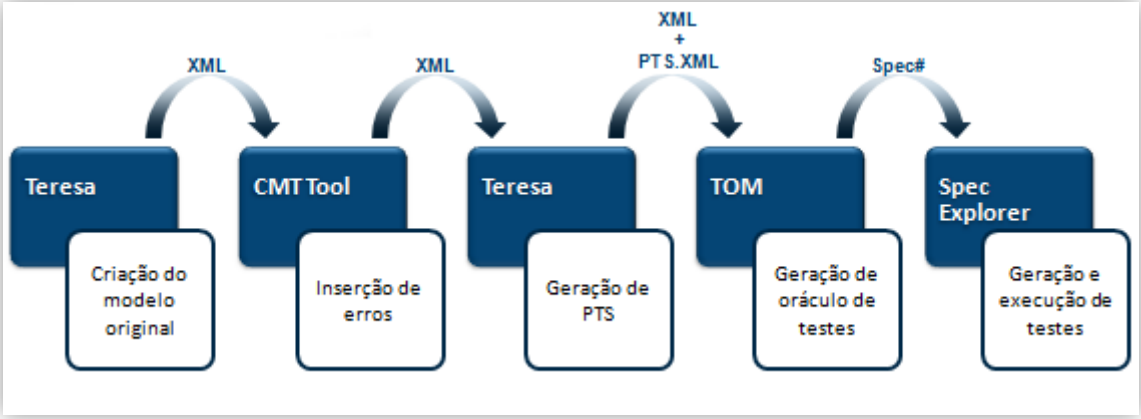


Figura 3.5: Metodologia

Como se pode observar na figura anterior, a metodologia definida compreende 5 etapas principais.

A primeira etapa corresponde à criação do modelo de tarefas original na notação CTT e exportação para ficheiro XML, que será realizada recorrendo às funcionalidades da ferramenta Teresa [MPS10].

Seguidamente, a segunda etapa consiste na introdução de erros no modelo original. Esta etapa será realizada pela ferramenta CMT Tool (*CTT Model Transformation Tool*) desenvolvida nesta dissertação de mestrado. A CMT Tool irá receber o modelo de tarefas original e realizar diversas transformações, construindo novos modelos XML. Os novos modelos contemplam permitem testar a GUI face a erros do tipo deslizamentos (*slips*) e lapsos (*lapses*). Para cada modelo transformado, será gerada a respectiva máquina de estados finita (PTS) e exportada para XML, recorrendo novamente à ferramenta Teresa [MPS10]. No caso dos erros do tipo enganos (*mistakes*), serão desenvolvidos diversos modelos de tarefas originais, cada um correspondendo a uma estratégia diferente de atingir o objectivo. Neste caso, não será necessária a realização de transformações pela CMT Tool, passando logo para a etapa de geração do PTS para cada modelo.

Na quarta etapa, para cada modelo transformado e respectivo PTS, será gerado o oráculo de testes na linguagem Spec#. Esta geração é automatizada, recorrendo à ferramenta TOM [CSP09, SCP08]. Por fim, com o Spec Explorer [VCGSTN08], serão gerados e executados os casos de teste a partir dos oráculos construídos anteriormente.

De seguida, apresenta-se uma descrição mais detalhada dos principais processos de cada etapa.

3.3.1 Especificação de modelos em CTT

Para ser possível a utilização do TOM para a geração do oráculo em Spec# a partir do modelo de tarefas foi necessário recorrer a algumas regras de construção de modelos definidas em [CSP09].

Para nomear as tarefas dos modelos, os autores de [CSP09] basearam-se nas palavras-chave de FIT (*Framework for Integrated Tests*) [MC05]. O principal objectivo do FIT consiste em ajudar no desenho e comunicação de casos de teste através de exemplos concretos. Para especificar os casos de teste foram desenvolvidas algumas notações com palavras-chave específicas. Como essas notações estão vocacionadas para teste de APIs (*Application Program Interfaces*), os autores de [CSP09] realizaram algumas alterações para que estas possam ser utilizadas no contexto de GUIs.

Após aplicação das alterações foi obtida a seguinte notação para especificar as tarefas do modelo de tarefas, sendo os campos entre parênteses rectos opcionais:

- **Start <tarefa>:** designa uma tarefa que representa um novo *namespace*. Um *namespace* consiste num espaço de nomes em que todos os nomes desse espaço são analisados no contexto do mesmo.
- **Enter <campo> <valor> [<tipo>]:** define tarefas relativas à entrada de dados do valor <valor> no campo <campo> com o tipo <tipo>. Se o tipo não for especificado é considerado o tipo *String*. Esta designação permite definir tarefas em que o utilizador insere dados numa caixa de texto, escolhe uma opção de uma lista, entre outras.
- **Press <botão> [<janela>]:** define tarefas que se referem a acções em que o utilizador pressiona o botão <botão> na janela <janela>. Se a janela não for especificada é assumida a janela corrente.
- **Show <janela>:** especifica tarefas em que a aplicação abre a janela <janela>, não modal.
- **ShowM <janela>:** especifica tarefas em que a aplicação abre a janela <janela>, modal. Uma janela modal consiste numa janela filho que requer que o utilizador a feche antes de retornar a interagir com a janela pai da mesma aplicação.
- **Display <valor> <janela>:** especifica tarefas em que a aplicação apresenta o valor <valor> na janela <janela>.
- **Close [<janela>]:** especifica tarefas em que a aplicação fecha a janela <janela>. Se a janela não for especificada é assumida a janela corrente.

A palavra-chave *Start* é tipicamente utilizada para definir a raiz de um modelo. As tarefas do tipo *Press* e *Enter* permitem especificar as tarefas de interacção em que o utilizador executa acções sobre a aplicação. As tarefas do tipo *Show*, *ShowM*, *Display* e *Close* permitem especificar tarefas de aplicação, ou seja, tarefas que não realizam qualquer acção sobre a interface.

Para além das palavras-chave referidas existe também a possibilidade de definir características de cada tarefa, proporcionada pela ferramenta Teresa. Assim, para cada tarefa, é possível especificar na sua descrição alguns detalhes da sua execução como, por exemplo, definir ou atribuir valores a variáveis, assim como definir pré-condições. Esta informação será interpretada pelo TOM e incluída no oráculo de testes em Spec# por este gerado.

3.3.2 Estratégia de alteração do modelo

Para alteração do modelo foram analisadas duas estratégias possíveis. A primeira consistia em, a partir do modelo de tarefas original, gerar automaticamente o modelo de um oráculo de teste baseado em modelos que permitisse a simulação de erros do utilizador. A segunda abordagem consistia em alterar directamente o modelo de tarefas introduzindo os erros e, a partir do modelo alterado, criar o modelo do oráculo de testes. Como a abordagem definida em

[CSP09, SCP08] apresenta uma ferramenta que permite a geração automática de oráculos de teste a partir de modelos de tarefas, optou-se por tirar partido das funcionalidades desta ferramenta. Assim, as introduções de erros seriam realizadas directamente sobre o modelo de tarefas original, e com a ferramenta referida em [CSP09, SCP08] seriam gerados oráculos para esses modelos transformados. Posteriormente esses oráculos seriam utilizados pela ferramenta de teste baseado em modelos.

Para alterar directamente o modelo de tarefas, foram analisadas diferentes abordagens. Estas abordagens consistiam em alterar os ficheiros com informação do modelo de tarefas gerados pela ferramenta Teresa. As hipóteses consideradas foram as seguintes:

- Alteração do modelo de tarefas original em CTT;
- Alteração do modelo de tarefas exportado para XML;
- Alteração do PTS exportado para XML.

No caso da primeira hipótese, o ficheiro do modelo de tarefas encontra-se codificado, não permitindo a sua leitura fora da ferramenta Teresa, não sendo assim possível realizar alterações a esse ficheiro. A terceira hipótese também foi descartada pois o PTS, por si só, não contém toda a informação de tarefas necessária, contendo apenas informação relativa às transições entre tarefas. Face à inviabilidade destas duas hipóteses, foi então adoptada a abordagem da segunda hipótese que consiste em:

- Criar modelo em CTT e exportar para XML;
- Realizar transformações ao modelo XML, gerando novos ficheiros XML;
- Criar ficheiros PTS para os novos ficheiros XML, possibilitando assim a sua utilização no TOM.

3.3.3 Algoritmo de transformações

Para a introdução de erros do tipo deslizamentos (*slips*) e lapsos (*lapses*) no modelo de tarefas foi necessária a definição de um algoritmo que permitisse, a partir do modelo original, gerar transformações ao mesmo. Este algoritmo de geração de transformações tinha que ser genérico de forma a funcionar para qualquer modelo de tarefas, independente da GUI a que este se referisse. Este passo de generalização foi o passo mais difícil na definição do algoritmo.

Numa primeira abordagem, foram analisados alguns casos concretos de interfaces de aplicações já existentes. Nesses casos, foram manualmente simulados erros do utilizador.

Existe um trabalho em desenvolvimento que procura detectar padrões em GUIs [Cun10]. A análise de alguns destes padrões permitiu identificar aplicações que possuem tarefas obrigatórias, sequências de acções, entre outras, que permitem a simulação dos erros pretendidos. Alguns destes padrões de interfaces gráficas encontram-se descritos na Tabela 3.2.

Tabela 3.2: Exemplos de padrões de Interfaces Gráficas com o Utilizador

Padrão	Descrição
Campo Obrigatório	Campo de um formulário que deve obrigatoriamente ser preenchido. Útil para testar deslizamentos.
Mestre/Detalhe	Um campo Mestre pode assumir vários valores. Consoante o valor escolhido para o Mestre, o Detalhe é actualizado. Útil para testar lapsos.
Formulário	Página com vários campos que devem ser preenchidos pelo utilizador.
Autenticação	Par de caixas de texto "Nome do Utilizador" e "Password" que permitem ao utilizador autenticar-se na aplicação/website.
Pesquisa	Pesquisa de um determinado conjunto de caracteres num texto.
Filtro	Perante um conjunto de linhas, o utilizador insere numa caixa de texto a que informação pretende aceder e as linhas que não estejam relacionadas com essa informação desaparecem.

Após análise das interfaces seleccionadas foi então possível detectar certos padrões na construção dos seus modelos que levam à geração de casos de teste semelhantes. Estes padrões estão relacionados com os vários operadores da notação CTT, assim como com a combinação dos mesmos. De seguida apresentam-se os padrões detectados e o respectivo algoritmo de transformação.

3.3.3.1. Sequências de tarefas de ordem obrigatória

Num modelo de tarefas CTT, uma sequência de tarefas com ordem obrigatória consiste num conjunto de tarefas separadas pelo operador >> ou []>>. Uma sequência deste tipo indica que as tarefas nela incluídas têm que ser realizadas obrigatoriamente pela ordem com que estão definidas. Nestas situações interessa testar se a ordem pela qual são realizadas as tarefas é, de facto, relevante, note-se que a troca de ordem é um dos erros típicos do utilizador.

Numa primeira abordagem apenas foi considerada a troca de ordem entre duas tarefas separadas pelo operador >> ou []>>. Contudo, esta abordagem revelou-se insuficiente pois, para sequências de maiores dimensões, não seriam testadas todas as possibilidades de ordem possíveis. Desta forma, foi adoptada uma segunda abordagem que permite realizar alterações ao modelo de tarefas testando as diversas possibilidades de ordem de tarefas.

Assim, se existir no modelo original a sequência T1>>T2, será necessário testar T1>>T2 e também T2>>T1. No caso de sequências em que existe o operador []>>, por exemplo T1 []>> T2, como há passagem de informação de T1 para T2, ao trocar a ordem é necessário trocar o operador para >> e apagar a pré-condição de T2, pois vai deixar de existir passagem de informação de uma tarefa para a outra e a pré-condição poderia inibir a realização da acção.

As tarefas do tipo aplicação não representam qualquer acção sobre a aplicação, limitando-se a verificar o estado da mesma, ou seja, verificar o resultado de uma sequência de acções. Por este motivo não faz sentido trocar de lugar este tipo de tarefas.

- **Generalização**

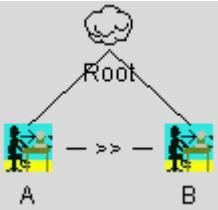
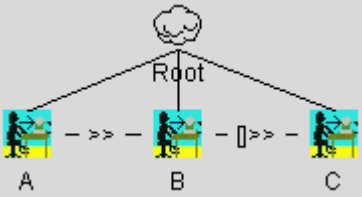
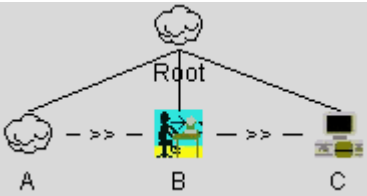
Após esta análise foi então possível obter uma estratégia generalizada que é descrita de seguida.

Para qualquer sequência de tarefas, que não seja do tipo aplicação, separadas pelo operador >> ou []>>, gerar todas as combinações de ordens possíveis e criar modelos transformados com cada uma dessas ordens. Sempre que existe passagem de informação de uma tarefa para a outra, ao trocar a sua ordem será necessário apagar a pré-condição da tarefa para a qual é transferida a informação para possibilitar a execução dos testes. Para simplificar, sempre que na sequência existir o operador []>> é necessário apagar a pré-condição da tarefa que se encontra à direita do mesmo e substitui-lo pelo operador >>.

- **Exemplos**

Na Tabela 3.3 são apresentados alguns exemplos de modelos em que estão presentes sequências de tarefas de ordem obrigatória e dos testes a realizar nesses casos. Os exemplos apresentados são exemplos genéricos e não se referem a nenhuma aplicação específica.

Tabela 3.3: Exemplos de sequências de tarefas de ordem obrigatória

Exemplo	Modelo	Testes
1		• A>>B • B>>A
2		• A>>B>>C • A>>C>>B • B>>A>>C • B>>C>>A • C>>A>>B • C>>B>>A (Em cada um dos casos apagar pré-condição de C.)
3		• A>>B>>C • B>>A>>C

3.3.3.2. Sequências de tarefas de ordem não obrigatória

Uma sequência de tarefas com ordem não obrigatória consiste num conjunto de tarefas separadas pelo operador |=|. Neste caso, o que interessa testar é se, de facto, as tarefas podem ser realizadas por qualquer ordem. Assim, de modo semelhante ao caso anterior, para a sequência T1 |=| T2, seria necessário testar T1>>T2 e T2>>T1.

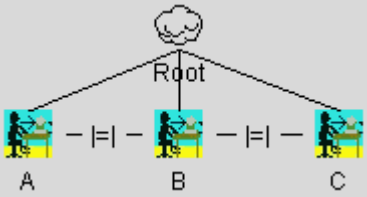
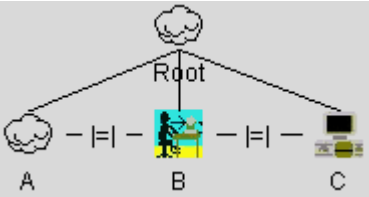
- **Generalização**

Para qualquer sequência de tarefas, que não seja do tipo aplicação, separadas pelo operador |=|, gerar todas as combinações de ordens possíveis e criar modelos transformados com cada uma dessas ordens. Substituir o operador |=| da sequência pelo operador >>.

- **Exemplos**

Na Tabela 3.4 são apresentados alguns exemplos de modelos em que estão presentes sequências de tarefas de ordem não obrigatória e dos testes a realizar nesses casos. Os exemplos apresentados são exemplos genéricos e não se referem a nenhuma aplicação específica.

Tabela 3.4: Exemplos de sequências de tarefas de ordem não obrigatória

Exemplo	Modelo	Testes
1		• A>>B>>C • A>>C>>B • B>>A>>C • B>>C>>A • C>>A>>B • C>>B>>A
2		• A>>B>>C • B>>A>>C

3.3.3.3. Tarefas opcionais e obrigatórias

Um dos erros típicos do utilizador consiste na omissão de tarefas que tem que realizar para atingir o objectivo. Num modelo de tarefas existem tarefas que são obrigatórias de realizar e tarefas que não o são. Para essas tarefas é necessário testar a interface nos casos em que estas são omitidas. As tarefas opcionais são tarefas que se encontram entre parênteses rectos, as restantes são consideradas obrigatórias.

Pela mesma razão dos casos anteriores, não faz sentido omitir tarefas do tipo aplicação já que estas não executam qualquer acção. Também não faz sentido omitir a raiz do modelo, pois é o mesmo que não existir modelo. Por outro lado, se a tarefa a omitir possuir o operador []>> à sua direita é necessário remover a pré-condição da tarefa que se encontra à sua direita, pois como vai deixar de haver passagem de informação, esta pode ser inibidora da acção. Num modelo de tarefas CTT, o operador associado a uma tarefa é o que se encontra imediatamente à sua direita, desta forma, ao omitir uma tarefa, esse será o operador a ser apagado, persistindo o que se encontra à sua esquerda. Por exemplo, no caso de A>>B[]>>C, ao omitir B, deve ficar A>>C, pois B ao ser omitido vai deixar de passar informação a C, persistindo o operador de A. Outro exemplo que permite perceber a adopção desta convenção é o caso de A[>B>>C ou A>>B[>C. No primeiro caso, A pode ser realizada até B e depois de B realiza-se C, ao omitir B, faz sentido dizer que A é realizado até C. De modo semelhante, no segundo caso, B é realizado depois de A e B pode ser realizado até C, ao omitir a tarefa B, ficaria A>>C.

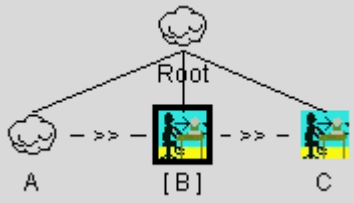
- **Generalização**

Para qualquer tarefa do modelo de tarefas que não seja de aplicação e que não seja raiz do modelo, gerar modelo transformado em que a tarefa é omitida. Se a tarefa a omitir se encontrar à esquerda do operador []>>, a pré-condição da tarefa que se encontra à sua direita deve ser apagada.

- **Exemplos**

Na Tabela 3.5 é apresentado um exemplo genérico de um modelo em que estão presentes tarefas obrigatórias e opcionais. Para este exemplo são apresentados os casos de teste necessários.

Tabela 3.5: Exemplos de modelos com tarefas opcionais e obrigatórias

Exemplo	Modelo	Testes
1		• A>>B>>C • A>>B • A>>C • B>>C

3.3.3.4. Escolha de tarefas

Num modelo de tarefas, uma escolha de tarefas consiste num conjunto de tarefas separadas pelo operador de escolha, []. Nestas situações o utilizador pode escolher qualquer uma dessas tarefas para realizar. Desta forma é necessário realizar testes em que cada uma das opções é

escolhida, verificando se isto é, de facto, uma escolha. Assim, se se estivesse perante a situação T1[]T2[]T3, teriam que ser realizados testes em que é executada T1, outro em que é executada T2 e outro em que é executada T3.

- **Generalização**

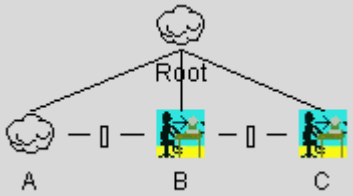
Para cada conjunto de tarefas separado pelo operador [], gerar modelos transformados em que mantém uma das tarefas e omite as restantes.

O TOM, ao gerar os modelos em Spec# já contempla estas situações, permitindo gerar testes em que cada opção é escolhida, pelo que não será necessário realizar transformações ao modelo de tarefas original.

- **Exemplos**

A Tabela 3.6 apresenta um exemplo genérico em que está presente uma escolha de tarefas e os respectivos testes a realizar.

Tabela 3.6: Exemplos de escolha de tarefas

Exemplo	Modelo	Testes
1		• A • B • C

3.3.3.5. Desactivação de tarefas

Uma desactivação de tarefas consiste em duas tarefas separadas pelo operador [>, por exemplo, T1 [> T2. Isto significa que a tarefa T1 está activa até T2 ser realizada e a qualquer momento de T1, T2 pode ser realizada. Esta situação pode também conduzir a erros, em que o utilizador tenta realizar T1 após realizar T2. Desta forma torna-se necessário testar a sequência T1>>T2 mas também T1>>T2>>T1', em que T1' é uma cópia da tarefa T1.

- **Generalização**

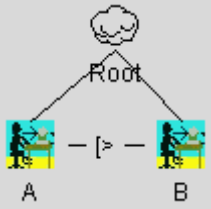
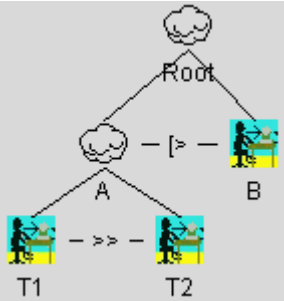
Para qualquer tarefa T que possua o operador [> à sua direita:

- Substituir operador [> por >>;
- Criar tarefa T' que é uma cópia de T;
- Identificar a tarefa T1, que se encontra à direita de T;
- Colocar T' à direita de T1, separada com o operador >>.

- **Exemplos**

Na Tabela 3.7 estão presentes alguns exemplos genéricos de modelos que possuem desactivação de tarefas, apresentando os casos de teste necessários segundo a estratégia de generalização identificada.

Tabela 3.7: Exemplos de desactivação de tarefas

Exemplo	Modelo	Testes
1		• A>>B • A>>B>>A’
2		• A (T1>>T2) >> B • A (T1>>T2) >> B >> A’ (T1’>>T2’)

3.3.3.6. Ciclos

Uma tarefa iterativa, T*, pode ser realizada iterativamente conduzindo a ciclos. Numa aplicação, um ciclo pode ocorrer, por exemplo, na interface de preenchimento de um formulário de registo, em que um utilizador pode ir preenchendo os dados iterativamente até pressionar o botão validar. Ao pressionar o botão validar, em muitos casos, quando há algum erro, a aplicação permanece na interface do formulário permitindo ao utilizador ir preenchendo dados e ir validando até que obtenha o sucesso. É muito comum nestas interfaces, nas situações de erro, o utilizador, após validação com erros, apenas necessitar de preencher a informação errada, sendo que a restante se mantém de umas iterações para as outras. Um erro típico que pode ocorrer é o utilizador esquecer-se de preencher um campo obrigatório e na segunda iteração apenas preencher esse campo assumindo que os restantes se mantiveram preenchidos. Contudo, nem todas as interfaces gravam informação de umas iterações para as outras, tendo o utilizador que voltar a preencher todos os campos. Face a estas situações é importante testar se uma

interface deste tipo grava informação de umas interacções para as outras, permitindo assim detectar a resposta da interface a este tipo de erros do utilizador.

Uma forma de simular estes erros é omitir uma tarefa obrigatória na primeira iteração e realizá-la sozinha na iteração seguinte, verificando assim se a informação das restantes tarefas foi gravada. No caso da tarefa a omitir pertencer a um conjunto de escolha de tarefas, tarefas separadas pelo operador [], não faz sentido efectuar esse teste pois ao omitir a tarefa na primeira iteração seria escolhida outra opção, logo na segunda iteração não fazia sentido escolher novamente uma tarefa dessas opções.

- **Generalização**

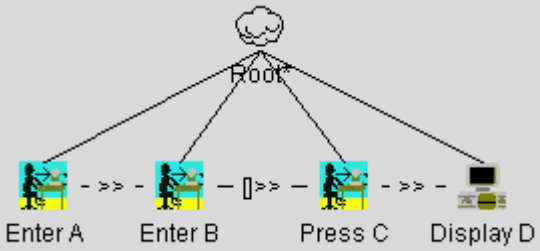
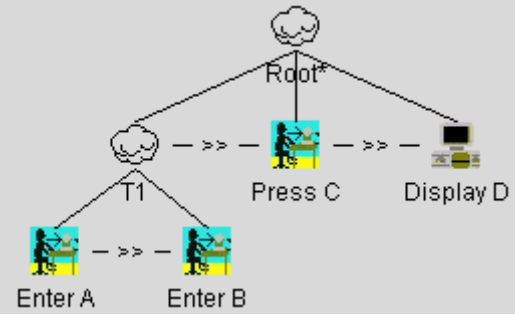
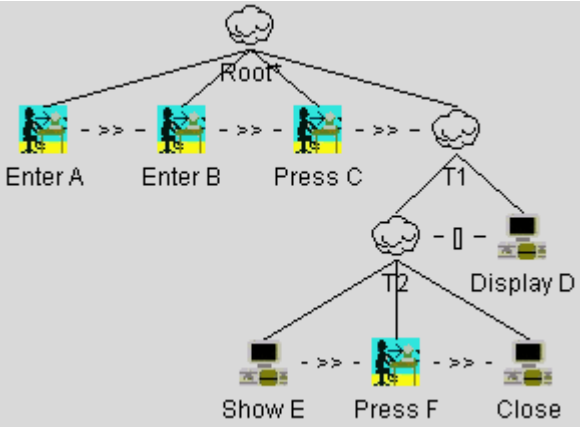
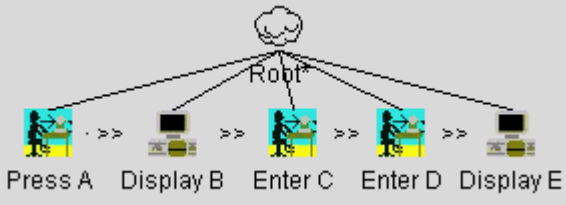
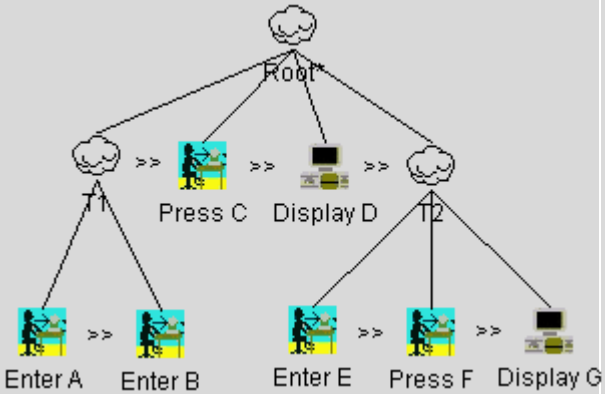
Um ciclo do tipo enunciado anteriormente consiste num conjunto de tarefas em que a raiz é uma tarefa iterativa que:

- Possua nos seus filhos uma tarefa do tipo "Press", que indica o final de uma iteração do ciclo;
- À esquerda da tarefa de "Press", todos as suas descendentes folha devem ser do tipo interacção e "Enter", que representam a inserção de dados por parte do utilizador;
- À direita da tarefa de "Press", todos os descendentes não podem ser do tipo "Enter", pois isso indicaria que a iteração ainda não tinha terminado.
- Para cada tarefa que verifique as condições anteriores e que não se encontre ao lado do operador [] deve ser gerado um modelo em que cada tarefa do conjunto de tarefas folha que se encontra antes do "Press" é omitida da sua posição actual. A tarefa omitida deve ser colocada no final dos filhos da raiz do ciclo seguida de uma copia da tarefa de "Press" e das tarefas que se encontram à sua direita. Se a tarefa a omitir possuir o operador []>> é necessário remover a pré-condição da tarefa que se encontra à sua direita.

- **Exemplos**

Apresentam-se de seguida, na Tabela 3.8, alguns exemplos que permitem perceber melhor este algoritmo de generalização de transformações de ciclos.

Tabela 3.8: Exemplos de ciclos

Exemplo	Modelo	Testes
1		• Enter B >> Press C >> Display D >> Enter A >> Press C >> Display D • Enter A >> Press C >> Display D >> Enter B >> Press C >> Display D (Apagar pré-condição de C)
2		• Enter B >> Press C >> Display D >> Enter A >> Press C >> Display D • Enter A >> Press C >> Display D >> Enter B >> Press C >> Display D
3		• Enter A >> Press C >> T1 >> Enter B >> Press C >> T1 • Enter B >> Press C >> T1 >> Enter A >> Press C >> T1
4		Não é considerado ciclo pois não existe nenhuma tarefa do tipo "Enter" antes da tarefa Press A e aparecem tarefas do tipo "Enter" depois da mesma.
5		Não é considerado ciclo pois depois da tarefa Press C existe uma tarefa do tipo "Enter", a tarefa Enter E.

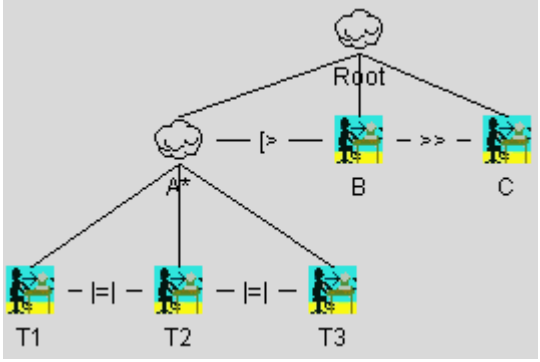
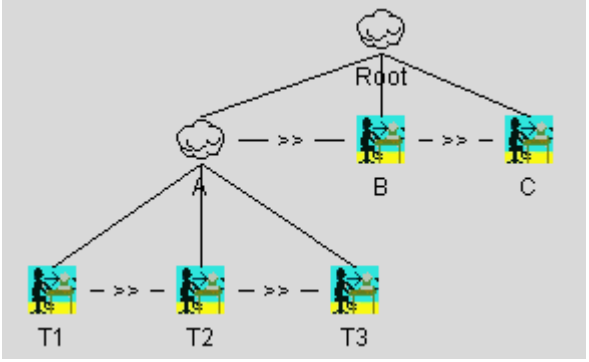
3.3.3.7. Modelo simplificado

Para possibilitar o teste das situações específicas identificadas nos casos anteriores foi necessária a realização de transformações ao modelo original, sendo criado um modelo simplificado. Este modelo tem como objectivo simplificar o modelo original, apresentando uma sequência de acções correspondente a um caso correcto de utilização da interface. Desta forma, para isto ser possível, foi necessário remover a iteratividade das tarefas, deixando assim de haver ciclos. Foi também necessário substituir o operador |=| pelo operador >>, fazendo assim com que a sequência de tarefas de ordem não obrigatória seja realizada pela ordem com que aparece no modelo. Outra transformação também necessária foi a substituição do operador [> pelo operador >> indicando, deste modo, que as tarefas separadas por este operador, serão realizadas pela ordem em que se encontram.

Este modelo simplificado, apresenta situações correctas de funcionamento para cada um dos casos identificados anteriormente. Deste modo, quando se pretende testar uma situação específica de um dos casos anteriores, as restantes devem estar definidas para que representem situações correctas de funcionamento. Só assim é permitido testar a situação pretendida, não havendo erros nas restantes. Para isto acontecer, as transformações indicadas nos algoritmos anteriores devem ser detectadas no modelo original mas realizadas sobre o modelo simplificado.

De seguida, Tabela 3.9, é apresentado um exemplo de um modelo de tarefas e o respectivo modelo simplificando, aplicando as transformações referidas anteriormente.

Tabela 3.9: Exemplo de simplificação de um modelo de tarefas

Modelo Original	Modelo Simplificado
	

Como se pode observar na tabela anterior, foi retirada a iteratividade da tarefa A, o operador [> que se encontrava à direita da tarefa A foi substituído pelo operador >> e os operadores que relacionavam as tarefas T1, T2 e T3 foram também substituídos pelo operador

>>. O modelo simplificado obtido a partir dessas transformações apresenta uma sequência correcta de funcionamento do modelo original.

3.3.4 Definição de diferentes estratégias

O algoritmo de transformações identificado na secção anterior permite transformar os modelos de forma a ser possível testar erros do tipo deslizamentos (*slips*) e lapsos (*lapses*). No entanto, não é possível testar os erros do tipo enganos (*mistakes*) recorrendo a transformações do modelo original.

Existem diversas GUIs que possuem funcionalidades semelhantes mas que apresentam estratégias diferentes para a sua realização. Isto pode contribuir para que o utilizador ao transitar de interfaces execute a estratégia errada, tornando assim necessário o seu teste perante estas situações.

Uma forma de testar as GUIs face a este tipo de erros consiste na definição de diferentes modelos de tarefas para a mesma interface. Cada um destes modelos iria definir diferentes planos de execução para alcance do mesmo objectivo. A interface seria testada com os diferentes planos de execução, permitindo assim o seu teste face a execuções do plano errado.

Em [CSP09], é apresentado um exemplo em que esta situação pode ocorrer. O exemplo apresentado diz respeito à funcionalidade de pesquisa da aplicação Notepad. Nesta funcionalidade é necessário introduzir o texto a pesquisar e clicar no botão de pesquisa para os resultados da pesquisa serem apresentados. Contudo, em algumas aplicações, a pesquisa é incremental e os resultados vão aparecendo à medida que se introduz o texto da pesquisa. Os autores de [CSP09] propuseram uma estratégia de teste desta funcionalidade do Notepad baseada na pesquisa incremental e verificaram que o comportamento da aplicação face a estas duas estratégias era, de facto, diferente.

3.3.5 Geração e execução dos testes

Como foi referido, com a ferramenta TOM [CSP09, SCP08], são gerados oráculos de teste, em Spec#[BLS05], que transformam as tarefas dos modelos de tarefas em acções desta especificação. Estes oráculos, após algumas adaptações e configurações, são então utilizados pela ferramenta Spec Explorer [VCGSTN08] para a geração e execução dos testes.

Em Spec#, as acções podem ser de diversos tipos: *controllable, probe*, *observable* e *scenario*. As tarefas de interacção correspondem a tarefas do tipo *controllable*, pois descrevem acções que são controladas pelo utilizador. As tarefas de aplicação correspondem a tarefas do tipo *probe* pois representam acções que apenas analisam o estado do sistema, não realizando

qualquer alteração. Estas tarefas permitem verificar se, numa dada altura, a aplicação se encontra no estado pretendido.

Para geração dos casos de teste, a ferramenta Spec Explorer [VCGSTN08], permite a definição dos valores com os quais se pretende executar cada acção do modelo. Com esta informação, é gerada uma máquina de estados finita (FSM) que representa as sequências de transições possíveis que os testes podem seguir. Após geração desta máquina de estados é possível gerar automaticamente casos de teste que testem as diversas transições da máquina de estados.

Para a execução dos testes gerados, existe uma funcionalidade que permite fazer a correspondência entre as acções do modelo e da aplicação. Como esta ferramenta não se encontra adaptada para o teste de interfaces gráficas surgiu a necessidade de mapeamento das acções lógicas do modelo para acções físicas a nível dos objectos da interface gráfica.

Com este mapeamento realizado, os testes podem então ser executados de forma automática pelo Spec Explorer, sendo apresentados os seus resultados de sucesso ou insucesso.

3.4 Resumo e Conclusões

A abordagem definida consiste numa abordagem modular que passa por diversas etapas de processamento. A primeira etapa consiste na especificação dos modelos de tarefas, a segunda consiste na elaboração de uma estratégia genérica de introdução de erros nestes modelos. Por fim a última etapa consiste na geração e execução dos testes.

Para especificação dos modelos de tarefas foi seleccionada a notação CTT. As características desta notação permitem a simplicidade de construção e análise de modelos de tarefas de interfaces gráficas com o utilizador. Por outro lado, a existência de ferramentas de suporte, nomeadamente a ferramenta Teresa, facilitam bastante o desenvolvimento destes modelos e permitem também a especificação de características adicionais para as tarefas.

Existem três tipos de erros típicos de comportamento do utilizador identificados por Reason em [Rea90]. Em termos de tarefas esses erros traduzem-se essencialmente na omissão ou troca de tarefas num modelo, ou então na utilização do modelo errado para atingir um objectivo.

Para introdução dos dois primeiros tipos de erros foi necessária a definição de um algoritmo genérico. A generalização deste algoritmo não foi uma tarefa trivial uma vez que os modelos de tarefas variam de GUI para GUI. Desta forma foram analisadas diversas GUIs a fim de detectar padrões nas especificações das suas tarefas. Após identificação destes padrões foi então definida a estratégia de alteração de modelos de forma a contemplar os diversos testes que cada um deve ter.

Para teste do terceiro tipo de erros a abordagem definida consiste no desenvolvimento de diferentes planos de execução para um mesmo objectivo. Posteriormente a aplicação em teste seria testada com cada um desses planos.

Para geração e execução dos testes, o auxílio prestado pela ferramenta Spec Explorer no sentido da sua automatização foi muito útil. Contudo foi necessária a realização de algumas adaptações para que esta fosse utilizada no contexto do teste de interfaces gráficas com o utilizador.

Capítulo 4

Detalhes de Implementação

Neste capítulo serão apresentados alguns detalhes e aspectos mais tecnológicos relativos ao desenvolvimento e implementação da ferramenta proposta, a CMT Tool, e ao mapeamento dos objectos da GUI para objectos do modelo.

4.1 CMT Tool

Para validar a abordagem identificada no capítulo anterior, foi desenvolvida a CMT Tool (*CTT Model Transformation Tool*). A CMT Tool recebe um modelo de tarefas especificado em CTT exportado para XML e aplica o algoritmo de transformações definido no capítulo da abordagem (secção 3.3.3), gerando novos modelos para as respectivas transformações.

Esta ferramenta foi desenvolvida em Java, sendo utilizada a linguagem XPath [CD10] para exploração do modelo de tarefas em XML. Esta linguagem permite construir expressões que processam documentos XML. Desta forma, foi possível extrair directamente do documento XML a informação do modelo de tarefas para uma lista de nós. Cada que nó representa uma tarefa do modelo com todos os seus atributos. Para cada uma das tarefas extraída, foi também possível extrair a informação relativa aos seus filhos, permitindo assim representar a estrutura em árvore do modelo de tarefas.

Após obtenção desta estrutura de representação do modelo de tarefas foram implementados os diversos algoritmos apresentados na secção 3.3.3. Os algoritmos recorrem à estrutura do modelo para realizar as transformações necessárias e posteriormente gravar os modelos transformados em novos ficheiros XML.

No caso do algoritmo relativo às sequências de tarefas obrigatórias e não obrigatórias, foi necessária a introdução de uma limitação do número de transformações realizadas. Para

testar estas sequências é necessário gerar modelos que contemplem todas as ordens possíveis de execução. No entanto, para sequências de maiores dimensões, contemplar todas as possibilidades, poderia levar a uma explosão de modelos transformados. Desta forma, foi imposta uma limitação ao número de transformações realizadas. Assim, foi decidido que, no máximo, seriam realizadas apenas 10 transformações, sendo estas escolhidas aleatoriamente de entre todas as combinações de ordem de tarefas possíveis.

4.2 Mapeamento de elementos da GUI

Para efectuar o mapeamento entre os elementos da implementação e da interface gráfica com o utilizador (GUI) foi criada uma biblioteca genérica que contém as acções básicas comuns a qualquer interface como, por exemplo, inserir texto, clicar num botão, entre outras. A construção desta biblioteca foi realizada recorrendo às funcionalidades permitidas pela Framework UI Automation [Cor10], desenvolvida pela Microsoft. Esta Framework permite o acesso a todos os elementos de interface com o utilizador existentes no Ambiente de Trabalho do computador e a realização de acções sobre os mesmos.

Após construção desta biblioteca genérica, para cada aplicação a testar, foi criada uma nova biblioteca com as funcionalidades específicas da aplicação em questão, recorrendo às funcionalidades da biblioteca genérica.

Este processo de mapeamento foi realizado manualmente, no entanto, existe uma ferramenta, a GUI Mapping Tool [PVST05], que permite a sua automatização.

4.3 Conclusões

Para validar a abordagem definida foi desenvolvida a ferramenta CMT Tool. Esta ferramenta recebe o ficheiro XML de um modelo de tarefas especificado em CTT e aplica um conjunto de transformações que geram modelos de teste definidos na secção 3.3.3. Esta ferramenta foi desenvolvida em Java e tira partido das facilidades da linguagem XPath [CD10]. Esta linguagem possibilita a rápida extracção da estrutura de tarefas do modelo de tarefas.

Para execução dos testes foi necessário efectuar o mapeamento entre os elementos da implementação e da interface gráfica com o utilizador (GUI). Este mapeamento foi possível recorrendo às funcionalidades da Framework UI Automation [Cor10].

Capítulo 5

Casos de Estudo

Para avaliar a abordagem referida foram seleccionadas algumas interfaces gráficas de aplicações já existentes para utilização como casos de estudo. Cada caso de estudo foi utilizado para testar alguns dos erros típicos do utilizador com base nas estratégias de teste propostas no Capítulo 3, que define a abordagem seguida. Como objecto de caso de estudo foram seleccionadas algumas aplicações baseadas na Web. As aplicações seleccionadas foram as seguintes:

- **Reservas online Vip Hotels:** Neste caso de estudo foi analisado o sistema de reservas online do grupo Vip Hotels, que pode ser acedido através do endereço www.viphotels.com;
- **Pesquisa de imóveis da imobiliária Agimoura:** Neste caso de estudo foi analisado o sistema de pesquisa de imóveis da imobiliária Agimoura, que pode ser acedido através do endereço www.agimoura.com;
- **DokuWiki:** Neste caso de estudo foram analisadas algumas funcionalidades de uma wiki DokuWiki. As funcionalidades estudadas foram as funcionalidades de Login e alteração de palavra-chave.
- **Conversor monetário:** Neste caso de estudo foi analisado o conversor monetário XE, que pode ser acedido através do endereço http://www.xe.com/ucc/po/;
- **Conversor de unidades:** Neste caso de estudo foram analisadas duas aplicações de conversão de unidades com estratégias de execução diferentes. As aplicações analisadas podem ser acedidas em http://www.peters1.dk/webtools/conversion.php?sprog=en e http://www.digitaldutch.com/unitconverter/area.htm.

De seguida apresenta-se uma descrição mais detalhada dos casos de estudo analisados e dos resultados observados.

5.1 Reservas online Vip Hotels

O grupo Vip Hotels possui uma cadeia de hotéis em várias regiões. Através do seu endereço, www.viphotels.com, é possível aceder a diversas funcionalidades. Neste caso de estudo apenas foi analisada a funcionalidade de reservas online.

A funcionalidade de reservas online permite fazer uma pesquisa dos quartos disponíveis nos hotéis do grupo para as datas pretendidas e posteriormente é realizada a reserva. Apenas foi analisada a parte de pesquisa. A Figura 5.1 apresenta o menu de reservas online.

Figura 5.1: Menu de reservas online do grupo Vip Hotels

No menu de reservas, todos os campos são de preenchimento obrigatório, sendo que as escolhas do campo Hotel variam de acordo com a cidade seleccionada, indicando os possíveis hotéis para a cidade seleccionada.

Após análise deste menu de reservas foi criado o modelo de tarefas respectivo, na ferramenta Teresa, apresentado na Figura 5.2.

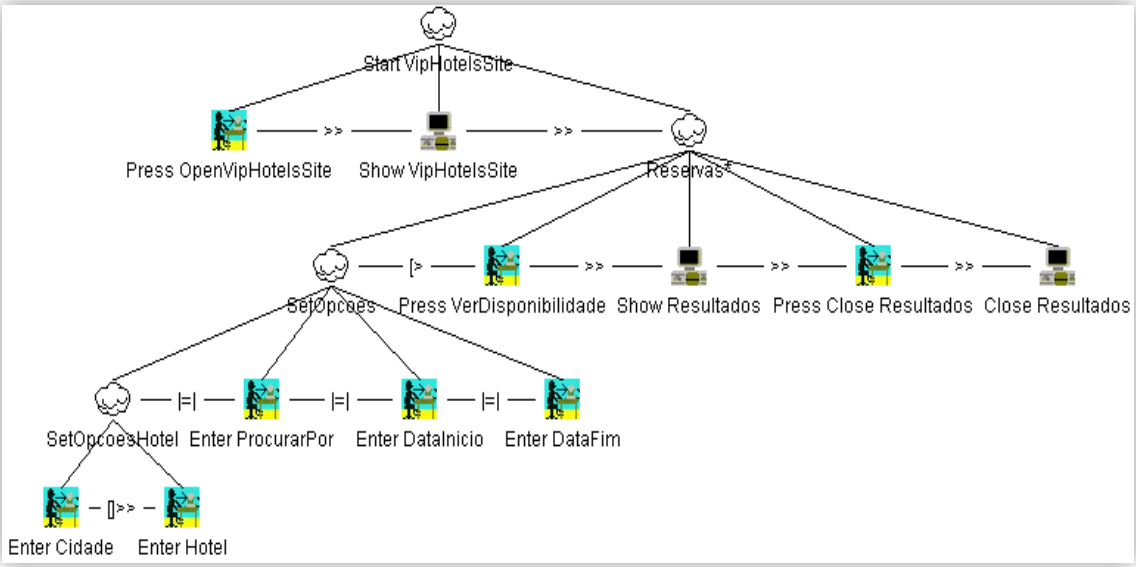


Figura 5.2: Modelo de tarefas do menu de reservas online do site do Vip Hotels

Como se pode observar pela análise do modelo da Figura 5.2, após abertura da página do grupo Vip Hotels, o menu de reservas encontra-se logo disponível. As opções do menu de reservas podem ser escolhidas por qualquer ordem à excepção das opções de cidade e hotel. Estas opções devem ser escolhidas pela ordem definida no modelo, ou seja, em primeiro lugar deve ser escolhida a cidade e só depois desta escolha é que deve ser escolhido o hotel. Isto acontece pois a lista de escolha de hotéis depende da cidade escolhida, deste modo, há passagem de informação de uma tarefa para a outra. Apesar de todos os campos serem obrigatórios, estes já possuem valores por defeito ao iniciar a aplicação, pelo que a qualquer momento da escolha das opções, tarefa abstracta "SetOpcoesHotel", é possível clicar no botão de "Ver Disponibilidade". Após execução desta acção é apresentada uma nova janela com os resultados dos quartos disponíveis para as opções indicadas.

Com a ferramenta desenvolvida, a CMT Tool, foram realizadas as diversas transformações ao modelo original consoante os algoritmos definidos previamente. Neste caso de estudo apenas vamos analisar alguns desses modelos transformados.

5.1.1 Modelo Simplificado

Em primeiro lugar foi utilizado o modelo simplificado para testar a aplicação com uma sequência correcta de funcionamento. A Figura 5.3 apresenta o modelo simplificado.

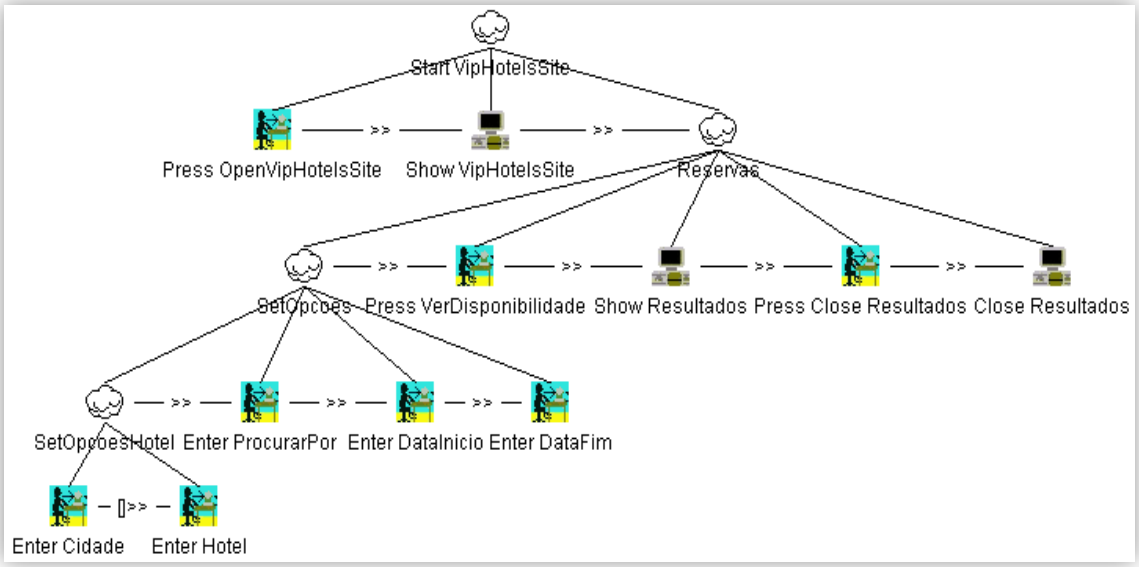


Figura 5.3: Modelo de tarefas simplificado do menu de reservas online do site do Vip Hotels

A partir do modelo simplificado, com a ferramenta Teresa, foi gerado o respectivo PTS. De seguida, recorrendo ao TOM, foi gerado o modelo em Spec# para utilizar em SpecExplorer. Para execução dos testes, foram escolhidos os seguintes valores válidos:

- **Cidade:** Lisboa;
- **Hotel:** Vip Inn Berna;
- **Procurar Por:** Quartos;
- **Data de Início:** 29/09/2010;
- **Data de Fim:** 30/09/2010.

Para os valores anteriores foi gerada a respectiva máquina de estados (FSM) no SpecExplorer que apresenta a sequência de tarefas identificada na Figura 5.3. A partir dessa máquina de estados foram então gerados os casos de teste que percorrem essa sequência de acções.

Ao executar os testes verificou-se o sucesso dos mesmos, o que era esperado visto que este modelo retrata uma sequência correcta de funcionamento. Para avaliar o sucesso dos testes foi criada uma acção do tipo *probe* muito simplificada que apenas se limita a procurar um texto específico na janela de resultados após execução da tarefa "Press verDisponibilidade". Caso este texto seja encontrado então significa que os testes executados tiveram sucesso. Esta acção poderia, no entanto, ser mais refinada, procurando exactamente os resultados que a janela deveria apresentar. Cabe ao profissional de testes definir a estratégia de validação dos testes que considere adequada.

5.1.2 Sequência de tarefas de ordem obrigatória

Ao utilizar esta aplicação, o utilizador pode-se enganar e trocar a ordem pela qual realiza a escolha das opções. Para testar a interface face a este engano foi utilizado o modelo transformado que troca a ordem das opções da tarefa SetOpcoesHotel, como ilustrado na Figura 5.4. Ao trocar a ordem destas duas tarefas foi necessário substituir o operador de sequência com passagem de informação pelo operador de sequência, pois deixa de haver passagem de informação da cidade para o hotel.

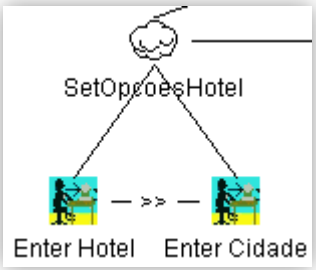


Figura 5.4: Ordem obrigatória trocada no menu de reservas online do site do Vip Hotels

Para realizar o teste com estas duas tarefas na ordem trocada foram utilizados os mesmos valores utilizados no teste do modelo simplificado. Uma vez que os valores utilizados são os mesmos e uma vez que a interface permite escolher primeiro o hotel e só depois a cidade, devido ao facto de estas opções já terem valores por defeito, o utilizador poderia esperar que os resultados da pesquisa fossem os mesmos. No entanto, ao executar os testes com o SpecExplorer verifica-se que existe uma falha pois os resultados obtidos são diferentes dos esperados. Isto acontece porque após a escolha da cidade, o valor de hotel é alterado automaticamente para o seu valor pré-definido que é a opção "Todos", o que leva a que os resultados da pesquisa sejam diferentes. Esta é uma característica que pode não ser muito intuitiva para o utilizador e que representa um dos erros típicos cometidos. Esta abordagem permitiu testar a interface contra este tipo de erros do utilizador, cabendo agora ao profissional de testes decidir os resultados esperados para estas situações, ou seja, decidir em que situações os testes passam ou falham.

5.1.3 Sequência de tarefas de ordem não obrigatória

Pela observação do modelo de tarefas original, representado na Figura 5.2, verifica-se uma independência na ordem de realização das tarefas de SetOpcoes. Ao gerar as transformações com a CMT Tool são gerados dez modelos transformados contendo a sequência de tarefas da tarefa "SetOpcoes" com diferentes ordens de tarefas. Para testar se estas tarefas se podem

realizar, de facto, por qualquer ordem teriam que ser gerados testes para cada um desses modelos e verificar se, de facto, ocorre sucesso dos mesmos. Neste caso de estudo foi analisado o modelo transformado em que as tarefas se encontram pela ordem indicada na Figura 5.5.

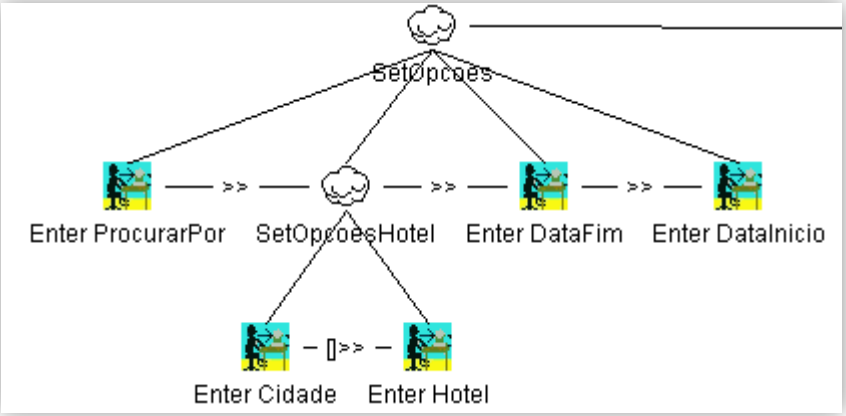


Figura 5.5: Ordem não obrigatória trocada no menu de reservas online do site do Vip Hotels

Ao realizar os testes com os mesmos dados e critérios de passagem de testes utilizados para o modelo simplificado não deveria ocorrer nenhuma falha visto que o modelo especifica que a ordem não é importante. Ao correr os testes no Spec Explorer verifica-se o sucesso dos mesmos.

5.2 Pesquisa de imóveis da imobiliária Agimoura

A sociedade Agimoura consiste numa sociedade de mediação imobiliária. A sua página Web, www.agimoura.com, possui um conjunto de funcionalidades de compra, venda, pesquisa de imobiliário, entre outras.

Neste caso de estudo apenas a funcionalidade de pesquisa rápida de imóveis foi analisada com maior detalhe. Esta funcionalidade trata-se de uma simples pesquisa em que é necessário fornecer o tipo de imobiliário, o tipo de informação que pretendemos fornecer para pesquisa (Concelho, Localidade, Preço, Referência, Tipologia, …) e a informação relativa ao tipo anterior, conforme é possível visualizar na Figura 5.6.

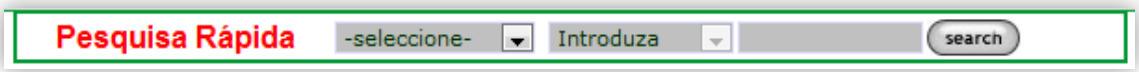


Figura 5.6: Menu de pesquisa rápida de imóveis da Agimoura

Semelhante ao caso de estudo anterior, foi desenvolvido na ferramenta Teresa o modelo de tarefas, que se apresenta na Figura 5.7.

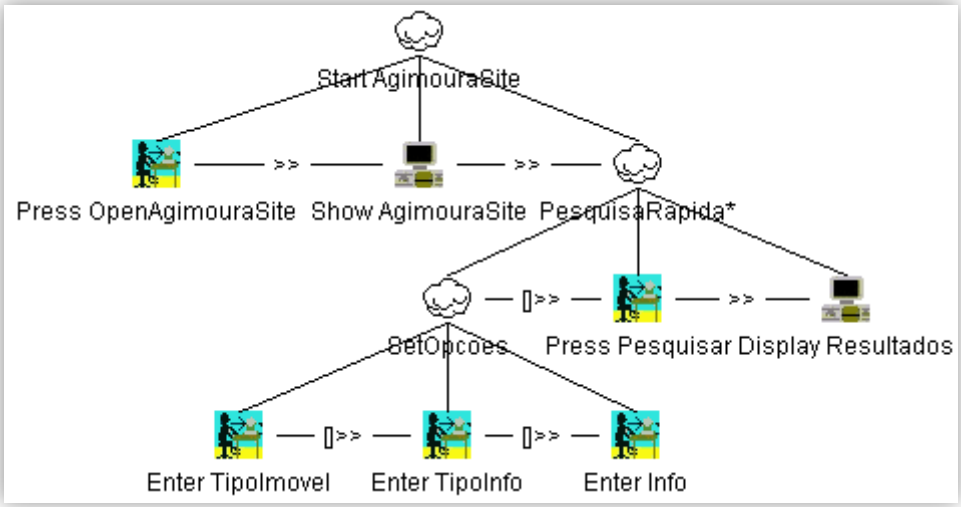


Figura 5.7: Modelo de tarefas do menu de pesquisa rápida do site da Agimoura

Como se pode observar na figura anterior, as opções de pesquisa, filhos da tarefa SetOpcoes, são todas de preenchimento obrigatório e têm que ser preenchidas pela ordem apresentada no modelo, havendo passagem de informação de umas para as outras. Para representar esta situação foi necessário adicionar algumas informações extra a essas tarefas, relativamente às suas pré-condições. A escolha da opção "TipoInfo" só pode ser realizada se a opção "TipoImovel" não for vazia, assim como a opção "Info" só pode ser realizada se a opção "TipoInfo" não for vazia. Foi também necessário adicionar uma pré-condição à tarefa "Press Pesquisar" indicando que as opções da tarefa "SetOpcoes" não podem ser vazias.

Com as pré-condições já definidas pode-se passar à geração das transformações ao modelo de tarefas pela CMT Tool. Para este caso de estudo serão analisadas as transformações relativas à troca de ordem das opções de pesquisa, assim como à omissão de uma delas.

5.2.1 Modelo Simplificado

Em primeiro lugar foram realizados testes relativos a uma sequência correcta de funcionamento utilizando, para isso, o modelo simplificado gerado, que se encontra na Figura 5.8.

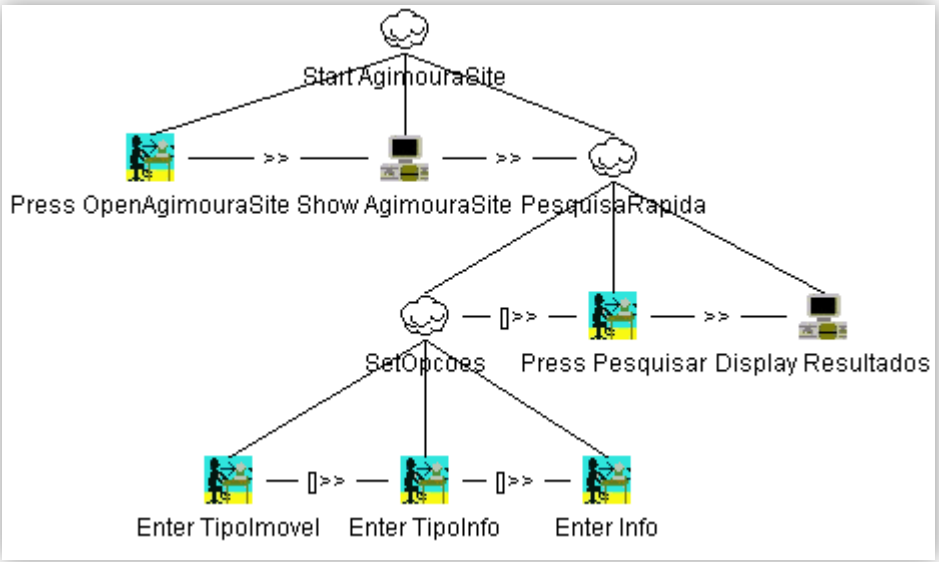


Figura 5.8: Modelo de tarefas simplificado do menu de pesquisa rápida do site da Agimoura

Como se pode observar pelo modelo da Figura 5.8, as únicas transformações que o modelo original sofreu foi a alteração do atributo iterativo da tarefa "PesquisaRapida".

Após construção do modelo Spec# para este modelo com a ajuda do TOM, foram então gerados e executados os testes em SpecExplorer. Para execução dos testes os dados escolhidos foram:

- **Tipo de Imóvel:** Vivenda;
- **Tipo de Informação:** Concelho;
- **Informação:** Albufeira.

De modo semelhante ao caso de estudo anterior, foi definida uma acção do tipo *probe* que verifica se, a seguir à realização da tarefa "Press Pesquisar", os resultados obtidos são os esperados. Para isso, esta acção procura na página o número de resultados encontrados em função do número de páginas.

Como seria de esperar, a execução dos testes em SpecExplorer foi um sucesso pois os dados introduzidos foram os correctos e a sequência de tarefas foi respeitada.

5.2.2 Sequência de tarefas obrigatória trocada

Apesar da ordem das opções especificada no modelo ser obrigatória, existe sempre a possibilidade do utilizador se enganar e tentar executar as tarefas por ordem errada. Para testar estas situações foi seleccionado um dos modelos transformados em que as tarefas de "SetOpcoes" se encontram especificadas por ordem diferente. O modelo transformado escolhido foi o que apresenta as tarefas pela ordem representada na Figura 5.9.

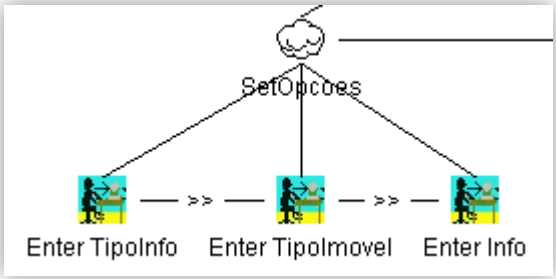


Figura 5.9: Ordem obrigatória trocada no menu de pesquisa rápida do site da Agimoura

Para o modelo transformado da Figura 5.9 foi gerado o respectivo PTS e posteriormente o seu modelo em Spec#. O utilizador, ao realizar a sequência de tarefas pela ordem errada, poderia estar à espera que os resultados da pesquisa fossem os mesmos que os obtidos no teste simplificado. Desta forma, foi utilizada a mesma acção de *probe* especificada para o teste do modelo simplificada.

Após execução dos testes verificou-se o insucesso dos mesmos. O que acontece é que, na execução dos testes, não é possível seleccionar o tipo de informação antes da selecção do tipo de imóvel, pois esta opção fica inactiva. Como não foi possível seleccionar o tipo de informação, o botão de pesquisa também não fica activo, não sendo possível simular o seu clique. No final da execução das acções, verifica-se que na interface não foi realizada nenhuma pesquisa pelo que a acção de *probe* obtém resultados diferentes dos esperados no teste do modelo simplificado, resultando no insucesso da mesma. Neste caso os testes falharam, o que indica que a sequência de tarefas não pode ser realizada por esta ordem.

O sucesso ou insucesso dos testes depende, mais uma vez, da especificação utilizada pelo profissional de testes. Se a função de *probe* estivesse definida, por exemplo, de forma a verificar que a pesquisa não tinha sido efectuada, os testes resultariam em sucesso, permitindo concluir o mesmo que o caso estudado.

5.2.3 Omissão de tarefa obrigatória

Outro erro típico que poderia ocorrer, também contemplado nas transformações do modelo, é o caso em que o utilizador se esquece de preencher uma das opções da pesquisa, que são tarefas obrigatórias. Para representar esta situação foi seleccionado o modelo em que a tarefa “Enter Info” foi omissa do modelo de tarefas. Nesta situação, seria esperado algo semelhante ao que aconteceu no caso anterior.

Na execução dos testes verifica-se que, como não foi seleccionado o tipo de informação, o botão de pesquisa fica inactivo, não permitindo a realização de nenhuma pesquisa. Deste modo,

como aconteceu no teste anterior, os resultados dos testes foram de insucesso, indicando que a tarefa “Enter Info” é realmente obrigatória.

5.3 DokuWiki

Este caso de estudo pretende testar algumas funcionalidades de uma wiki do tipo DokuWiki. Numa DokuWiki existem diversas funcionalidades que se podem realizar. Para este caso de estudo foram seleccionadas duas dessas funcionalidades: login e alteração de palavra-chave. Para cada uma dessas funcionalidades foram realizados alguns testes. Segue-se a descrição de cada um destes casos.

5.3.1 Login

Neste caso de estudo foi analisada a funcionalidade de login da dokuwiki. Para aceder a esta funcionalidade é necessário aceder ao site da dokuwiki e em seguida clicar no botão de login, aparecendo o menu de login identificado na figura Figura 5.10.

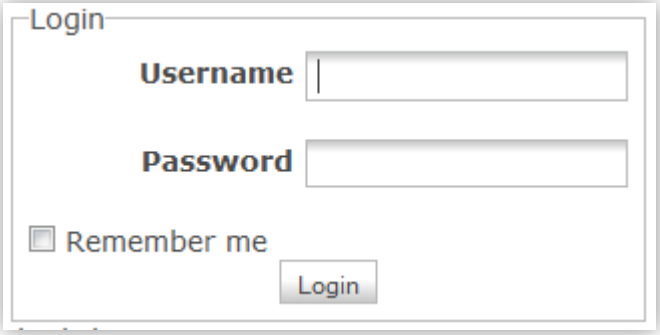


Figura 5.10: Menu de login da DokuWiki

Para este menu de login foi criado o respectivo modelo de tarefas que se encontra na Figura 5.11.

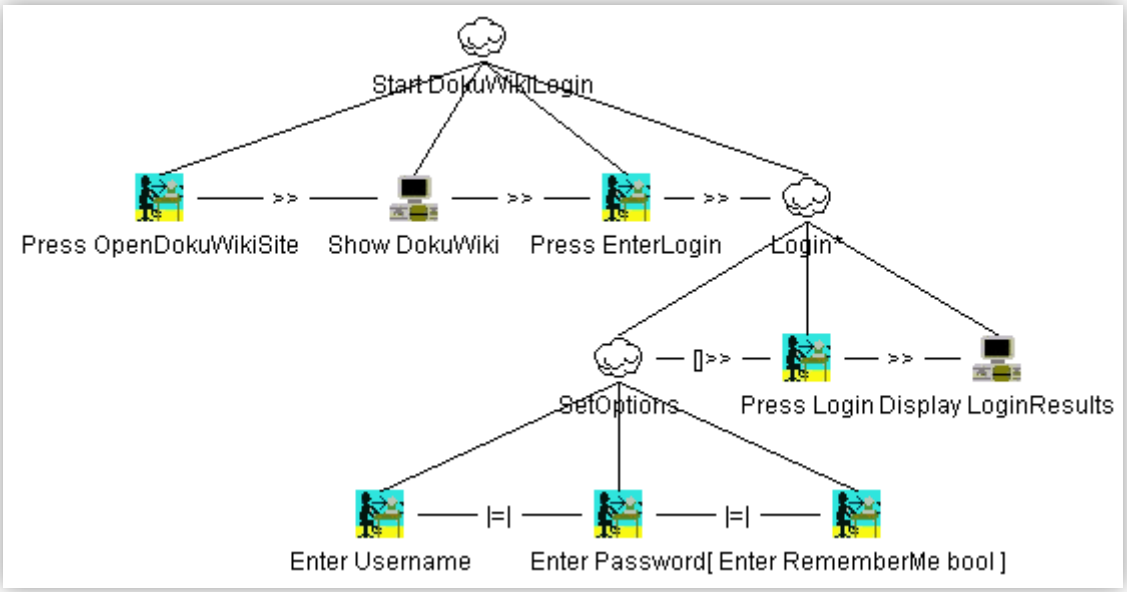


Figura 5.11: Modelo de tarefas do menu de login da DokuWiki

Como se pode observar na figura anterior, após realização da tarefa "Press EnterLogin" o menu de login fica activo. No menu de login, é possível realizar as tarefas de inserção de nome do utilizador, inserção de palavra-chave e selecção da opção de memorizar dados do login. Estas tarefas constituem a tarefa "SetOptions" e podem ser realizadas por qualquer ordem, sendo que a última é opcional. Após selecção das opções é pressionado o botão de login, tarefa "Press LoginDisplay". Dependendo dos dados das opções de login preenchidos são apresentados resultados de sucesso ou insucesso do login. Este menu de login está representado como um ciclo (Login*), uma vez que este é realizado iterativamente até que o resultado do login seja de sucesso.

5.3.1.1 Modelo simplificado

Para testar uma sequência correcta de funcionamento do menu de login, foi utilizado o modelo simplificado gerado pela CMT Tool. Este modelo é apresentado na Figura 5.12.

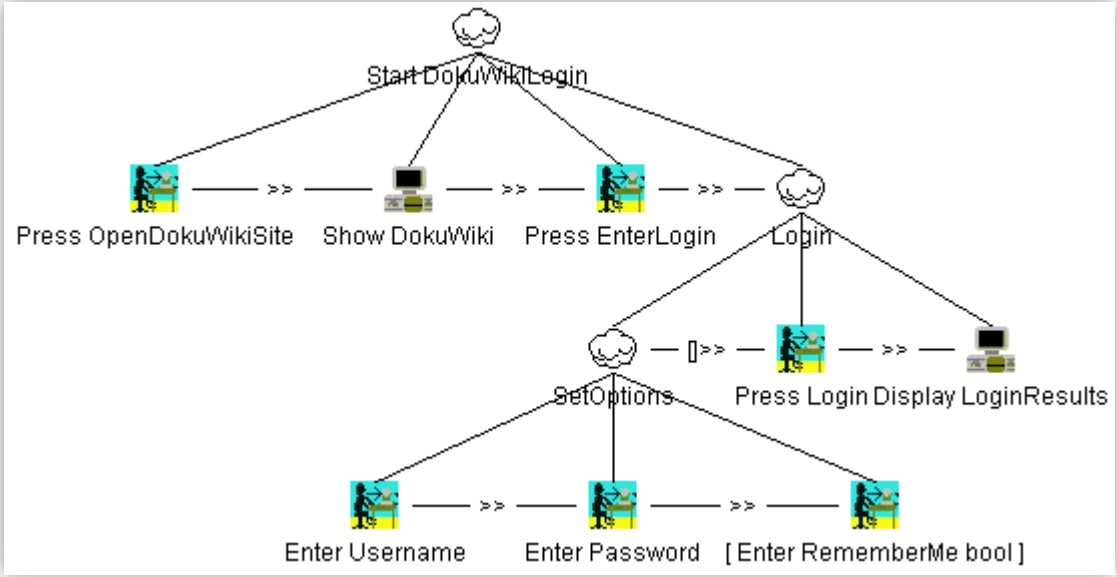


Figura 5.12: Modelo simplificado do menu de login da DokuWiki

Para este modelo foi gerado o respectivo PTS na ferramenta Teresa e posteriormente o modelo Spec# no TOM. Para execução dos testes foram utilizados dados válidos de um utilizador registado na wiki de teste. Para verificar o sucesso dos testes foi criada uma acção de *probe* que verifica se o login foi realizado com sucesso, ou seja, se após realização da sequência de tarefas a wiki apresenta informação indicando que o login teve sucesso. Ao executar os testes verificou-se o sucesso dos mesmos, como seria de esperar.

5.3.1.2 Omissão de tarefa opcional

Como já foi referido, a tarefa "Enter RememberMe" é opcional pelo que a sua omissão não deveria afectar o sucesso do login. Para analisar se, de facto, esta opção é mesmo opcional na interface do menu de login foi utilizado o modelo transformado em que a mesma é omissa, como se apresenta na Figura 5.13.

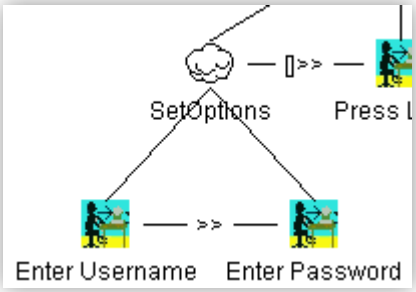


Figura 5.13: Omissão de tarefa opcional no menu de login da DokuWiki

De modo semelhante ao teste do modelo simplificado, foram gerados e executados os testes e observaram-se os mesmos resultados. Isto permite concluir que a tarefa "Enter RememberMe" é mesmo opcional e que a interface, neste aspecto, se encontra especificada de acordo com o modelo de tarefas.

5.3.1.3 Ciclo

Como se pode observar no modelo de tarefas da Figura 5.11 a tarefa iterativa "Login" apresenta as características de um ciclo, definidas no algoritmo de transformações apresentado na secção 3.3.3. Desta forma, pretende-se testar se, de umas iterações para as outras, a informação especificada é guardada. Para testar estas situações foram gerados vários modelos transformados pela CMT Tool, cada um desses modelos permite testar a gravação da informação das tarefas de "SetOpcoes", omitindo cada uma delas na primeira iteração do ciclo. Neste caso de estudo apenas foi analisado o teste relativo à omissão da tarefa "Enter Password" na primeira iteração, cujo respectivo modelo se encontra na Figura 5.14.

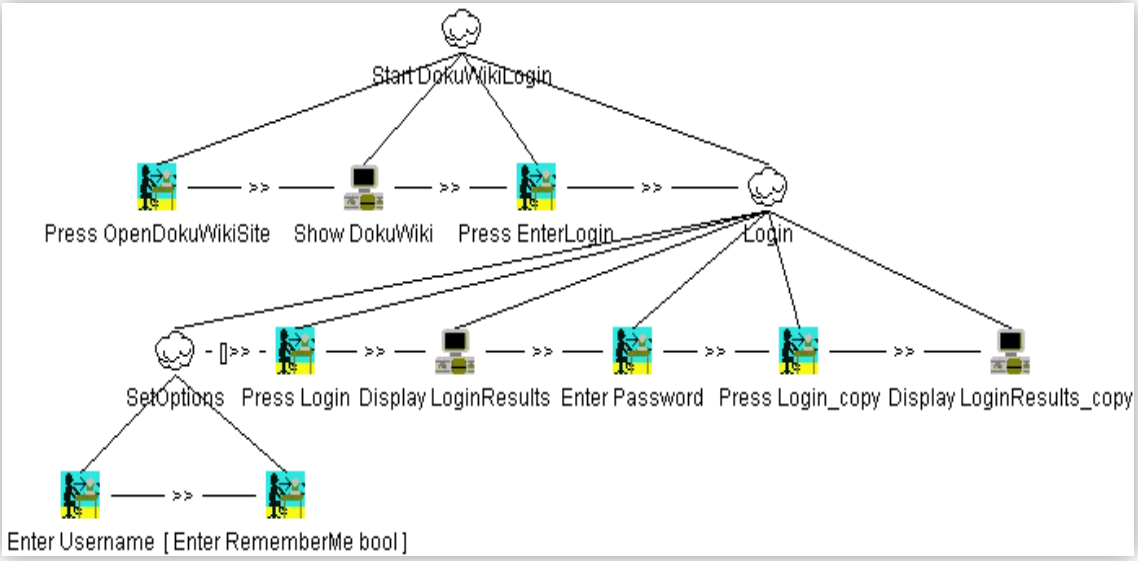


Figura 5.14: Modelo de tarefas de teste de ciclo do menu de login da DokuWiki

Como se pode observar no modelo da Figura 5.14, a tarefa "Enter Password" foi omitida na primeira iteração do ciclo e executada na segunda, permitindo assim analisar se as restantes tarefas mantiveram os seus dados de uma iteração para a outra. Para realizar esta análise foi necessária a criação de duas acções do tipo *probe*. A primeira verifica o insucesso do login na primeira iteração, pois o campo da palavra-chave não é preenchido. A segunda verifica o sucesso do login no final da segunda iteração, indicando assim se houve ou não gravação de informação. Após execução dos testes, verificou-se o sucesso dos mesmos, o que significa que para esta situação há gravação de informação de uma iteração para a outra. Para comprovar se

isto se verifica sempre teriam que ser testados todos os modelos transformados relativos a este ciclo.

5.3.2 Alteração de palavra-chave

Neste caso de estudo foi analisada a funcionalidade de alteração de palavra-chave da DokuWiki. Esta funcionalidade, após efectuar login na wiki, encontra-se disponível após acesso ao menu de actualização de perfil do utilizador. A Figura 5.15 apresenta o menu de actualização de perfil do utilizador da DokuWiki.

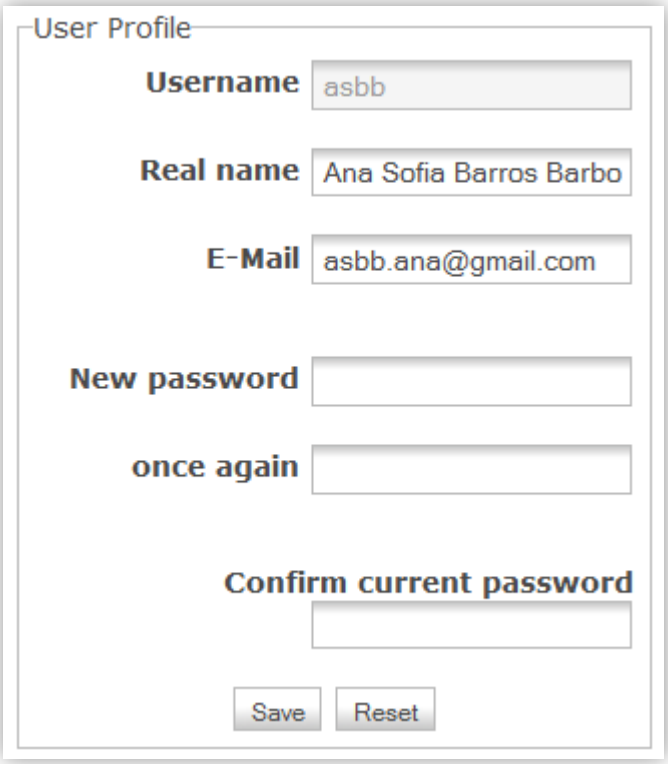


Figura 5.15: Menu de actualização de perfil de utilizador da DokuWiki

Para simplificar, neste caso de estudo apenas foram contemplados os três últimos campos do menu de actualização de perfil, que dizem respeito à alteração da palavra-chave do utilizador. A Figura 5.16 apresenta o modelo de tarefas relativo a esta funcionalidade.

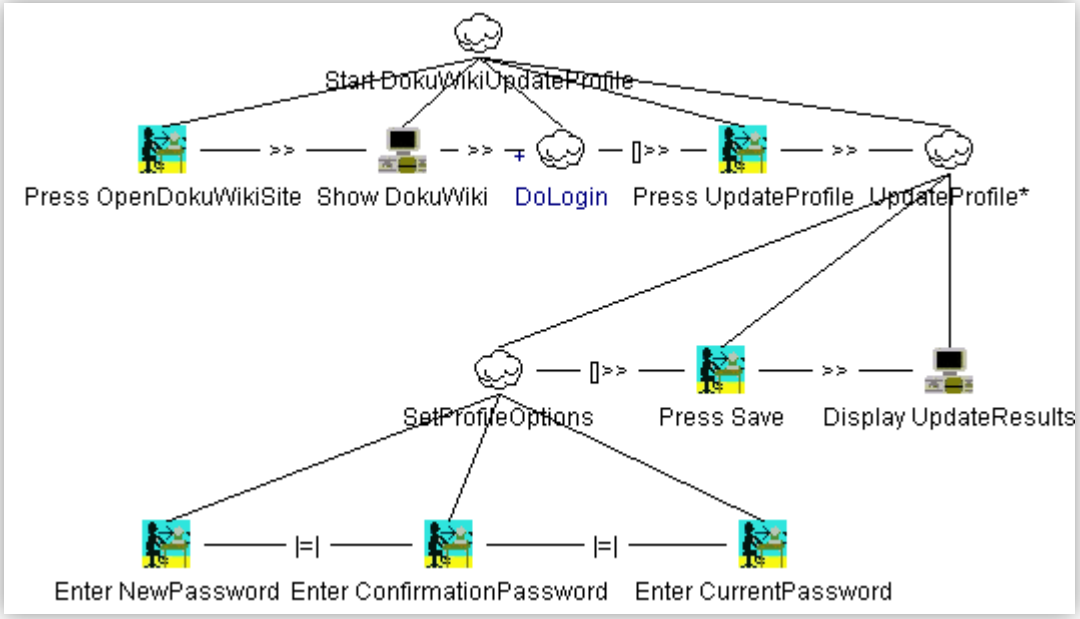


Figura 5.16: Modelo de tarefas da funcionalidade de alteração de palavra-chave da DokuWiki

No modelo apresentado, os detalhes da tarefa de login foram escondidos para facilitar a leitura do mesmo, uma vez que estes já foram analisados no caso anterior. Analisando este modelo verifica-se que a opção de acesso ao menu de actualização do perfil do utilizador apenas se encontra activa após o sucesso do login. Desta forma, foi necessária a adição de uma pré-condição que inibe a realização da acção enquanto o login não tiver sucesso. A tarefa "Enter NewPassword" diz respeito à inserção da nova palavra-chave, para a qual se pretende mudar. A tarefa "Enter Confirmationpassword" diz respeito à inserção da nova palavra-chave repetida. Por fim, a tarefa de "Enter CurrentPassword" diz respeito à inserção da palavra-chave actual do utilizador. Estas tarefas de inserção de palavras-chave podem ser realizadas por qualquer ordem.

Analisando agora a tarefa "updateProfile" verifica-se a existência de um ciclo semelhante ao analisado no caso do menu de login. No ciclo anterior, verificou-se que a informação era guardada de uma iteração para a outra. Desta forma, um utilizador que verificou o caso anterior poderia estar à espera que neste caso a informação também fosse gravada de uma iteração para a outra. Neste sentido foi utilizado o modelo que realiza o teste ao ciclo que se apresenta na Figura 5.17.

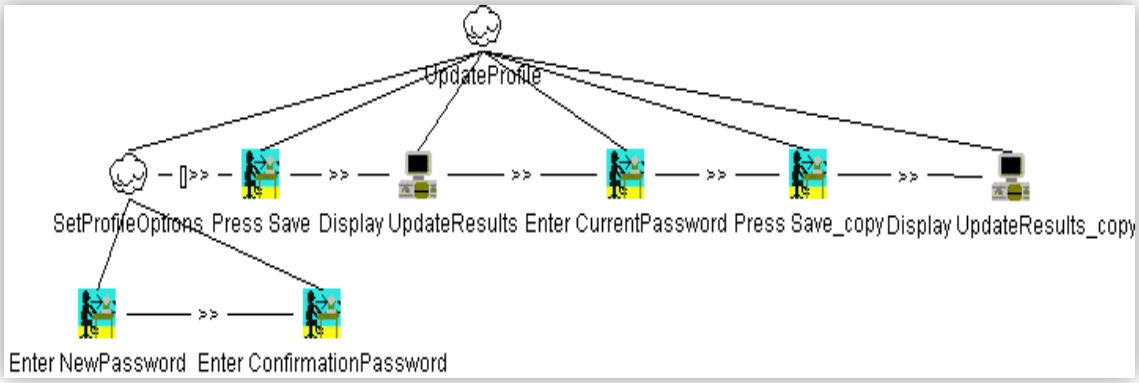


Figura 5.17: Modelo de tarefas de teste do ciclo da função de alteração de palavra-chave da DokuWiki

Ao executar os testes relativos a este modelo verificou-se que o seu resultado foi de insucesso. Isto acontece pois, o campo da palavra-chave actual não foi preenchido na primeira iteração e na segunda iteração apenas esse campo foi preenchido e os restantes não foram guardados. Este teste permitiu verificar que a interface deste menu, não gravou informação da primeira iteração para a segunda, ao contrário do que aconteceu com o teste do ciclo do menu de login. Esta situação controvérsia pode induzir o utilizador em erro, pelo que estes testes permitem ao profissional de testes, testar a resposta da interface face a estas situações.

5.4 Conversor Monetário XE

O conversor XE consiste num conversor monetário universal que permite converter uma quantia de um tipo de moeda para outro. Esta aplicação pode ser acedida em http://www.xe.com/ucc/po/. Neste caso de estudo foi analisada a funcionalidade de conversão deste conversor. A Figura 5.18 apresenta o menu de conversão desta aplicação.

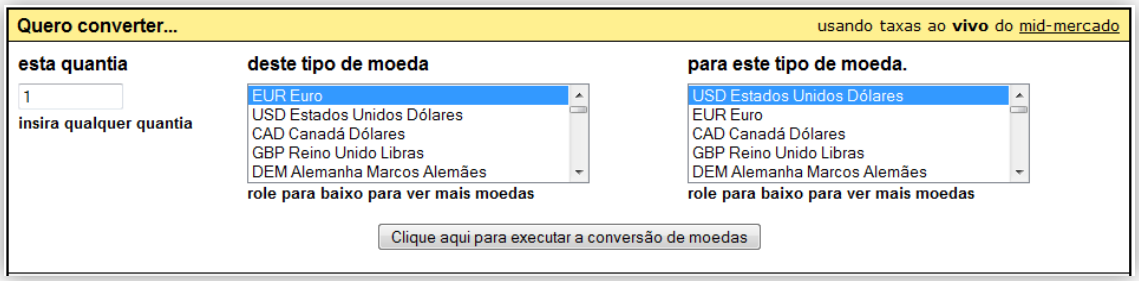


Figura 5.18: Menu de conversão do conversor XE

Para este menu foi construído o respectivo modelo de tarefas, que se apresenta na Figura 5.19.

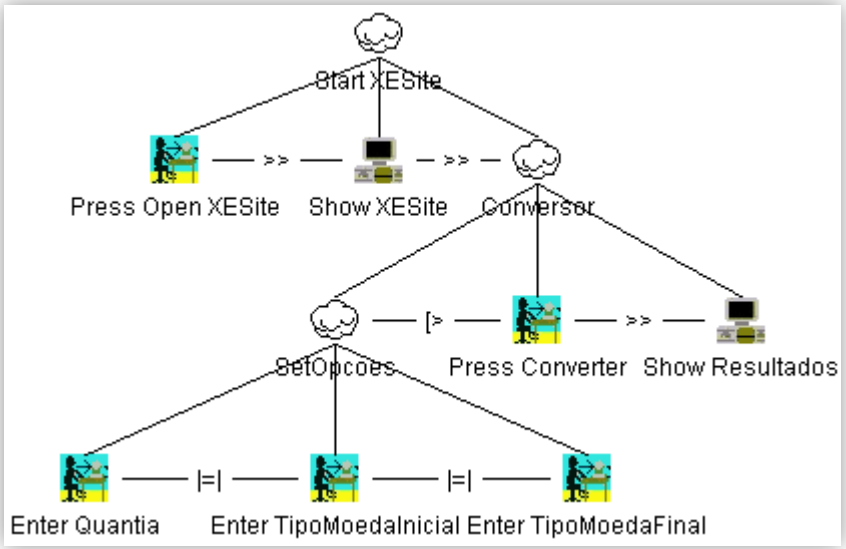


Figura 5.19: Modelo de tarefas do menu principal do conversor XE

Como se pode observar no modelo da figura anterior, após aceder ao conversor é possível seleccionar as opções de conversão pretendidas, tarefa "SetOpcoes". As tarefas de "Setopcoes" podem ser realizadas por qualquer ordem até que seja realizado o clique no botão que realiza a acção de conversão. A partir desse momento as tarefas anteriores deixam de estar activas e são apresentados os resultados da conversão.

Neste modelo de tarefas verifica-se a existência da situação "Desactivação de tarefas", identificada no Capítulo 3, na secção 3.3.3, que define o algoritmo de transformações. Para testar esta situação foram gerados os modelos de teste pela ferramenta CMT Tool. Desses modelos foi seleccionado o que corresponde à situação identificada, que se apresenta na Figura 5.20.

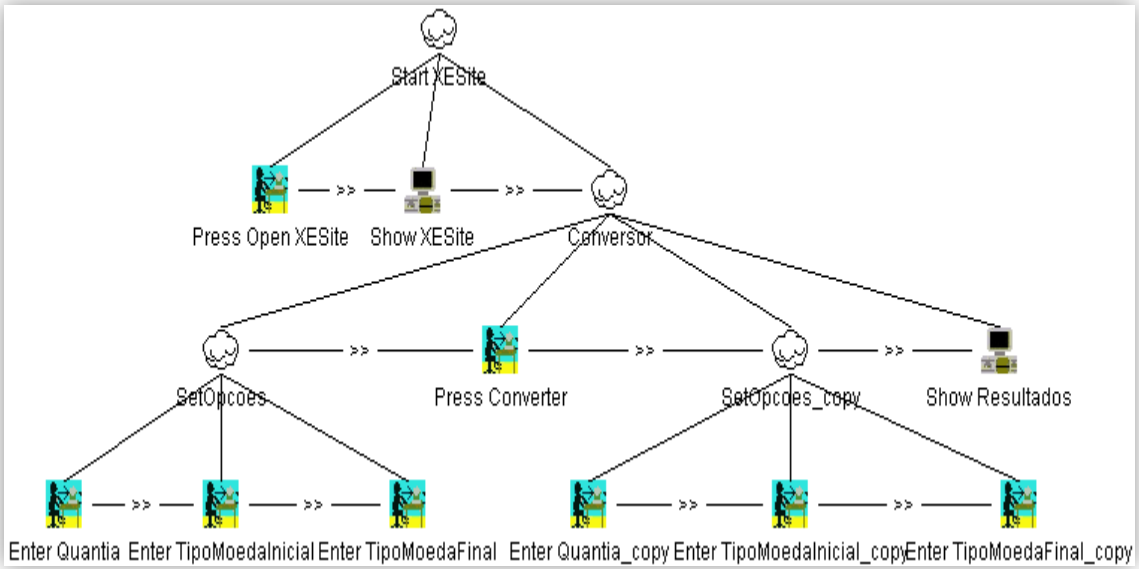


Figura 5.20: Modelo de teste de desactivação de tarefas do conversor XE

Como se pode observar figura anterior, a tarefa "SetOpcoes" é repetida depois da tarefa "Press Converter". Esta repetição permite assim testar se as opções de "SetOpcoes" deixam de estar activas após clique no botão. Para este modelo foi gerado o respectivo PTS no Teresa e o modelo Spec# no TOM.

Para testar a interface foram escolhidos os valores:

- **Quantia:** 2;
- **Tipo de moeda inicial:** USD Estados Unidos Dólares;
- **Tipo de moeda final:** EUR Euro.

Para execução dos testes, em Spec Explorer, as tarefas de "SetOpcoes" e "SetOpcoes_copy" foram definidas como retornando um valor booleano. Assim, caso o valor seja verdadeiro, significa que a tarefa foi executada com sucesso, caso seja falso, significa que ocorreu algum problema. Desta forma, para as tarefas de "SetOpcoes" foi definido no modelo que estas deveriam retornar o valor verdadeiro. Para as tarefas "SetOpcoes_copy" foi definido que estas deveriam retornar o valor falso, uma vez que estas não deverão estar activas após realização da tarefa "Press Converter". Ao executar os testes verificou-se o sucesso dos mesmos, o que significa que a aplicação está desenvolvida de acordo com o modelo de tarefas.

5.5 Conversor de Unidades

Neste caso de estudo foram analisadas duas aplicações de conversão de unidades: Online Converter e Unit Converter. Estas encontram-se disponíveis em

http://www.peters1.dk/webtools/conversion.php?sprog=en e http://www.digitaldutch.com/unitconverter/area.htm, respectivamente.

Estas aplicações permitem conversão de diferentes tipos de unidades. Neste caso de estudo apenas foi estudada a conversão de unidades relativas à área. Nesta conversão basta inserir um número numa unidade, por exemplo, metros quadrados, e são apresentadas as respectivas conversões para outras unidades, por exemplo, quilómetros quadrados.

A Figura 5.21 apresenta, em modelo de tarefas, a estratégia da aplicação Online Converter, sem entrar em muitos detalhes.

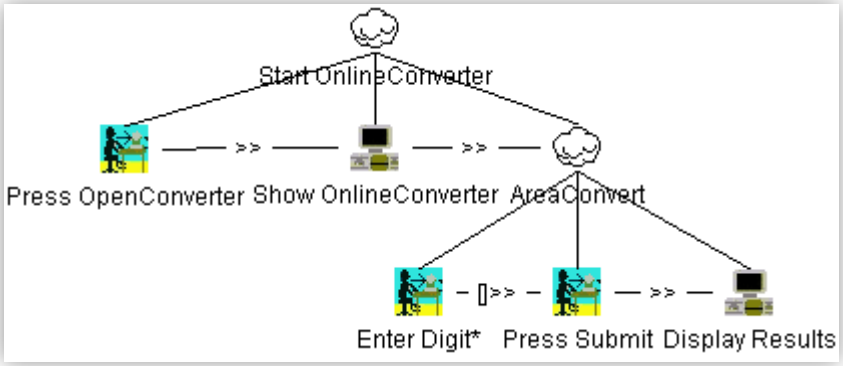


Figura 5.21: Modelo de tarefas da estratégia de área de Online Converter

Como se pode observar na Figura 5.21, a estratégia desta aplicação consiste em inserir os dígitos pretendidos e em seguida clicar no botão de submissão para o resultado da conversão ser apresentado.

No caso da aplicação Unit Converter, a estratégia utilizada já é diferente, sendo o respectivo modelo de tarefas apresentado na Figura 5.22.

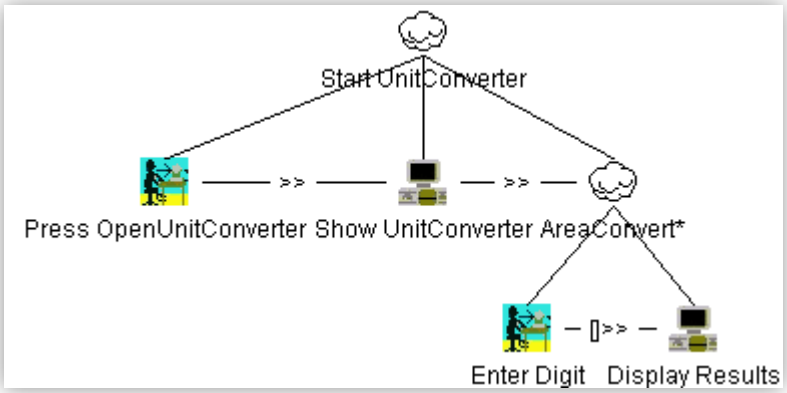


Figura 5.22: Modelo de tarefas da estratégia de Online Converter

Como se pode observar pela análise da Figura 5.22, a estratégia da aplicação Unit Converter é diferente da anterior. Neste caso, à medida que os dígitos vão sendo inseridos, os

resultados vão sendo apresentados automaticamente sem necessidade de pressionar nenhum botão de submissão.

Estas duas aplicações, embora apresentem o mesmo objectivo, possuem estratégias diferentes para o alcançar. Isto pode induzir o utilizador em erros ao transitar entre as duas aplicações. Estes erros correspondem ao tipo de erros enganos (*mistakes*) apresentado na secção 3.2, que corresponde à utilização do plano errado para alcance de um objectivo. Desta forma, torna-se necessário, nestes casos, testar a interface com os diversos planos de execução verificando a resposta da mesma a cada um deles.

Generalizando um pouco mais as estratégias apresentadas, é possível obter os modelos apresentados na Figura 5.23.

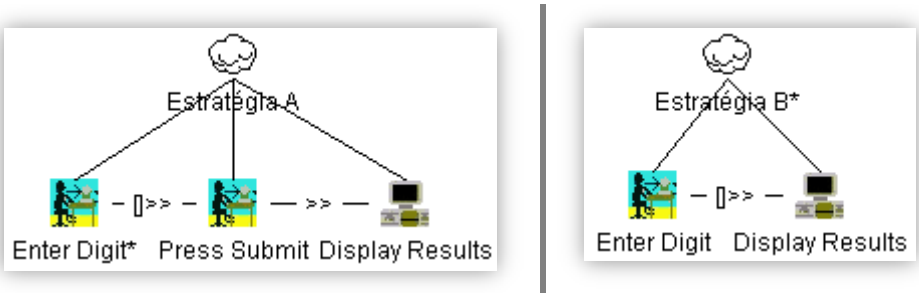


Figura 5.23: Estratégias genéricas dos conversores

Após definição destas estratégias de conversão, cada uma das aplicações foi testada com cada uma das estratégias.

Para realizar os testes foi utilizada a conversão de 123 centímetros quadrados para milímetros quadrados.

5.4.1 Online Converter

No caso do Online Converter, com a estratégia A, o resultado obtido foi de 12300 milímetros quadrados. Para simular a estratégia B, após cada inserção de dígito foi pressionado o botão de submissão simulando a apresentação dos resultados à medida que os dígitos são inseridos. Assim, ao inserir o dígito 1 o resultado apresentado foi de 100, ao inserir o 2 o resultado foi de 1200 e ao inserir o dígito 3 o resultado apresentado foi de 12300 milímetros quadrados. Como se pode verificar o resultado observado coincide com o obtido na estratégia A, sendo que, de certa forma, se pode afirmar que esta aplicação suporta a estratégia B.

Se, por exemplo, após clique no botão de submissão os dígitos introduzidos fossem apagados, os resultados obtidos já não seriam os mesmos. Para este caso após introdução do dígito 1 o resultado obtido seria de 100, após introdução do 2 o resultado seria 200 e após

introdução do 3 o resultado seria 300. Neste caso a estratégia B já não seria suportada, apresentando resultados finais diferentes dos esperados.

5.4.2 Unit Converter

Ao testar a aplicação Unit Converter com a estratégia B observam-se os seguintes resultados: após inserção do dígito 1 é apresentado o resultado 100, após inserção do dígito 2 é apresentado o resultado 1200 e após inserção do dígito 3 é apresentado o resultado 12300. Após inserção de todos os dígitos o resultado obtido de 12300 é o esperado, correspondendo à conversão de 123 centímetros quadrados para milímetros quadrados.

Para simular a estratégia A, basta introduzir todos os dígitos de uma vez e observar o resultado no final da inserção dos dígitos todos. Neste caso, o resultado obtido também é o de 12300, pelo que se pode concluir que a aplicação apresenta os resultados correctos com qualquer estratégia.

5.6 Conclusões

Para avaliar a solução proposta na abordagem foram seleccionadas diversas aplicações a utilizar como caso de estudo: reservas online Vip Hotels, pesquisa de imóveis da imobiliária Agimoura, DokuWiki, conversor monetário XE e conversor de unidades.

Nos primeiros quatro casos de estudo, foram analisados diversos modelos de teste gerados pela ferramenta CMT Tool, cada um deles permitindo avaliar as diferentes transformações identificadas na secção 3.3.3. Ao analisar os resultados obtidos nos testes realizados pode-se concluir que a abordagem definida permite testar as interfaces gráficas com o utilizador face a erros do tipo deslizamentos (*slips*), lapsos (*lapses*) que o utilizador comete ao utilizá-la. As decisões relativas ao sucesso e insucesso dos testes devem ser definidas pelo profissional de testes.

No último caso de estudo foram analisadas duas aplicações muito semelhantes com o mesmo objectivo mas com estratégias de execução de tarefas diferente. Para testar estas interfaces foram definidos dois modelos genéricos que apresentam cada uma das estratégias e as aplicações foram testadas com cada uma delas. Para aplicação das estratégias foi necessária a realização de algumas adaptações uma vez que, por vezes, essas estratégias pressupõem o clique em botões que a interface não possui ou o contrário. Pelos resultados observados concluiu-se que apesar de as estratégias serem diferentes os resultados finais foram os mesmos. Este caso de estudo permitiu testar a abordagem definida relativamente aos erros do tipo enganos (*mistakes*), concluindo-se que a mesma permite o teste destes erros. Mais uma vez,

cabe ao profissional de testes decidir sobre as condições de sucesso ou insucesso dos testes realizados.

Capítulo 6

Conclusões e Trabalho Futuro

Actualmente, a interacção entre os sistemas de software e o utilizador é cada vez mais realizada através de interfaces gráficas com o utilizador (GUIs). Desta forma, é necessário testar as GUIs para melhorar a qualidade das mesmas. Para enfrentar as dificuldades deste tipo de teste é possível utilizar teste baseado em modelos para automatizar a geração dos casos de teste. Contudo, a construção dos modelos acarreta também diversas dificuldades. Para superar esta barreira é possível abstrair o nível de construção dos modelos utilizando modelos de tarefas como base do teste baseado em modelos.

Existem diversas notações de modelação de tarefas com objectivos e características diversificadas. A notação CTT trata-se de uma notação bastante completa que facilita a construção dos modelos de tarefas devido à sua notação gráfica e à existência de ferramentas de suporte.

Existem também algumas ferramentas de teste baseado em modelos que permitem a automatização de alguns aspectos de teste de GUIs. Algumas dessas ferramentas não são adaptadas ao teste de GUIs, no entanto, existem algumas abordagens que permitem a sua utilização nesse contexto, como é o caso do Spec Explorer e da GUI Mapping Tool.

Uma das abordagens analisadas nesta dissertação propõe a utilização de modelos de tarefas em CTT como base do teste de GUIs baseado em modelos. Contudo, os modelos utilizados não contemplam situações de comportamentos errados do utilizador que ocorrem tipicamente na sua interacção com a GUI. Existem três tipos de erros típicos de comportamento do utilizador. Em termos de tarefas esses erros traduzem-se essencialmente na omissão ou troca de tarefas num modelo, ou então na utilização do modelo errado para atingir um objectivo.

Neste sentido, a abordagem definida nesta dissertação pretendeu contribuir para introdução desses erros típicos no modelo de tarefas original.

6.1 Satisfação dos Objectivos

A abordagem descrita nesta dissertação define uma estratégia que permite a utilização dos modelos de tarefas para teste de GUIs face a erros típicos de comportamento do utilizador. Para testar a GUI face a erros relativos a trocas ou omissões de tarefas foi definido um algoritmo de introdução de erros. Este algoritmo detecta padrões de sequências de tarefas e define um conjunto de transformações a realizar para cada padrão, permitindo obter modelos de teste que contemplem as situações de erro referidas. Para o último tipo de erros (utilização do modelo errado) a estratégia definida consiste na elaboração de modelos com estratégias diferentes para um mesmo objectivo. Posteriormente a GUI em teste seria testada com cada um desses planos para avaliar a resposta da mesma face a planos de execução errados.

Para validar a abordagem definida foi desenvolvida a ferramenta CMT Tool. Assim, após construção do modelo de tarefas original, são gerados diversos modelos de teste pela ferramenta. Para avaliar a qualidade dos modelos de teste foram analisados diversos casos de estudo.

Pela observação dos resultados obtidos nos casos de estudo verificou-se que a abordagem definida permite a definição de modelos de teste que permitem o teste da GUI face aos erros típicos do utilizador enunciados anteriormente. Estes modelos de teste, em cada caso de estudo, permitiram a detecção de erros ou falhas, a nível de usabilidade, na resposta da GUI face a estas falhas do utilizador. Verificou-se também que o sucesso dos testes do modelo de testes depende da especificação do profissional de testes, cabendo a este decidir qual a resposta da interface face a cada situação de teste.

De um modo geral, pode-se concluir que a abordagem apresentada se trata de uma abordagem válida e que os modelos de tarefas podem ser utilizados para testar erros típicos de comportamento do utilizador na sua interacção com a GUI. Desta forma verifica-se a satisfação dos objectivos propostos no início desta dissertação.

6.2 Trabalho Futuro

A abordagem apresentada nesta dissertação consiste numa abordagem modular na medida em que, nas suas etapas, recorre a diversas ferramentas. Nesta abordagem é possível distinguir cinco etapas distintas. Inicialmente, o modelo de tarefas é construído na ferramenta Teresa. De seguida, a ferramenta CMT Tool realiza transformações ao modelo original gerando diversos modelos de teste. Posteriormente, cada modelo de teste terá que ser tratado de forma independente pela ferramenta Teresa de forma a obter a respectiva máquina de estados associada (PTS). Cada modelo de teste e respectivo PTS, com a ajuda da ferramenta TOM, origina o respectivo oráculo de teste em Spec#. Esse oráculo é utilizado na ferramenta Spec Explorer para gerar e executar os testes.

A CMT Tool foi desenvolvida para prova do conceito permitindo automatizar a geração dos modelos de teste. Como trabalho futuro seria interessante analisar uma possível integração das diversas etapas. Esta integração poderia consistir na integração de algumas das ferramentas utilizadas em cada etapa ou de algumas das suas funcionalidades.

Analisando a abordagem verifica-se que é necessário recorrer à ferramenta Teresa duas vezes, uma no início para criação do modelo de tarefas original e outra na terceira etapa para geração das máquinas de estados para cada modelo transformado na CMT Tool. Uma possível melhoria seria integrar na ferramenta CMT Tool as funcionalidades da ferramenta Teresa relativamente à geração dos PTSs. Desta forma, após construção do modelo original, com a ferramenta CMT Tool seriam automaticamente gerados os modelos transformados e os respectivos PTSs.

Outra integração possível seria na passagem dos modelos da ferramenta CMT Tool e respectivos PTSs para a ferramenta TOM, correspondente à quarta etapa. Este processo poderia também ser integrado numa única etapa que, para além das funcionalidades referidas no ponto anterior, integrava as funcionalidades do TOM para geração dos oráculos a ser utilizados pela ferramenta Spec Explorer. Assim, após construção do modelo original, seriam automaticamente gerados os modelos de teste e respectivos PTSs e para cada um deles o respectivo oráculo em Spec#.

Quanto à ferramenta CMT Tool poderiam ser introduzidas melhorias relativamente ao *feedback* por esta apresentado. Esta ferramenta, na sua versão actual, apenas apresenta feedback relativamente ao sucesso ou insucesso das transformações realizadas. Seria interessante, de futuro, que esta produzisse também alguma informação quanto às transformações que ocorreram em cada um dos modelos de teste gerados. Esta informação poderia ir sendo incluída num ficheiro de texto à medida que os modelos de teste vão sendo criados pela ferramenta e ser apresentada no final da execução da mesma.

Referências

[Bel01] Fevzi Belli, Finite-State Testing and Analysis of Graphical User Interfaces, apresentado em Proceedings of the 12th International Symposium on Software Reliability Engineering (ISSRE'01), IEEE Computer Society, 2001.

[BLS05] M. Barnett, The Spec# programming system: an overview, *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices. International Workshop, CASSIS 2004. Revised Selected Papers (Lecture Notes in Computer Science Vol.3362),* vol. ISSU, pp. 49-69, 2005.

[BLV10] G.C. van der Veer, B.F. Lenting, and B.A.J. Bergevoet. *The Groupware Task Analysis - Homepage*. Disponível em http://www.cs.vu.nl/~gerrit/gta/. Acesso em Janeiro de 2010.

[BM09] Armin Beer and Stefan Mohacsi, Test-case Generation with IDATG, *Siemens IT Solution AK Software-Technologie 2 and Services,* 2009.

[BMS98] A. Beer, IDATG: An open tool for automated testing of interactive software, *TWENTY-SECOND ANNUAL INTERNATIONAL COMPUTER SOFTWARE & APPLICATIONS CONFERENCE - PROCEEDINGS,* pp. 470-475, 1998.

[CD10] James Clark and Steve DeRose. *XML Path Language (XPath)*. Disponível em http://www.w3.org/TR/xpath/. Acesso em Junho de 2010.

[CNM83] Stuart K. Card, Allen Newell, and Thomas P. Moran, *The Psychology of Human-Computer Interaction*, L. Erlbaum Associates Inc., 1983.

[Cor10] Microsoft Corporation. *UI Automation Fundamentals*. Disponível em http://msdn.microsoft.com/en-us/library/ms753107.aspx. Acesso em Junho de 2010.

[CSP09] J. C. Campos, J. L. Silva, and A. C. R. Paiva, Task Models in the model-based testing of user interfaces, *Software Testing, Verification and Reliability, Wiley InterScience,* 2009.

[Cun10] Marco A. M. Cunha, Padrões de Teste para Interfaces Gráficas, Tese de Mestrado, Departamento de engenharia informática, Faculdade de Engenharia da Universidade do Porto (FEUP), Porto, 2010.

[Dij72] Edsger W. Dijkstra, Chapter I: Notes on structured programming, em *Structured programming*, Academic Press Ltd., pp. 1-82, 1972.

[GRS03] Y. Gurevich, B. Rossman, and W. Schulte, Semantic essence of AsmL: Extended abstract, *FORMAL METHODS FOR COMPONENTS AND OBJECTS,* vol. 3188, pp. 240-259, 2003.

[Ham96] Fraser Hamilton, Predictive evaluation using task knowledge structures, *Conference on Human Factors in Computing Systems - Proceedings*, vol. ISSU, pp. 261-262, 1996.

[Hoc02] Lorin Hochstein. *GOMS*. Disponível em http://www.cs.umd.edu/class/fall2002/cmsc838s/tichi/printer/goms.html. Acesso em Janeiro de 2010.

[HSH90] H. Rex Hartson, Antonio C. Siochi, and D. Hix, The UAN: a user-oriented representation for direct manipulation interface designs, *ACM Trans. Inf. Syst.*, vol. 8, pp. 181-203, 1990.

[JJ91] H. Johnson and P. Johnson, TASK KNOWLEDGE STRUCTURES - PSYCHOLOGICAL BASIS AND INTEGRATION INTO SYSTEM-DESIGN, *ACTA PSYCHOLOGICA*, vol. 78, pp. 3-26, 1991.

[KK05] Tobias Klug and Jussi Kangasharju, Executable task models, apresentado em Proceedings of the 4th international workshop on Task models and diagrams, Gdansk, Poland, ACM, 2005.

[MBN03] Atif Memon, Ishan Banerjee, and Adithya Nagarajan, GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing, *Reverse Engineering - Working Conference Proceedings*, vol. ISSU, pp. 260-269, 2003.

[MC05] R. Mugridge and W. Cunningham, Fit for Developing Software: Framework for Integrated Tests, *Prentice Hall*, 2005.

[Mem07] A. M. Memon, An event-flow model of GUI-based applications for testing, *SOFTWARE TESTING VERIFICATION & RELIABILITY*, vol. CCCT, pp. 137-157, 2007.

[MPS01] Atif M. Memon, Martha E. Pollack, and Mary Lou Soffa, Hierarchical GUI Test Case Generation Using Automated Planning, *IEEE Trans. Softw. Eng.*, vol. 27, pp. 144-155, 2001.

[MPS02] Giulio Mori, Fabio Paternò, and Carmen Santoro, CTTE: Support for developing and analyzing task models for interactive system design, *IEEE Transactions on Software Engineering*, vol. 28, pp. 797-813, 2002.

[MPS04] Giulio Mori, Fabio Paternò, and Carmen Santoro, Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions, *IEEE Trans. Softw. Eng.*, vol. 30, pp. 507-520, 2004.

[MPS10] Giulio Mori, Fabio Paternò, and Carmen Santoro. *Multimodal Teresa: Transformation Environment for inteRactivE Systems representAtions*. Disponível em http://giove.isti.cnr.it/tools/TERESA/. Acesso em Janeiro de 2010.

[PFV07] A. C. R. Paiva, J. C. P. Faria, and R. F. A. M. Vidal, Towards the Integration of Visual and Formal Models for GUI Testing, Electronic Notes in Theoretical Computer Science, *Proceedings of the Third Workshop on Model Based Testing (MBT 2007)*, vol. 190, pp. 99-111, 2007.

[Pim07] Ana C. R. P. Pimenta, Automated Specification-Based Testing of Graphical User Interfaces, Tese de Doutoramento, Departamento de engenharia electrotécnica e de computadores, Faculdade de Engenharia da Universidade do Porto (FEUP), Porto, 2007.

[PMM10] F. Paterno, C. Mancini, and S. Meniconi. *The ConcurTaskTrees Environment - Home Page*. Disponível em http://giove.isti.cnr.it/tools/ctte/index.html. Acesso em Janeiro de 2010.

[PMM97] F. Paterno, C. Mancini, and S. Meniconi, ConcurTaskTrees: A diagrammatic notation for specifying task models, *INTERACT '97: Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction*, vol. 96, pp. 362-369, 1997.

[PVFT] Ana C. R. Paiva, Raul A. M. Vidal, J. P. Faria, and Nikolai Tillmann, Modeling and Testing Hierarchical GUIs, *12th International Workshop on Abstract State Machines*, 2005.

[PVST05] Ana C. R. Paiva, Raul A. M. Vidal, J. P. Faria, and Nikolai Tillmann, A model-to-implementation mapping tool for automated model-based GUI testing, *FORMAL METHODS AND SOFTWARE ENGINEERING, PROCEEDINGS*, vol. 3785, pp. 450-464, 2005.

[Rea90] J. Reason, *Human Error*. Cambridge University Press, Press Syndicate of the University of Cambridge, 1990.

[Rud10] Rebecca Rudigkeit. *GOMSED*. Disponível em http://www1.tu-darmstadt.de/fb/fb3/psy/kogpsy/indexgoms.htm. Acesso em Janeiro de 2010.

[SCP08] J. L. Silva, J. C. Campos, and A. C. R. Paiva, Model-based User Interface Testing With Spec Explorer and ConcurTaskTrees, *2nd International Workshop on Formal Methods for Interactive Systems (FMIS 2007) - Electronic Notes in Theoretical Computer Science*, vol. 208, pp. 77-93, 2008.

[SDMS98] Elton J. da Silva, Beatriz M. Daltrini, José G. V. Moreira, and Igor P. Soaresa, Uma Ferramenta Baseada em Tabelas UAN para Apoio à Modelagem de Interfaces, *IHC*, 1998.

[SS97] Richard K. Shehady and Daniel P. Siewiorek, A Method to Automate User Interface Testing Using Variable Finite State Machines, *Digest of Papers. Twenty-Seventh Annual International Symposium on Fault-Tolerant Computing (Cat. No.97CB36054)*, vol. ISSU, pp. 80-8, 1997.

[Tar04] Adriana Mihaela Tarta, Task Modeling in System Design, *STUDIA UNIV. BABES-BOLYAI, INFORMATICA*, vol. XLIX, Number 2, 2004.

[Tea02] CMMI Product Team, *Capability Maturity Model Integration for Software Engineering (CMMI-SW, V1.1)*. Pittsburg, Carnegie Mellon, Software Engeneering Institute, 2002.

[Uni09] Carnegie Mellon University. *CogTool*. Disponível em http://cogtool.hcii.cs.cmu.edu/. Acesso em Janeiro de 2010.

[VCGSTN08] M. Veanes, C. Campbell, W. Grieskamp, W. Schulte, N. Tillmann, and L. Nachmanson, Model-based testing of object-oriented reactive systems with spec explorer, em *Formal methods and testing: an outcome of the FORTEST network*, Springer-Verlag, pp. 39-76, 2008.

[VWC02] Gerrit C. van der Veer, Martijn van Welie, and Cristina Chisalita, Introduction to Groupware Task Analysis, apresentado em Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design, INFOREC Publishing House Bucharest, 2002.

[WA00] Lee White and Husain Almezen, Generating Test Cases for GUI Responsibilities Using Complete Interaction Sequences, apresentado em Proceedings of the 11th International Symposium on Software Reliability Engineering, IEEE Computer Society, 2000.

[Wil10] Peter Wild. *About TKS - Website*. Disponível em http://www.cs.bath.ac.uk/~hci/TKS/aboutTKS.html. Acesso em Janeiro de 2010.

[WV03] M. Van Welie and G. C. Van Der Veer, Groupware task analysis, em *HANDBOOK OF COGNITIVE TASK DESIGN*, Ed. E. Hollnagel, Lawrence Erlbaum Associates, New Jersey, US, pp. 447-476, 2003.

[WVE98] M. Van Welie, G. C. van der Veer, and A. Eliens, An ontology for task world models, *DESIGN, SPECIFICATION AND VERIFICATION OF INTERACTIVE SYSTEMS'98, Springer-Verlag, Wien, Austria,* pp. 57-70, 1998.

[Xie06] Qing Xie, Developing cost-effective model-based techniques for GUI testing, *28th International Conference on Software Engineering Proceedings,* vol. ISSU, pp. 997-1000, 2006.