FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Automação de testes funcionais a aplicações Java Swing

Álvaro Gabriel Machado Caldas

VERSÃO FINAL

Relatório de Projecto
Mestrado Integrado em Engenharia Informática e Computação

Orientador: Ana Cristina Ramada Paiva Pimenta (Professor Auxiliar)

Julho de 2009

Automação de testes funcionais a aplicações Java Swing

Álvaro Gabriel Machado Caldas

Relatório de Projecto
Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: António Augusto de Sousa (Professor associado)

_____

Arguente: José Francisco Creissac Freitas Campos (Professor Auxiliar)

Vogal: Ana Cristina Ramada Paiva Pimenta (Professor Auxiliar)

24 de Julho de 2009

Resumo

Este projecto enquadra-se na área de teste de software apontando-se como objectivo a construção de uma aplicação de automação de testes, direccionada para aplicações construídas em Java Swing. Teste é um processo importante no ciclo de vida do software, descuidos na detecção de erros podem trazer consequências muito graves. Negócios, e até mesmo vidas, podem correr risco quando uma organização não é capaz de testar adequadamente software em busca de bugs e/ou problemas de performance, nem determinar se o programa satisfaz requisitos de negócio e as necessidades de utilizadores finais. No entanto, teste de software poderá ser algo extremamente maçador e cansativo, sendo necessário ter especial atenção ao contratar pessoal para trabalhar nesta área. Metodologia e persistência são características necessárias para este tipo de funções. No entanto, todo este trabalho pode ser, não só facilitado, como também ajudado com o auxílio de ferramentas desenvolvidas para testes. Exemplo disso são ferramentas de automação de testes que permitirão, por exemplo, poupar trabalho aos testadores, dando a hipótese de efectuar testes de uma forma automática, especialmente testes que tenham de ser repetidos inúmeras vezes, dando a oportunidade ao testador de se libertar deste trabalho maçador e prolongado para que possa preocupar-se mais com a construção de novos testes.

Era imposta a construção de um motor de gravação e execução de testes, ou seja, a ferramenta teria de gravar interacções com uma interface gráfica num script próprio, para que depois essas interacções pudessem ser, mais tarde, reproduzidas automaticamente, fornecendo um log sobre possíveis problemas existentes durante a reprodução. Interoperabilidade, devido à integração futura com uma aplicação desenvolvida em C# era pedido, bem como uma gravação selectiva de eventos, ou seja, gravar apenas interacções que produzam alguma alteração na interface. A minimização de problemas inerentes a este tipo de teste automatizado a interfaces gráficas também era um objectivo. Problemas de sincronia de acções, bem como a fragilidade que este tipo de aplicações tem a alterações à interface gráfica, deveriam ser minimizados. Tanto na gravação como reprodução, é usado reflexão. A gravação de eventos é feita com a leitura a uma fila de eventos e com eventos de componentes. Na reprodução, é usado o tempo de CPU utilizado por parte da aplicação a ser testada, na tentativa de minimizar o problema de sincronia entre acções presente nas ferramentas de automação. Durante todo o processo de implementação desde estruturas de dados usadas até à linguagem de script utilizada foi necessário despender de alguma atenção para que a integração futura fosse o mais simples possível.

De um modo geral, todos os testes à aplicação desenvolvida foram concluídos com sucesso. Em todas as aplicações testadas a sincronia de acções é conseguida, mesmo quando

existiam tempos de carregamento de dados. A linguagem de script criada para escrita de testes é bastante funcional sendo possível escrever um script de teste facilmente. Os objectivos foram todos cumpridos. A aplicação comporta-se portanto como esperado e pronta para ser efectuada a integração..

Palavras-chave: Reflexão, Gravação/Reprodução, Webservices, fila de eventos, Teste Caixa Preta,Testes de Software, Automação de Testes

Abstract

This project falls within the area of software testing with the objective of building an application for automated testing, targeted for applications built in Java Swing. Testing is an important process in the software lifecycle, carelessness in the detection of errors may have very serious consequences. Business, and even lives, may be risk when an organization is unable to adequately test software in search of bugs and / or performance problems, or whether the program meets the requirements and business needs of end users. However, testing software can be extremely tiring and boring, special attention to hiring staff to work in this area is needed. Methodology and persistence are required characteristics for this type of functions. However, all this work may be not only easier but also helped with the aid of tools developed for testing. For example, tools for testing automation will save time to testers, giving a chance to do the tests in an automated way, particularly tests that have to be repeated many times, giving the tester the opportunity to be released from this boring and long work and let him to give more attention to the construction of new tests. It was imposed the construction of an engine for recording and execution of tests, ie, the tool would record interactions with a graphical interface in a script, so that after these interactions could be, later, played automatically, providing a log on possible problems during playback. Interoperability, because of a future integration with an application developed in C # was required, and so as a selective recording of events by recording only interactions that produce a change in the interface. The minimization of problems inherent in this type of automated testing to graphical interfaces was also a goal. In both phases, recording and playback, reflection is used. The recording of events is done by reading a queue of events and events of components. In the replay phase, is used for the CPU time used by the application to be tested in an attempt to minimize the problem of synchronization between the actions in the tools of automation. Throughout the implementation process, from data structures used to the script language was necessary to spend some attention to the future integration and try to make it as simple as possible.

In general, all tests to the tool implemented were successfully completed. In all applications tested the synchrony of action is achieved, even when there were times of loading. The language created for writing the script of tests is very functional and can a script can be written easily. The goals were all met. The application therefore behaves as expected and is ready for the integration.

Keywords: Reflection, Recording / Playback, Webservices, Eventqueue, Black Box Testing, Software Testing, Automation Testing

Agradecimentos

É com muita satisfação que expresso aqui o mais profundo agradecimento a todos aqueles que tornaram a realização deste trabalho possível.

Gostaria antes de mais de agradecer à Professora Ana Cristina Ramada Paiva Pimenta, orientadora desta tese, pelo apoio, incentivo e disponibilidade demonstrada em todas as fases que levaram à concretização deste trabalho.

Gostaria ainda de agradecer:

Ao José Pedro Tavares, orientador da tese na empresa pelos comentários, sugestões, incentivo feitos.

À empresa Telbit, pela oportunidade que me deu de mostrar o meu trabalho com este projecto.

Ao Vítor Duarte, Sérgio Silva, Renato Sousa, David castanheira, Tiago Lages e Antero Silva amigos que me apoiaram e ajudaram psicologicamente em todos os momentos desta fase.

A todos os meus familiares que estiveram presentes nos momentos menos bons para me apoiar e incentivar a continuar.

Aos meus país, Gabriel e Maria do Rosário Caldas, pelo apoio nesta fase e também pela sólida formação dada na minha juventude, que me fez ser aquilo que hoje sou, meus eternos agradecimentos.

Álvaro Caldas

Índice

Lista de Figuras

Abreviaturas e Símbolos

CPU	Central Processing Unit
JVM	Java Virtual Machine
API	Application Programming Interface
JAXB	Java Architecture for XML Binding
GUI	Graphical User Interface
AUT	Application Under Test
WPF	Windows Presentation Foundation
WinForms	Windows Forms
XML	eXtensible Markup Language
SOAP	Simple Object Access Protocol
WSDL	Web Services Description Language
UDDI	Universal Description, Discovery and Integration
DSL	Domain Specific Language

1 Introdução

Esta tese é o resultado de um projecto, que decorreu ao longo de dezasseis semanas, com o objectivo de produzir uma aplicação que permitirá automatizar testes a aplicações produzidas em Java Swing.

No presente relatório poderemos encontrar toda a informação necessária para a contextualização do problema, a motivação, toda a investigação realizada, a abordagem optada, bem como, os resultados alcançados e a sua avaliação.

1.1 Contexto

Os "Graphical User interfaces" (GUIs) são uma parte fulcral no desenvolvimento de software podendo representar até 60% do código produzido de uma aplicação.[1]

Dada a actual importância dos GUIs no desenvolvimento de Software, não só de uso pessoal mas também em áreas críticas em termos de segurança, é imperativo assegurar o seu correcto funcionamento.

Diversos tipos de teste podem ser feitos a um software em desenvolvimento, com objectivos muito distintos como, por exemplo, testes de performance, de carga ou de funcionalidades.

Os testes funcionais são testes derivados da especificação do software ou componente, sendo aqueles onde o testador se preocupa com a funcionalidade do sistema, abstraindo-se da sua implementação. A necessidade de garantir que todas as especificações foram cumpridas, especificações essas, que foram acordadas com o cliente na fase de análise de requisitos, fazem deste tipo de testes os mais frequentes [14] sendo nessa área que se insere este trabalho.

A execução dos testes poderá ser feita tanto de forma manual como automatizada, consistindo sempre na reprodução de testes definidos previamente.

Estas formas de teste apresentam vantagens e desvantagens. Uma das suas grandes diferenças é o tempo que demorará a realizar um teste manual que é consideravelmente maior do que na sua versão automatizada.

Os testes funcionais devem ser automatizados quando são muito repetitivos e exigem um esforço considerável de tempo quando realizados manualmente[17].

Este tipo de testes, apesar da "aparente" facilidade inicial em que o utilizador só terá de efectuar interacções com a interface, rapidamente se mostram bastante complexos com o aumento da complexidade da interface.

Imaginar todas as formas em que um GUI poderá falhar, de maneira a poderem ser desenhados os testes no sentido de detectar algum problema, apresenta-se como maior desafio. Para além disso, a rotina inerente à criação destes testes, bem como o tempo que poderá demorar o processo, poderão fazer diminuir a atenção do testador, fazendo-o ignorar certos aspectos/oportunidades, que se evidenciem, e que poderiam levar a detecção de algum erro. É de facto um dos grandes problemas deste tipo de aplicações ou testes, estes estarem limitadas à imaginação e criatividade do testador para a geração de interacções com o GUI.[1]

1.1.1 Telbit – tecnologias da informação

Telbit foi fundada em 1998 como uma spin-off da PT Inovação (empresa de I & D da Portugal Telecom) em Aveiro, Portugal. Oferece serviços e soluções que englobam o ciclo de desenvolvimento de software: desenvolvimento (core-business), testes e suporte&manutenção.

O mercado das telecomunicações é o principal contratante.

A investigação e desenvolvimento é considerado um condutor fundamental para o crescimento sustentado e maior oportunidade de conquista estando neste momento a apostar na área de testes de software.

1.1.2 TeStudio

Para se afirmar no mercado, a Telbit conta com o TeStudio, uma ferramenta de automação do processo testes de software. Permite a colaboração entre testadores, programadores e gestores trazendo reutilização, automação e organização em todos os testes projecto.

Este software apresenta como principais recursos:

- Planeamento de testes que vão de encontro com os requisitos pretendidos, execução de testes, gestão de defeitos, análise do estado do projecto numa interface única;

- Controlo de acesso com base no papel do utilizador;

- Abstracção do modelo de base de dados, permitindo o desenho de testes sem ter conhecimento da modelação dos dados permitindo mantê-los actualizados quando existem mudanças na base de dados;

- Manual e testes automatizados de gestão;

- Fornece documentação, gráficos e relatórios de geração em qualquer ponto do processo de realização do teste;

- Possibilita um processo de teste consistente e determinista fornecendo um repositório central reutilizável para todos os testes activos.

Este projecto vem no encalço desta aposta, pretendendo adicionar o suporte de testes a aplicação Java Swing ao TeStudio, que no momento suporta aplicação desenvolvidas em WinForms e WPF.

1.2 Projecto

Com base na importância que representam os GUIs na actualidade segue a proposta de um projecto com a intenção de automatizar testes a aplicações Java Swing.

A proposta teve como objectivo o desenvolvimento de uma aplicação gravação/reprodução de eventos gerados pelo utilizador no GUI com particularidades já presentes em algumas aplicações já existentes.

Com base na especificação de requisitos e procurando cumprir todos os objectivos, foi criada um aplicação autónoma, não dependendo de qualquer outra aplicação já existente. No entanto, a sua implementação foi feita sempre pensando na interoperabilidade, para que a ferramenta produzida por este trabalho pudesse vir a integrar uma aplicação de testes já existente, o TeStudio. É dentro desta aplicação que o resultado da reprodução das acções é interpretado pelo testador.

1.3 Motivação e Objectivos

O processo de teste envolve a execução de uma aplicação, a aplicação a ser testada (AUT), com os dados de entrada do teste, para depois se analisar as suas saídas e o seu comportamento, para se verificar se a aplicação teve o comportamento esperado.

O custo da eliminação de um erro no software é muito maior durante a fase final de um projecto, do que durante o seu desenvolvimento [11]. Dessa forma, o processo de testes é uma etapa critica no desenvolvimento de software. Apesar de durante todo o ciclo de vida de desenvolvimento de uma aplicação se usem técnicas, procedimentos e ferramentas para que

erros não sejam introduzidos no produto, nunca são eliminadas todas as possibilidades de erro. Assim, o processo de teste de software continua a mostrar-se extremamente útil e necessário para a eliminação de erros.

Este trabalho que tem como objectivos:

- Especificação e implementação de um motor de gravação e execução de testes a aplicações desenvolvidas em Java Swing;
- Permitir a execução automática de testes a aplicações via instrumentação do GUI;
- Facilitar a execução de testes de regressão minimizando os custos de desenho dos testes;
- Maximização da "tolerância" dos testes a alterações no GUI entre várias versões da mesma aplicação;
- Maximização da "tolerância" dos testes a alterações nos dados tratados pela aplicação;
    - Gravação selectiva de eventos, filtrando eventos que não produzem qualquer efeito/alteração no GUI;
- Interoperabilidade para facilitar a integração com o TeStudio.

1.4 Estrutura da Dissertação

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

- **Capítulo** 2 - "Estado da arte" faz uma descrição da área de testes de software e do seu enquadramento, descrevendo também tipos e técnicas de testes. Aprofunda a automação de testes e apresenta algumas aplicações com que se interagiu.

- **Capítulo** 3 - "Descrição e abordagem ao problema" é dada uma descrição do problema que foi apresentado e uma abordagem para solucionar o mesmo, são apontadas as opções tomadas, no que diz respeito à linguagem de script, integração, gravação e reprodução, reutilização de casos de teste e, por fim, as tecnologias utilizadas.

- **Capítulo** 4 - "Implementação" Neste capítulo descreve-se como a aplicação foi desenhada e implementada, as formas de interagir com ela e referência todas as decisões relevantes tomadas durante o projecto. Uma avaliação do trabalho feito bem como os testes realizados estão também aqui documentados.

- **Capítulo** 5, "Conclusões e Trabalho Futuro" faz uma avaliação de todo o trabalho realizado e apresenta as conclusões necessárias. São também referidos alguns pontos de melhoramento para um futuro trabalho que possa ser feito

2 Estado da arte

O teste é uma actividade do ciclo de vida de um projecto de software que visa medir e melhorar a qualidade da aplicação a ser testada (AUT). É um processo de interacção com a aplicação onde se procura detectar defeitos, que comprometam a estabilidade da aplicação, para poderem ser corrigidos antes da entrega final do produto.

Destaca-se como sendo o método principal para assegurar que o design e implementação da aplicação estão conforme o especificado. Este processo é no entanto dispendioso (ver Anexo B:), e muito intensivo no que diz respeito a trabalho, não acrescentando nada de novo às funcionalidades da aplicação.[9]

2.1 Verificação e Validação

Teste de software é um dos elementos de um tema mais amplo denominado por Verificação e Validação, onde[12][16]:

Verificação - Está ligada a conjuntos de actividades que procuram garantir que o software implementa correctamente uma função específica;

Validação - refere-se ao conjunto de actividades que procuram garantir que o software que foi construído atende às exigências do cliente.

Este tema divide-se em duas técnicas, sendo elas[12]:

Inspecções de software (V & V estática)

– Documentação e código fonte da aplicação são analisados;

– Ferramentas de depuração para auxílio podem ser utilizadas.

Testes de software (V & V dinâmica)

– É feita uma execução à aplicação ou ao protótipo;

– Casos de testes são executados observando-se o comportamento da aplicação no sentido de encontrar os seus defeitos.

Esquematicamente temos:

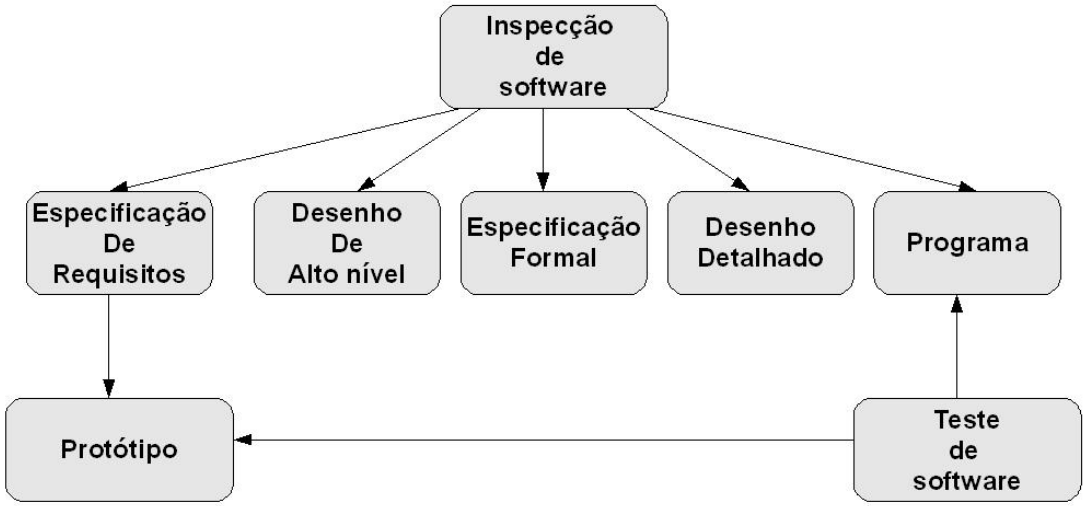

Figura 1: Verificação e validação estática e dinâmica[12]

2.2 Casos de teste

O processo de teste de software implica também a elaboração de casos de teste para que a aplicação possa ser de facto testada. Um caso de teste típico é então definido por [13]:

- Dados de entrada
    - Pré-Condições – São condições necessárias para o teste ser iniciado;
    - Inputs – Passos a seguir para a realização do teste.
- Dados de saída
    - Outputs esperados – Resultados esperados da aplicação através da execução dos inputs;
    - Pós-Condições – Representará o estado final do sistema.

Para além da informação descrita acima, o caso de teste pode incluir também: um identificador, uma descrição, dependências de outros casos de teste, o procedimento do teste e o item que de facto está a ser testado.

Findada a elaboração do caso de teste, o testador irá seguir as especificações que nele se encontram, confirmando que as pré-condições estão cumpridas e efectuando os passos definidos. No final, os resultados obtidos são comparados com os resultados esperados da aplicação.

A capacidade de encontrar falhas, especialmente as de grande risco, o custo (em termos do seu desenvolvimento,rapidez de execução e análise de resultados), a sua facilidade de manutenção e a capacidade de testar vários aspectos da aplicação são factores que determinam a mais valia do teste.[16]

O processo de geração de testes poderá ser automatizado reduzindo os custos do seu desenvolvimento. No entanto, a capacidade de detectar defeitos é por norma menor apesar de, num panorama geral, o teste ser mais bem sucedido, já que desta forma o número de testes gerados é maior. Analogamente, a execução dos testes pode ser também automatizada, o que obriga à codificação dos testes numa linguagem executável. Esta codificação irá provocar um aumento de custos, no custo de desenvolvimento dos testes que será compensada pela, quase total, eliminação dos custos de re-execução extremamente importante nos testes de regressão.

É nesta parte de automação da execução de testes que o trabalho a desenvolver está inserido. A aplicação gravação/reprodução a desenvolver, recolhe/grava as interacções do utilizador com a interface gráfica para depois reproduzir essas interacções automaticamente.

2.3 Técnicas de teste de Software

Actualmente existem diversas formas de se testar um software. No entanto, existem técnicas predominantes, que sempre foram usadas sobre linguagens estruturadas, e ainda hoje mostram a sua mais valia quando aplicadas em sistemas desenvolvidos em linguagens orientadas ao objecto. Apesar dos paradigmas do desenvolvimento não permanecerem sempre os mesmos o objectivo destas técnicas mantém-se inalterável: Encontrar falhas no software.

Duas técnicas habituais no processo de teste de software podem ser mencionadas [14]:

- Caixa preta (conhecida também por “Functional Testing”), define-se por uma abordagem que ignora o mecanismo interno do sistema ou componente, para se focar apenas nos outputs gerados através dos inputs;
- Caixa branca (conhecida também por “Structural testing” ou “Glass Box Testing”), ao contrário da abordagem anterior, os mecanismos internos do sistema/componente são uma característica presente.

Estas duas abordagens completam-se e são as duas importantes para efectuar testes completos [9].

2.3.1 Caixa preta

Neste tipo de teste não existe acesso ao código fonte da aplicação. O código é considerado como sendo uma "caixa preta" em que não se poderá ver o conteúdo.

O testador sabe apenas que pode introduzir informações na "caixa preta" e irá receber algo como resposta. Baseado no conhecimento dos requisitos o testador sabe o que esperar como resposta e testa se de facto o que a "caixa preta" enviou como resposta corresponde ao esperado.

Gravação/Reprodução

Sendo que o processo de testes poderá ser algo penoso, a automação deste processo é de facto uma mais valia. As aplicações de gravação/reprodução são frequentemente usadas para facilitar os testes, permitindo, capturar acções do utilizador com a interface da AUT. Mais tarde, estas acções são reproduzidas automaticamente em diferentes, ou até, na mesma versão da aplicação onde foram recolhidas.

Ferramentas de gravação/reprodução são muito usuais para testes a GUIs. Abbot/Costello é exemplo de uma dessas ferramentas desenvolvida sob licença GPL, existindo versões comerciais, como por exemplo, o WinRunner produzido comercialmente pela Mercury Interactive Corp.

Com recurso a ferramentas deste tipo, o testador interage com o interface para gerar eventos de rato e teclado. A ferramenta intercepta os eventos gravando-os, habitualmente, num script para mais tarde serem reproduzidos.

Levantam-se grandes problemas neste tipo de ferramentas:

- este é um processo penoso, exigindo muito trabalho e esforço por parte do testador e baseando-se muito na sua capacidade de gerar interacções com o GUI que sejam realmente relevantes[1], sendo que um gerador de casos de teste automático poderá fornecer um suporte, aumentando, no entanto, o trabalho do programador que terá de codificar todos as pontos de decisão do GUI. De qualquer das formas, esta estratégia poderá deixar passar facilmente pontos muito importantes durante a interacção com o GUI;
- Outro dos problemas será a edição dos scripts de testes que poderá ser extremamente difícil. Se o sistema mudar, pode ser necessário alterar o script ou mesmo proceder a uma nova gravação desse mesmo script;
- O facto de esta técnica não ser baseada numa especificação [8], acrescenta um problema adicional. No processo de gravação os scripts gravados serão usados mais tarde para efectuar principalmente testes de regressão, ou seja, serão usados para repetir testes ao programa depois de serem feitas alterações ao código final, não existindo nenhuma especificação sobre qual será considerado o correcto comportamento do sistema perante as interacções. Uma vez que os testes são escolhidos com muita especificidade é impedida a sua utilização em diferentes cenários de teste.

Baseado em modelos

Este tipo de teste, baseado em modelos, é uma abordagem "caixa preta" para geração de testes de software a partir de modelos da aplicação[17], onde casos de testes são executados para avaliar a correspondência entre o modelo e a aplicação. Para isto, a especificação da AUT necessita de estar formalmente ou semi-formalmente descrita por meio de um modelo, de modo a caracterizar com exactidão o seu comportamento.

Relacionado com este tipo de testes estão as seguintes actividades:

- **Construir o modelo:** o modelo formal ou semi-formal é construído a partir dos requisitos da aplicação;
- **Gerar inputs:** os inputs do teste são gerados partindo do modelo. Estes inputs são passos que servirão para interagir com a AUT.
- **Gerar outputs esperados:** os outputs esperados do teste são gerados partindo do modelo formal e servirão de referência para comportamento esperado do sistema.
- **Executar testes:** a aplicação é executada com os inputs gerados, gerando outputs;
- **Comparar outputs reais com os esperados:** os outputs da aplicação que está a ser testada são comparados com outputs esperados gerados a partir do modelo.
- **Decidir acções futuras:** Após o resultado final, analisar a necessidade de gerar mais testes ou para-los, fazer uma estimativa sobre a fiabilidade (qualidade) do software ou até mesmo ponderar uma mudança no modelo.

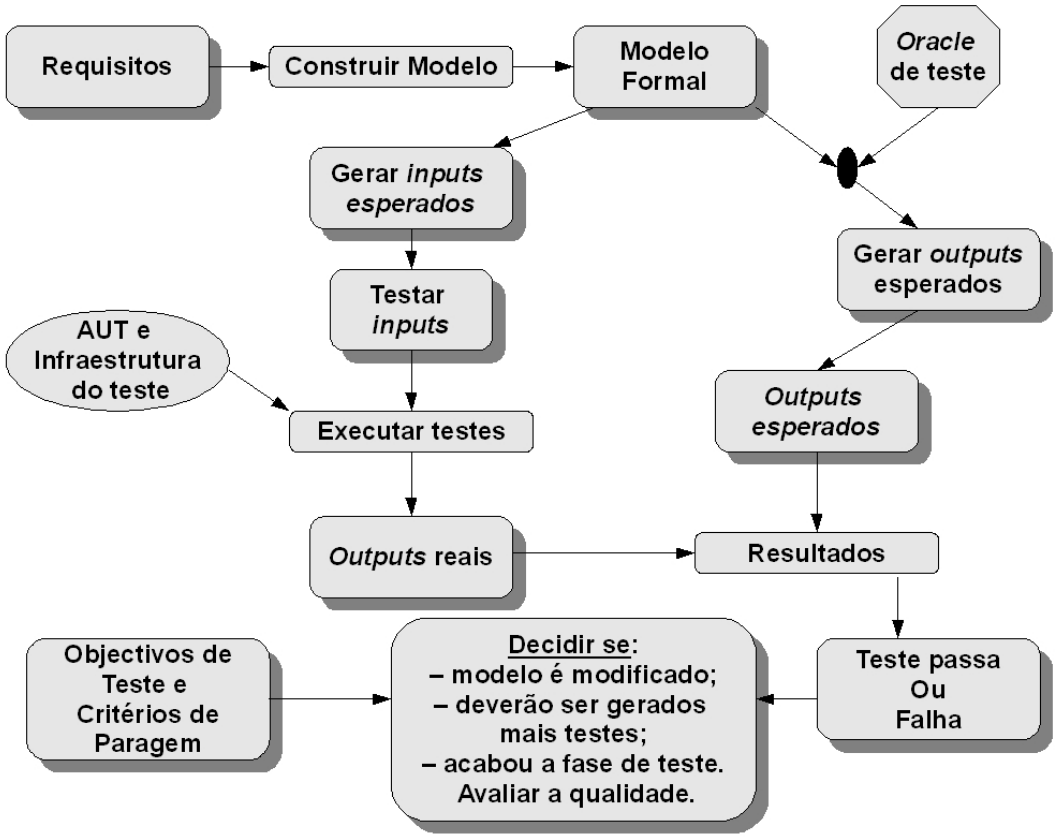

Figura 2: Passos do processo de teste baseado em modelos[17]

O processo de testes baseado em modelos, inicia-se com os requisitos. O que significa que o processo de teste pode ser iniciado, assim que os requisitos da aplicação estiverem definidos. Com os requisitos definidos, o próximo passo é a construção de um modelo que retrate na íntegra o comportamento requisitado. A partir deste modelo construído, poderemos ter um melhor entendimento da aplicação e obter casos de teste, sendo estes, por norma, obtidos de forma automática. A especificação dos casos de teste inclui inputs e outputs esperados. Com os inputs, executaremos a aplicação para observar o comportamento da aplicação, comparando, de seguida, os outputs reais com os esperados. Tal comparação servirá para avaliar a presença de defeitos na aplicação e os dados obtidos servirão, por exemplo, para decidir se o teste deverá terminar e o sistema ser entregue.

2.3.2 Caixa branca

Utilizando técnicas “caixa branca”, podem ser elaborados casos de teste que:

- Testem caminhos independentes dentro de um módulo ou unidade;
- Testem decisões lógicas em ambos os lados (verdadeiro e falso);
- Executem ciclos nos seus limites e dentro de seus limites operacionais;
- Testem estruturas de dados internas para garantir a sua validade.

Devidamente planeadas com combinações explícitas de input/output, "caixa branca" é uma técnica de Verificação e Validação controlada. Ao executar um teste as linhas de código que estarão a ser executadas são bem definidas, sendo do conhecimento do testador a resposta que o sistema deverá ter.

Se a resposta certa não for obtida, a falha é revelada pelo teste e, uma vez que, as linhas de código que estão a ser testadas são conhecidas, a localização do erro torna-se bastante simples, assim sendo, a eliminação de defeitos em testes unitários torna-se menos dispendiosa do que em fases posteriores no ciclo de desenvolvimento. Em fases posteriores que envolvam testes "caixa preta", um caso de teste deixa de revelar um problema e um local aproximado onde ele foi verificado, revelando apenas que ocorreu uma falha na AUT. Pode ser difícil e demorar uma quantidade imprevisível de tempo a detecção e correcção do erro, portanto, tudo que for possível deverá ser testado numa fase de testes unitários[18].

2.4 Tipos de teste Software

Para além dos já mencionados, existem vários tipos de testes a ser efectuados a uma aplicação. Cada teste contém a especificação do correcto funcionamento do que está a ser analisado para que reacções incorrectas sejam identificadas.

São definidos 5 tipos de teste e para cada um deles a sua ligação com as técnicas de teste (“caixa preta” e "caixa branca")[15][16] :

1. **Testes unitários**

Recorrendo ao uso de técnicas "caixa branca", os responsáveis pelos testes irão efectuar toda a verificação a um nível estrutural muito baixo. Irá então ser verificado se o código tem o output que realmente é esperado.

Estes testes referem-se a testes de unidades individuais de hardware/software ou grupos de unidades relacionadas. Nesta fase de testes, o testador irá escrever código para testar um método, passando certos parâmetros de forma a verificar se o valor de retorno é o esperado.

Casos condicionais no código são identificados e um segundo caso de teste é criado para que o método possa passar pelo código ainda não testado.

2. **Testes de integração**

Neste tipo de testes componentes de software, componentes de hardware, ou ambos são combinados e testados para verificar a interacção entre eles. Usando técnicas de “caixa preta” e "caixa branca" o testador verifica que as unidades funcionam em conjunto ao ser integradas.

Estes são levados a cabo por uma equipa de testes independente que, baseando-se na especificação do sistema, procura encontrar defeitos que ocorrem em unidades de uma interface ou na interface das unidades.

3. **Testes de sistema**

O alvo destes testes são sistemas completos e integrados tendo como objectivo apurar o grau de cumprimento da aplicação com os requisitos especificados. Recorrendo a técnicas "caixa preta", os testadores, analisam a um alto nível o design e os requisitos para que os testes sejam planeados e seja assegurado o cumprimento do funcionamento especificado.

Os testes são geralmente a GUIs sendo da responsabilidade de uma equipa independente de testes, que avaliam tanto, o comportamento funcional do sistema, como requisitos de qualidade (fiabilidade, usabilidade, desempenho, e segurança).

4. **Testes de aceitação**

Este tipo de testes vai determinar se um sistema satisfaz, ou não, os seus critérios de aceitação, permitindo também ao cliente determinar se aceita ou não o que foi desenvolvido.

Este tipo de testes são normalmente elaborados pelos clientes e têm como principal objectivo verificar se tudo estará de acordo com as expectativas e requisitos desses mesmos clientes.

5. **Testes de regressão**

Neste tipo de testes a aplicação é submetida a testes feitos anteriormente, que serão desta vez seleccionados (são um sub-conjuntos dos testes iniciais) para garantir que alterações

que foram introduzidas não introduziram comportamentos indesejados na aplicação e que esta ainda cumpre todos os requisitos. Este tipo de testes surgem, quando existe uma nova versão da aplicação, para que se teste se o seu funcionamento é o correcto após as alterações feitas.

Uma vez que este tipo de testes ocorrem na fase de desenvolvimento podem existir testes do tipo "caixa branca" aos níveis de unidade e integração ou "caixa preta" na parte de integração sistema e aceitação [15].

2.4.1 Outros tipos de teste

Existem outras técnicas de teste, em que a sua utilização pode ser considerada dependendo com as necessidades de negócio ou restrições tecnológicas. Pode-se destacar as técnicas de teste de performance, usabilidade e de carga.

Pode ser mencionada ainda uma técnica de Teste Caixa Cinza, que é considerada uma mistura entre técnicas caixa preta e caixa branca onde se junta conhecimento interno do produto e saídas esperadas. São testes baseados no conhecimento do algoritmo, estados internos, arquitectura ou outras descrições mais alto-nível do comportamento do programa. Dois tipos de problemas podem ser encontrados durante os testes de caixa-cinza:

- O componente encontra uma falha de algum tipo, fazendo com que a operação seja abortada. A interface com o utilizador irá indicar que ocorreu algum problema;
- Os testes executam com sucesso, mas o conteúdo dos resultados está incorrecto. Um sistema processando dados incorrectos causa erro no resultado.

2.5 Automação de testes

A tendência actual do desenvolvimento de aplicações de software é estas serem cada vez mais complexas. Com os tempos de desenvolvimento cada vez mais reduzidos, torna-se premente o uso de ferramentas que tornem o processo de testes o mais eficiente possível.

Para se testar uma característica específica, será necessário, provavelmente, executar os testes não uma vez, mas potencialmente dezenas de vezes, é importante também verificar que erros encontrados nos ensaios anteriores foram de facto corrigidos e que não foram introduzidos novos erros na aplicação. Testes de regressão são portanto muito importantes.

Se um pequeno projecto tiver um número elevado de testes, poderá apenas haver tempo suficiente para executá-los uma vez. Executar várias vezes os testes pode ser extremamente monótono, desgastante e consumir muito tempo. Ferramentas de teste e automação podem ajudar a resolver este problema, fornecendo um meio mais eficiente de executar o teste sem ser por meio manual.

A automação de testes não será um procedimento que trará maior qualidade ao teste[19]. Com a automação dos testes ao software, a eficiência dos mesmos não será alterada, ou seja, a

sua eficiência será igual a testes realizados manualmente. A grande vantagem introduzida pela automação, é que, se de facto existirem erros, estes serão encontrados mais rapidamente.
No entanto o custo de criar e manter este tipo de teste é consideravelmente maior, quando comparado com o caso de testes manuais[20]. Uma abordagem devidamente pensada, à automação de testes é imperativa, para que os custos da sua implementação e manutenção sejam vantajosos.

2.5.1 Dificuldades da automação de testes de software

Automatizar os testes ao software não é uma tarefa fácil, sendo necessário ter em consideração que[21]:

- Automação não é barato

Por norma este tipo de testes podem levar entre três a dez mais vezes tempo para criar, verificar e documentar minimamente, do que criar e executar o teste a mão. Daí, testes que serão realizados poucas vezes não devem ser automatizados.

- Esta abordagem cria riscos de novos custos

O custo dos erros no software aumenta no ciclo de vida do software. Assim se for perdido muito tempo na criação de scripts, irá obrigar, a que a estes testes sejam feitos já numa fase avançada do desenvolvimento.

- Estes testes não são robustos

Os testes que são automatizados, são testes que já foram realizados. Poucos erros se encontrarão desta forma. Considerando a criação dos testes, o número de erros encontrados é maior, no entanto, a sua criação, é feita por testes manuais, não devendo ser então incluído nos testes automatizados. Na prática os testes automatizados serão aqueles mais simples.

2.5.2 Factores de sucesso na automação de testes de software

Segundo [6], podemos referir alguns passos a seguir para criar uma estratégia que seja um sucesso na automação de testes:

- Não criar falsas expectativas em relação ao tempo dos benefícios da automação.

Provavelmente, o benefício da automação não será percebido logo no inicio do começo da sua utilização. A diminuição do esforço irá ocorrendo à medida que o tempo for passando, e novas versões do software para teste apareçam.

- Desenvolvimento de uma automação de testes é um desenvolvimento de software.

Os testes criados não irão sobreviver, ser úteis em várias versões do software e ter um custo baixo de manutenção, sem que exista um bom planeamento. A automação utiliza uma linguagem de programação, mesmo que simplificada, como todo desenvolvimento de software. Cada caso de teste pode ser encarado como uma funcionalidade.

Os testadores, da mesma forma que os programadores devem: entender os requisitos e adoptar uma arquitectura robusta que permita desenvolver, integrar e manter as funcionalidades.

- Usar uma arquitectura de testes orientada a dados.

Testes orientados a dados irão aumentar claramente a facilidade de manutenção dos scripts de testes, uma vez que, os dados de entrada são armazenados, separadamente, dos scripts de teste, sendo lidos na altura de execução, não estando dentro do código do próprio script.

- Usar uma arquitectura baseada numa framework

Uma framework irá dar uma abordagem diferente, combinando várias estratégias de testes orientados a dados. Isola também a AUT dos scripts de teste, fornecendo um conjunto de instruções que serão como comandos básicos de uma linguagem para a criação dos scripts de teste.

- Reconhecer a realidade da equipa

Frameworks mal projectados podem destruir o projecto. Experiência em programação não é um requisito para ser um bom testador. Eles são imprescindíveis para um esforço de teste, mas não para escrever códigos de automação.
Não-programadores podem ser beneficiados por uma abordagem dirigida a dados, pois eles podem desenvolver casos de testes apenas preenchendo tabelas.

- Utilizar outros tipos de automação além do GUI

Automação de Regressão através do GUI pode trazer uma falsa ideia de cobertura que não existe. Isto pode causar uma sobrecarga na equipa deixando os profissionais mais experientes criando e mantendo scripts, em vez de procurarem erros.

Ferramentas de automação de testes pela interface com utilizador são bastante úteis, mas requerem um investimento significativo, um planeamento detalhado, uma equipa bem treinada e bastante cuidado.

2.5.3 Vantagens das ferramentas de automação

Os principais atributos de ferramentas deste tipo são:

Velocidade - Quanto tempo levaria para efectuar manualmente alguns milhares de casos de teste na Calculadora do Windows? Poderá em média levar alguns segundos a executar um dos testes. Com automação os testes podem ser realizados muito mais rapidamente.

Eficiência – Como é uma actividade que ocupa bastante tempo, enquanto se executa os testes outras coisas poderão estar a ficar em espera. Se dispormos de uma ferramenta que reduz o tempo que se leva para a execução dos testes, mais tempo sobrará para planeamento e desenho de novos casos de teste.

Exactidão e precisão - Após a realização de algumas centenas de testes a monotonia irá tomar conta do processo, onde possivelmente a atenção do testador irá diminuir, provocando distracções, o que poderá originar a que erros sejam cometidos. Uma ferramenta de teste executa todos os testes, de todas as vezes da mesma maneira, verificando os resultados obtidos na perfeição.

Redução de Recursos - Por vezes, pode ser fisicamente impossível de realizar um determinado caso de teste. O número de pessoas ou a quantidade de material necessário para criar as condições para teste, poderia ser proibitivo. Uma ferramenta de teste pode ser usada para simular o mundo real e reduzir fortemente os recursos físicos necessários para a execução do teste.

Simulação e Emulação - Ferramentas de teste são muitas vezes utilizadas para simular o hardware ou software que normalmente interage com o produto. Este "falso" dispositivo ou aplicação pode então ser utilizada para criar situações ao software a ser testado que seriam difíceis de alcançar por métodos manuais.

No entanto é sempre necessário ter em mente que estas ferramentas de teste não são um substituto para o testador, elas vão apenas ser uma ajuda para que todo o trabalho seja executado com maior fiabilidade.

É importante notar que utilizar este tipo de ferramentas nem sempre é a resposta certa. Às vezes não há nenhum substituto para a análise manual.

2.5.4 Ferramentas testadas

Foram testadas as seguintes aplicações Open Source (de notar que a aplicação disponível para testes pode influenciar nas prestações da framework, já que era realmente uma aplicação bastante simples):

→ Abbot/Costello

Framework testada que revelou melhor desempenho, a par com o Marathon. Foram, no entanto, detectados alguns problemas no que diz respeito a uma selecção de texto com o rato que se notaria, por vezes, quando ao seleccionar o texto, se libertava o botão do rato fora da caixa de texto. Um outro problema detectado, diz respeito a eventos de teclado onde o apagar de texto, quando feito, pressionando as teclas para o efeito fixamente não funciona correctamente. No entanto, de uma maneira geral teve de facto uma boa prestação no que diz respeito a gravação/reprodução mostrando alguma tolerância a alterações na interface. Os scripts de testes podem ser criados manualmente ou recorrendo ao ambiente gráfico providenciado, para gravar interacções com o interface gráfico.

→ Marathon

Esta framework é muito semelhante ao Abbot, baseando-se nos mesmos princípios de gravação/reprodução. Nos testes feitos, de uma maneira geral, esta framework teve uma performance semelhante à mencionada acima.

→ Pounder

Projecto que já não tem um desenvolvimento activo por parte dos seus criadores, fazendo referência aos projectos descritos acima como substitutos. Feitos alguns testes foram comprovadas as deficiências indicadas no site do projectos, estando estas ligadas com a gravação de eventos, principalmente eventos de rato e suas coordenadas. O suporte para componentes AWT era de facto mínimo e existiam alguns problemas relativamente ao facto de a aplicação não ser multi-plataforma não sendo este ultimo confirmado nos testes efectuados.

→ Jacareto

Esta ultima framework praticamente não foi testada, tendo demonstrado logo à partida um problema grave onde ao iniciar a captura de eventos todo o GUI ficaria branco impossibilitando a continuidade dos testes.

2.5.5 Principais falhas da automação

Como documentado em [7], os GUIs têm inúmeros estados diferentes que devem ser testados, originando assim uma enorme combinação de eventos para que tudo seja testado correctamente, podendo existir ainda dependências complexas entre os GUIs. Estes são ainda afectados por condicionantes externas que podem ditar a falha de um teste por um simples atraso num acesso a uma base de dados.

De uma maneira geral, a automação dos testes ainda não estará no que seria ideal encontrando-se diversos problemas. Um dos que merecem, desde já, uma grande atenção será o problema que se prende na localização dos componentes na interface, onde através de simples testes efectuados foi possível verificar a fragilidade das frameworks a alterações mínimas numa interface onde uma simples mudança de posicionamento de um componente na interface ou mudança de nome irá pôr em causa a integridade dos testes como descrito em [2].

Outro problema muito usual é o sincronismo entre as acções gravadas. A simples mudança de ambiente entre a máquina onde o teste é gravado e onde será reproduzido poderá trazer condicionantes. Uma vez que a rapidez de execução de um programa está intrinsecamente ligado com a capacidade da máquina onde está em funcionamento, um simples atraso entre o clique e o menu onde existiu o clique aparecer poderá originar a falha do teste. Acessos a bases de dados podem ser também um exemplo para este tipo de problema. Assim sendo, o sincronismo entre Acção1/Resultado/Acção2 terá de ser um ponto merecedor de atenção.

2.6 Resumo

Este capítulo apresenta-nos o resultado de uma pesquisa não só sobre os objectivos deste projecto, mas também, sobre a própria área de teste de software.

Foram apresentadas diversas técnicas existentes, vantagens e desvantagens, partindo depois para a área de automação a testes de software, que é onde o projecto é baseado.

Foi também feita uma análise a frameworks já existentes, mas no entanto nenhuma delas correspondia positivamente a todos os objectivos deste projecto, mostrando-se, em alguns casos, bastante frágeis, revelando então a oportunidade de se desenvolver um software que procurasse cobrir os defeitos documentados nas secções acima.

3 Descrição e abordagem ao problema

Neste capítulo iremos conhecer mais detalhadamente o projecto e os seus requisitos, bem como a abordagem proposta para conseguir cumprir o que era esperado.

3.1 Requisitos do projecto

Tendo como tema a automação de testes a interfaces gráficos em Java Swing, era então necessário desenvolver uma aplicação, que permitisse, capturar as interacções com a interface para mais tarde as reproduzir, fornecendo também informações sobre o cumprimento ou incumprimento do teste. A aplicação propriamente dita teria de permitir testar aplicações já em formato JAR ou até mesmo recorrendo apenas aos ficheiros ".class" da aplicação.

Alguma robustez era pedida para tentar minimizar os problemas deste tipo de testes, bem como permitir que testes a um determinado componente, pudessem ser gerados por uma linguagem de script não sendo, portanto, necessário recorrer sempre à captura de eventos para o mesmo componente. A integração com o TeStudio era uma ideia presente, daí ter de ser previsto uma forma de facilitar a integração com essa ferramenta.

Após o fim da execução de um teste, seria importante também, poder guarda-lo para uso futuro e sendo assim, outro dos requisitos, seria a possibilidade de gravar e depois carregar testes que estivessem guardados.

3.2 Abordagem ao problema

Nesta subsecção estão detalhados todos os passos tomados para cumprir os objectivos pedidos.

3.2.1 Linguagem de script

Um dos objectivos deste projecto, para além da criação do script por interacções com o GUI, era existir a possibilidade de acrescentar passos novos ou até mesmo eliminar alguns sem ser necessário qualquer interacção com a AUT.

Impunha-se a utilização de um idioma específico de domínio (Domain Specific Language, DSL). Uma DSL é uma linguagem criada para resolver problemas num dado domínio, não sendo a sua intenção resolver situações fora deste. Estas linguagens podem ser textuais, como a maior parte das linguagens de programação ou gráficas. Este tipo de linguagens são extremamente úteis quando permitem que determinadas situações ou problemas possam ser expressos mais claramente, do que usando uma linguagem de programação[23][24].

Existem vários padrões de utilização de um idioma específico de domínio:

- Processamento em ferramentas autónomas, invocadas directamente (linha de comando ou a partir de um Makefile);
- Idiomas de domínio específicos que são chamados (em tempo de execução) de programas escritos em linguagens como C, para executar uma função específica, muitas vezes os resultados da operação são devolvidos para a linguagem de programação "anfitriã" para processamento posterior. Geralmente, um intérprete ou máquina virtual para a DSL é embutido na aplicação anfitriã;
- Idiomas de domínio específicos estão embutidos em aplicações, são usados para executar o código que é escrito pelos utilizadores da aplicação, código gerado dinamicamente pela aplicação, ou ambos.

Muitos idiomas de domínio específicos podem ser utilizados em mais de uma maneira.

Assim sendo está definida uma linguagem, à qual foram acrescentadas algumas instruções, que permitirá escrever os testes manualmente. A linguagem tenta prever todo o tipo de acções que podem ser efectuadas sobre cada componente do GUI. Por sua vez nesta linguagem os componentes serão indicados através de um *alias* que lhe é atribuído, automaticamente ou pelo utilizador da aplicação desenvolvida.

No Anexo C: poderemos encontrar a lista das instruções definidas e perceber quais os conceitos transportados de uma linguagem já definida e quais as alterações feitas a essa mesma linguagem.

Existem então as seguintes instruções:

Operações de rightclick

Existem três operações deste tipo. A razão de existir mais do que uma instrução deste tipo, foi a necessidade de se distinguir certos componentes, como por exemplo, neste tipo de operação sobre uma tabela terá de existir a indicação da coluna e linha onde será feito o clique com o botão direito. Assim sendo as instruções deste tipo disponíveis são:

- **rightclick on** "Item"@alias - Esta operação está definida para simular o clique no botão direito do rato sobre um nó de uma árvore;

- **rightclick on** (Row,Column)@alias - Esta instrução será utilizada quando for necessário simular este tipo de clique sobre uma tabela;

- **rightclick on** alias - Instrução usada para as simulações deste tipo de acção sobre o resto dos componentes.

Operações de clique

Existem dois tipos de instruções para este tipo de acção, uma para cilques sobre quase a totalidade dos componentes e um outro para indicar cliques no botão de expansão de uma árvore, a sua estrutura terá de ser a seguinte:

- **click on** "Item"**@**alias - Esta instrução existe apenas para indicar que terá de existir uma expansão de um nó de uma árvore;

- **click on** alias - Instrução mais simples para simular um clique, poderá ser usada com os restantes componentes.

Operações de select e deselect

Existem quatro tipos de instruções de select para seleccionar elementos numa lista, combobox, árvore ou tabela, podendo nos primeiros dois exemplos de componentes seleccionar-se elementos através do seu índice ou então do seu texto. Uma das instruções deste tipo, existe, para efectuar selecções de texto. A sua estrutura será a seguinte:

- **select** "nome" **from** alias - esta instrução será usada para selecções sobre uma lista, combobox ou árvore, sendo que o campo "nome", indicara o texto do elemento a seleccionar. Poderão existir múltiplas selecções sendo para isso necessário adicionar vários campos "nome" separados por um espaço. Analogamente existirá uma operação exactamente igual para se fazer a desselecção substituindo-se apenas a primeira palavra (select) pela palavra "deselect";

- **select** 234 **from** alias - À semelhança da instrução anterior esta é aplicada sobre os mesmos componentes (excluindo a árvore), com a excepção de que o argumento será o índice do item e não o seu texto. A operação de selecções múltiplas também está

disponível sendo apenas necessário separar os índices com uma virgula. A desselecção funcionará exactamente da mesma maneira só que, mais uma vez, a palavra inicial terá de ser "deselect";

- **select** (Row,Column) **from** alias - Indicação de um select de uma determinada célula de uma tabela. Para seleccionar uma linha a instrução é exactamente a mesma podendo ser indicada qualquer um dos índices de coluna. selecções múltiplas e desselecções também estão disponíveis para este componente e funcionarão exactamente como descrito na primeira instrução deste tipo;

- **select** "text" **in** alias - Esta operação poderá ser realizada sobre componentes de texto, e é usada para fazer uma selecção de parte de um texto ou a sua totalidade.

Operação de write

Uma operação de write existe para inserir texto nos componentes que aceitem a sua inserção. O formato definido foi o seguinte:

- **write** "text" **to** alias - O campo "text" será o texto que irá ser introduzido.

Operação de press

Três operações de press existem para simular o input de teclado sobre componentes ou então um simples input de TAB para mudar o foco de um componente para outro. Estas instruções aceitam como argumentos as teclas ENTER, TAB, DELETE e BACKSPACE. Uma das instuções prevê também a inserção de texto numa localização especifica de um componente de texto para que, por exemplo, exista a possibilidade de apagar dois caracters no meio de uma frase. A sua estrutura terá de obedecer a um dos seguintes formatos:

- **press** KEY(@Index)? **in** alias - Instrução que aceita como argumento KEY o texto ENTER, TAB, DELETE e BACKSPACE. Existirá também o campo opcional index que é para indicar o índice (por índice, entende-se a localização de onde colocar o cursor, como por exemplo, no caracter 4) dentro de um componente de texto;

- **press** 'KEY' **in** alias - Esta instrução está definida para o press de uma só tecla sobre um componente;

- **press TAB** - Esta instrução esta definida para a mudança de foco de um componente para outro através do teclado.

Operações de assert

Existem quatro operações de assert para se verificar o estado dos componentes depois de efectuada uma dada acção. Existe ,por exemplo, a possibilidade de verificar o número de

elementos que existe num componente ou se de facto um determinado elemento existe no componente. As instruções terão o seguinte formato:

- **assert that** alias **text is** "whatever" - Com o uso desta instrução poderemos verificar se o texto do componentes corresponde ao esperado;

- **assert that** alias **has** 34 **elements** - O intuito desta operação será verificar se o componente terá, de facto, o número de elementos esperado;

- **assert that** alias **has element** "sed" - Esta instrução servirá, para verificar a existência de determinado elemento, num certo componente;

- **assert that** alias **is** (not)? **selected** - Instrução para verificar o estado de uma checkbox, se estará ou não seleccionada.

Operações de log

Existem três operações de log usadas para, enviar para um ficheiro de texto, os estados de certos tipos de componentes. Podemos ver abaixo então a sua estrutura :

- **log text of** alias - Esta operação gravará em um ficheiro o conteúdo de um componente de texto;

- **log selected item in** alias - O registo em ficheiro do item seleccionado num componente está também previsto e pode ser feito através desta instrução;

- **log number of items in** alias - Para se gravar no ficheiro o número de items existente num dado com componente, esta instrução será usada.

O ficheiro de texto com o log estará localizado na pasta raiz da aplicação desenvolvida e será criado usando a o texto "LOG_" e adicionando o tipo da janela. Uma entrada no log terá o seguinte aspecto:

Data --->Tipo de Componente:tipo --->Name:nome --->(tipo de log): dados recolhidos

O texto (tipo de log), será variavel consoante a operação, podendo ser "número de items", "selected items" ou "texto".

Operação de close/maximize/minimize

Estas instruções terão como objectivo reproduzir o fecho, maximização e minimização de uma janela. O formato especificado para elas foi:

- **close** alias - operação para efectuar a acção de fecho de determinada janela;

- **maximize** alias - operação para efectuar a acção de maximização de determinada janela;

- **minimize** alias - operação para efectuar a acção de minimização determinada janela.

Operação de wait

Existem dois tipos de operação de wait que deverão seguir o seguinte formato:

wait 12345 - Esta instrução será para forçar uma espera na reprodução de "12345" milisegundos. Onde este tempo de espera poderá ir de 1 até 99999 milisegundos;

wait on alias - Este tipo de espera, será para obrigar a reprodução a parar até que uma barra de progresso esteja completa, indicando que a AUT acabou a acção que estaria a efectuar.

Ambas as acções existem para tentar minimizar os efeitos de acções executadas antes do tempo, sendo um acréscimo a uma outra funcionalidade(descrita mais a frente) para prevenir este tipo de problemas.

Operação de capture

Esta operação existe para gravar uma imagem (screenshot) da aplicação. A instrução é constituída apenas pela palavra "capture".

Operação de drag and drop

Esta instrução existe para reproduzir um arrastamento de um certo componente quando sobre ele uma operação deste tipo for possível. A instrução terá o seguinte formato:

drag ("item"@)?sourcealias **to** ("item"@)?targetalias - Nesta instrução temos dois campos opcionais podendo existir apenas um ou outro, que servirão para indicar um item especifico de um componente (um item de uma árvore, por exemplo). Se este campo não estiver definido será interpretada como apenas um mover de um componente para outro identificados por "sourcealias" e "targetalias", que serão os *alias* que identificam os componentes usados nesta operação.

3.2.2 Integração com TeStudio

Para fazer face a um dos objectivos e possibilitar uma futura integração com uma outra aplicação já desenvolvida, o TeStudio, o processo de comunicação entre a o GUI da aplicação desenvolvida e o módulo de gravação e reprodução não poderiam estar presos à linguagem Java. Para esse efeito, o processo de comunicação será baseado num Web Service e os dados transportados por objectos XML.

A aplicação foi dividida em dois módulos onde claramente se distingue o módulo de ambiente gráfico/Web Service e o módulo de gravação e reprodução. Esta divisão foi assim estruturada para que depois o módulo que realmente faz a automação dos testes a GUIs em Java

(Módulo de gravação e reprodução) possa ser integrado, mais facilmente, como "plugin" no TeStudio. Na imagem abaixo poderemos ver um diagrama que nos mostra a divisão modular da aplicação bem como a comunicação entre eles:

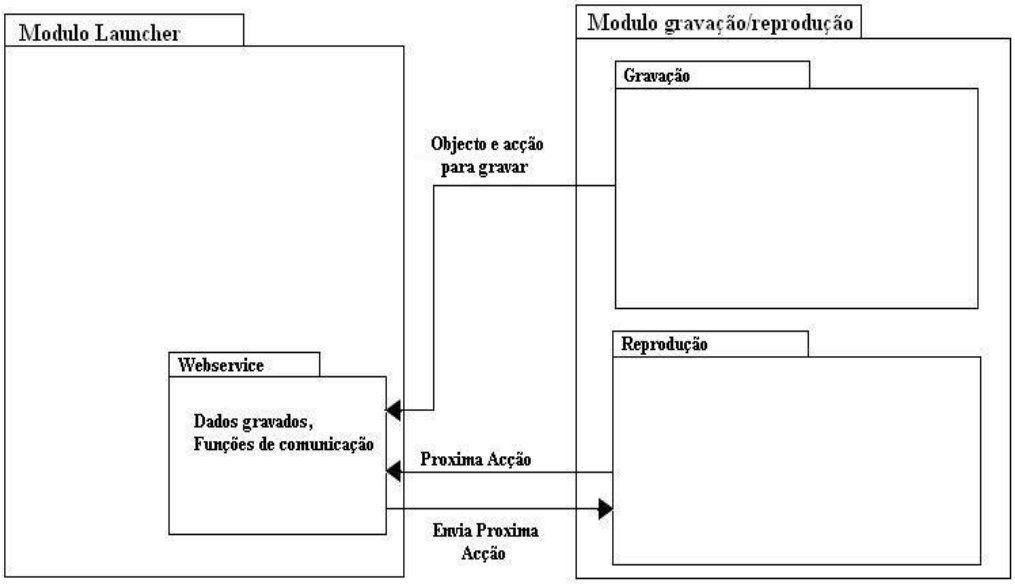

Figura 3: Módulos da aplicação e comunicação entre eles

3.2.3 Gravação dos eventos

Todos os eventos de I/O em Java são enviados para uma fila de eventos própria para serem processados, assim sendo, para detectar eventos de rato, bem como eventos de teclado foi substituída a fila de eventos do Java por uma costumizada (como demonstrado abaixo no diagrama abaixo), tornando mais fácil a recolha de informações e dando maior controlo sobre os eventos.

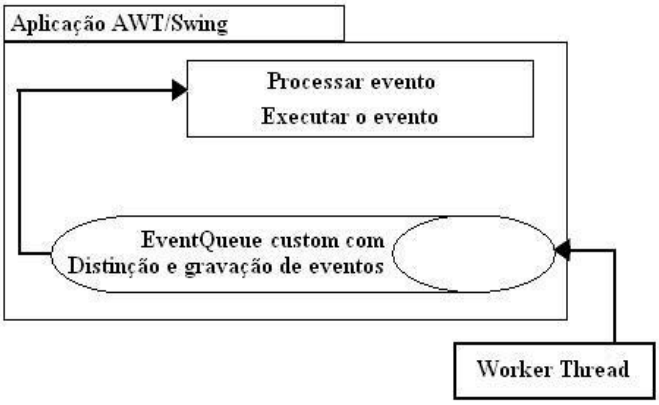

Figura 4: Eventqueue custom

Com a detecção destes eventos podemos conseguir todas as informações sobre o componente onde existiu interacção e usando reflexão, poderemos correr qualquer dos métodos disponíveis para esse componente de modo a conseguir uma informação mais precisa e detalhada.

Para a gravação dos eventos está idealizado um formato específico introduzindo-se os conceitos de "ControlItem" e de "ControlAction" sendo que:

ControlItem - é o objecto que guardará um componente especifico e guardará as informações necessárias para a sua identificação em execuções de um teste. Este objecto irá guardar informações sobre o tipo do componente, o "Name" atribuído a ele, a framework a que pertence, o texto do componente, o seu índice dentro do pai e o pai propriamente dito. O pai por sua vez é também um objecto do tipo ControlItem.

ControlAction - é o objecto que servirá para armazenar a acção sobre um dado componente. Terá informações sobre a acção propriamente dita, tendo em conta a linguagem de script especificada anteriormente, o componente sobre o qual será efectuada a acção e ainda o tipo da acção.

A gravação de tempos entre acções durante a gravação, na tentativa de efectuar um melhoramento na sincronia entre as acções foi um método testado. No entanto, não se mostrou, de modo algum, uma boa forma para se controlar quando a acção deveria ser efectuada. Existiam demasiadas variáveis incontroláveis para este ser um método considerado aceitável, de forma a, tentar minimizar uma das falhas deste tipo de aplicações. Em alternativa, o método pensado para tentar colmatar esta falha, foi usar a medida de tempo de CPU usado pelo programa, sendo que, se a aplicação não estivesse a usar o CPU seria um indicador para que a próxima acção poderia ser executada. Para prevenir o caso da aplicação estar a efectuar alguma operação em segundo plano e, consequentemente, estar sempre a usar o CPU, foi introduzido um temporizador que visa forçar a execução da próxima acção mesmo que o CPU esteja a ser usado.

Para tentar optimizar o processo de gravação, sempre que exista um clique sobre qualquer componente, irá ser testado se este componente tem algum evento semântico associado, não sendo gravada a acção se não o tiver.

O diagrama abaixo representa apenas uma pequena parte do que será o processo de gravação. Mostram-se três componentes exemplo e o processo que ocorrerá quando existir um evento de rato. O funcionamento é semelhante para todos os componentes, mudando obviamente a acção a gravar já que cada componente tem uma função especifica na interface. Irão ser consideradas todas as acções de rato ou teclado e também rato + teclado (por exemplo ctrl + clique, para múltiplas selecções).

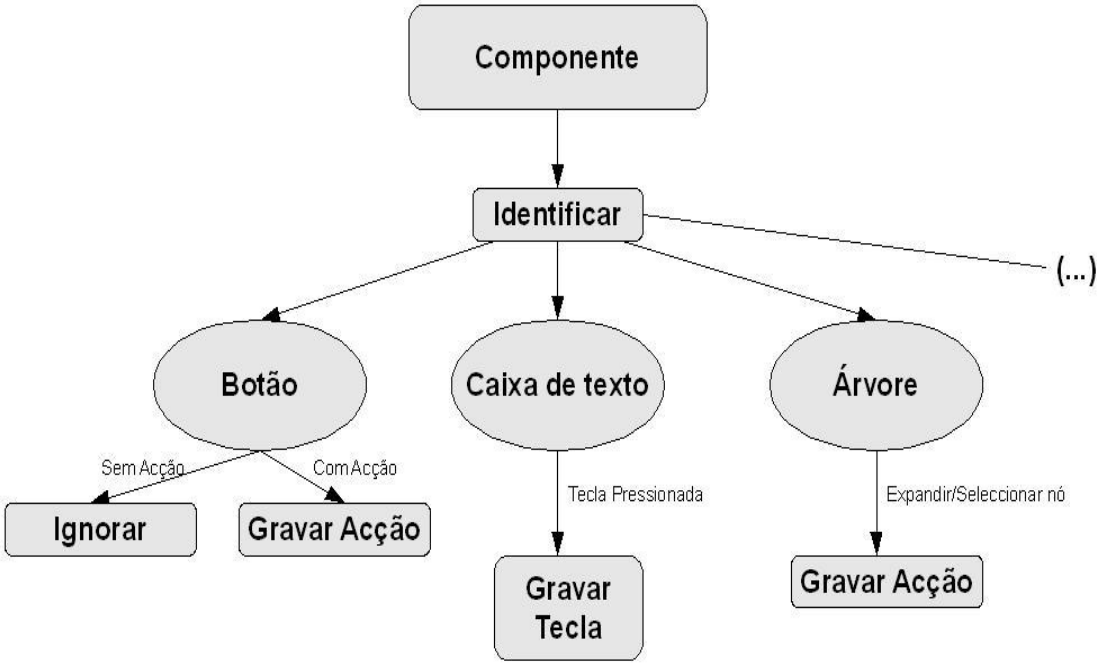

Figura 5: Diagrama de gravação de um tipo de evento para certos componentes

De notar que a robustez da aplicação terá sempre a sombra de falhas que afectam todas as aplicações deste género.

A cada componente, na altura da gravação e quando ocorre uma interacção, é atribuído um *alias* automático, podendo ser alterado mais tarde. Este *alias* identificará o componente, para que, depois, os testes possam ser escritos manualmente e directamente, numa área de texto disponível para o efeito. Estes testes terão de ser escritos, obrigatoriamente, na linguagem que existe para o efeito. A linguagem é de uma maneira geral, extremamente simples e fácil de usar como foi visto no primeiro ponto desta secção.

Assim sendo nesta fase de gravação, podemos optar por duas opções:

- Interagir com o GUI para que o script de acções seja criado automaticamente;

- Iniciar o processo de gravação e em alternativa atribuir só os *alias* aos componentes para que o script seja escrito manualmente.

3.2.4 Reprodução de eventos

O processo de reprodução poderá ser inicializado logo após a criação do script, ou então, carregando um ficheiro que contenha o script a ser executado e todos os ControlItems em XML que foram obtidos durante a gravação.

Quando a reprodução dos eventos começa o módulo de reprodução envia então um pedido ao Web Service para que este retorne um objecto do tipo ControlAction. Estes objectos são

construídos e enviados no momento em que o pedido é feito e tendo em conta as instruções que foram gravadas sendo então enviados para a classe responsável pela reprodução do teste. Após receber o objecto este é trabalhado, separando-se os argumentos e a informação sobre o componente sobre o qual decorre a acção. É verificada a validade da expressão e que tipo de acção está presente recorrendo a expressões regulares.

Os passos que ocorrerão de seguida dizem respeito a interacção com a AUT processando-se da seguinte forma:

- Conjugando todas as características guardadas no ControlItem o componente descrito por este irá ser "procurado". Como está disponível toda a hierarquia do componente e dos seus "containers", até à janela principal, irá ser feita uma pesquisa começando nesta e percorrendo toda a hierarquia descrita no ControlItem, para que, se alcance o "container" que é o pai do componente procurado e então depois recorrendo as informações guardadas sobre o componente tentar distingui-lo dos outros que poderão existir nesse mesmo "container".

  - A falha desta pesquisa no entanto poderá acontecer, uma vez que, o teste que está a ser efectuado no momento poderá ter sido gravado com outra versão da aplicação e eventualmente o componente descrito pelo ControlItem ter sido posicionado num outro espaço do GUI e, consequentemente, toda a sua hierarquia ter mudado. Neste caso a solução escolhida foi fazer uma pesquisa recursiva por toda a hierarquia de componentes nela presente, procurando o componente que mais se aproxima ao descrito e sendo esse devolvido.

- Tendo a referência para o componente, obtida através de reflexão poderemos obter a posição (x,y) do componente, e nesse momento usar a API Robot disponível em Java para simular o movimento do rato e a acção que é indicada pela instrução.

Ainda durante a reprodução, e quando uma acção está a ser efectuada, podemos encontrar problemas do tipo:

- Não ser encontrado nenhum componente;
- Falha na condição de um assert;
- Problemas com a movimentação do rato;
- Componente não estar visível ou desactivado.

Nestes casos, e caso existe algum problema, que gere alguma excepção na acção de reprodução é enviado um erro para uma área de texto, para que no fim seja decidido se o teste foi então bem sucedido ou não.

Existe também o conceito de "warning" que, por si só, não determina a falha nos testes, este tipo de aviso será lançado caso a identificação do componente, tenha sido feita por alguma característica que não permita dar certeza sobre a identificação do componente. Exemplificando,

se for usado apenas o índice do componente que está presente no ControlItem, a probabilidade do componente devolvido para se efectuar a acção, ser o componente procurado, reduz-se bastante e assim sendo um aviso é enviado.

No caso de ocorrer algum erro durante a reprodução, a continuidade dos testes poderá ser comprometida e por isso é dado a escolher, no inicio, se o testador pretende parar a reprodução ao primeiro erro ou prosseguir até ao fim.

Todo este processo é então repetido até que mais nenhuma acção seja devolvida pelo Web Service e é então mostrado o log da reprodução juntamente com a mensagem atestando o sucesso do teste.

Durante este processo cíclico de reprodução podemos distinguir duas opções tomadas para tentar obter sincronia em todas as acções. Para controlar quando uma acção deverá ser efectuada irá ser tido em conta o uso do CPU por parte da AUT e um temporizador de sete segundos.

Uma vez que, o tempo que alguma aplicação demora a executar alguma tarefa poderá ser variável foi optado, a cada ciclo, obrigar a aplicação de reprodução a esperar que a AUT deixe de usar o processador. Desta forma, quando o a percentagem de uso de CPU por parte da AUT chegar a zero sabe-se que é o momento de executar a próxima acção, já que a aplicação não está a realizar nenhuma tarefa.

Esta solução levantou um outro problema, aplicações que estivessem a executar alguma operação em segundo plano, iriam fazer com que a aplicação de reprodução pudesse ser parada por mais tempo que o que supostamente seria necessário. Por essa razão foi introduzido um temporizador para que a aplicação de reprodução se liberte da sua espera e prossiga com a execução das acções.

3.2.5 Reutilização de casos de teste

A reutilização dos testes foi um dos requisitos iniciais. Para conseguir esta reutilização foi tomada a opção de guardar dos dados recolhidos durante a gravação num ficheiro de texto.

O script sendo ele texto usual, irá ser guardado exactamente como está na aplicação, por sua vez, os objectos ControlItem como são uma estrutura, irão ser guardados em formato XML para simplificar o seu carregamento para a aplicação.

Poderia ter-se optado pela gravação de uma ControlAction (que incluiria tanto a acção como o componente a que a acção é dirigida). No entanto, o facto de poder existir várias acções para um mesmo componente, iria originar que o ficheiro contivesse informação redundante (vários ControlItems guardados para um mesmo componente). Como podem existir componentes que foram gravados e não têm nenhuma acção contemplada, estes componentes iriam ser negligenciados o que não seria a situação ideal.

Assim sendo o formato do ficheiro será:

[**Instruções definidas**]
-a-b-c-d-e-f
alias**!_!_!**ControlItemXML
-a-b-c-d-e-f
alias**!_!_!**ControlItemXML
-a-b-c-d-e-f

As "strings" a negrito são strings delimitadoras dos dados.
No Anexo D: podemos ver o esquema em xml dos ControlItems e também dos ControlActions.

3.2.6 Tecnologias utilizadas

Tratando-se de testes a aplicações de Java Swing dispomos de duas API´s disponibilizadas pelo Sun no Java para facilitar a produção de aplicações deste género, sendo elas:

- **Reflexão**

Com recurso a esta API poder-se-á examinar e introduzir certo tipo de alterações ao programa a ser testado em tempo de execução, não sendo necessário dispor do código fonte da aplicação. Esta API irá ser usada para introduzir “event listeners” na AUT, eventos esses que nos permitirão identificar quais as alterações a um dado componente.

Iremos recorrer a reflexão também para identificar quais dos cliques deverão ser gravados, procurando na altura em que um componente é pressionado se este tem alguma acção associada. Esta API servirá também,obviamente, para iniciar o programa a ser testado.

- **Robot**

No caso da API Robot, esta irá ser utilizada para reproduzir eventos de rato e teclado, gravados na fase anterior.

No que diz respeito a funções que estão indirectamente envolvidas no processo de gravação e reprodução foram então utilizadas mais 3 API´s disponíveis:

- **JAXB (Java Architecture for XML Binding)**

Java Architecture for XML Binding (JAXB) permite mapear classes Java para representações XML. JAXB prevê duas características principais: a capacidade de converter objectos Java em XML e o inverso, ou seja, a reverter o processo de conversão e passar de XML para objectos Java. Por outras palavras, o JAXB permite converter de XML para Objectos Java e o inverso sem ser necessário implementar qualquer tipo de parsers.

No diagrama abaixo podemos verificar como todo o processo se desenrola, desde a facultação do esquema xml até a conversão entre objectos em XML.

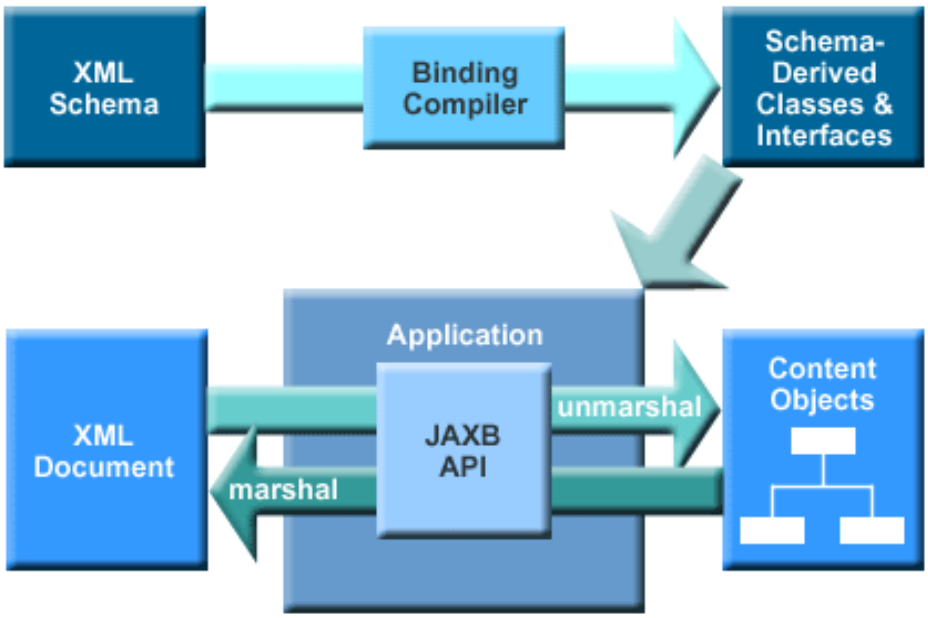

Figura 6: Operações passiveis de serem efectuadas com JAXB

Toda a complexidade dos passos representados no diagrama é invisível para quem está a usar a API, começando na conversão do esquema em classes até a operação de marshal. Estas são vistas como uma simples chamada a um método.

- **JNI (Java Native Interface)**

JNI é usado em determinadas situações em que um código Java não pode aceder a certos recursos, por causa da estrutura dos Sistemas Virtuais (sob qual a Máquina Virtual de Java foi implementada).
O JNI permite que um código escrito em Java, utilize a implementação de uma biblioteca escrita em C/C++, Assembler, e outras linguagens de programação. Assim sendo, podem existir um conjunto de bibliotecas escritas em diversas linguagens, e usar o JNI para as disponibilizar a uma aplicação em Java. A sua usabilidade estende-se também a implementação de pontos críticos, em aplicações que exijam recursos de baixo-nível, podendo a aplicação Java invocar todas essas funções.
A JNI permite o uso das vantagens da Linguagem de Programação Java nos métodos nativos, a captura e lançamento excepções de uma biblioteca qualquer está disponível, podendo ser tratadas dentro da aplicação Java.
No caso da aplicação que foi desenvolvida, o JNI foi usado para invocar código escrito na linguagem C para poder aceder aos tempos de uso do CPU por parte da aplicação.

- **JAX-WS**

API de Java para serviços Web XML (JAX-WS) é uma parte importante da plataforma Java EE 5. Veio dar continuidade a API de Java para RPC (JAX-RPC) baseado em XML, JAX-WS simplifica a tarefa de desenvolvimento de serviços Web utilizando a tecnologia Java. Fornecendo suporte a múltiplos protocolos, como SOAP 1.1, SOAP 1.2, XML, e fornecendo um recurso para dar suporte a protocolos adicionais junto com HTTP. JAX-WS usa JAXB 2.0 para vinculação de dados e dá suporte a personalizações para controlar interfaces de ponto de extremidade de serviço geradas. Com suporte a anotações, JAX-WS simplifica o desenvolvimento do serviço Web e reduz o tamanho de arquivos JAR do tempo de execução. Esta foi a API escolhida para implementar o suporte da comunicação entre os dois módulos da aplicação desenvolvida.

3.3 Escalonamento dos objectivos

O projecto foi subdividido em algumas etapas, etapas essas a que foi atribuído um tempo estimado. Foram elas:

Contextualização - 2 semanas
análise e concepção - 2 semanas
Implementação e testes funcionais - 12 semanas
Elaboração do relatório - 4 semanas

Como podemos ver no diagrama de Gantt (ver anexo A) o tempo gasto no total não foi o somatório de todo o tempo estimado uma vez que existiram fases que foram sendo abordadas simultaneamente.

3.4 Resumo

Este capítulo começa por dar uma ideia dos requisitos deste projecto, e no seu decorrer entramos com mais profundidade sobre esses objectivos, dando a conhecer a solução escolhida para o problema. Toda a abordagem ao problema é descrita, começando por dar a conhecer a estrutura das instruções disponíveis para a escrita de testes, passando pela separação em dois grandes módulos: Launcher e Gravação/Reprodução, para fazer face a uma futura integração. De seguida é aprofundado o funcionamento da aplicação no que diz respeito ao processo de gravação e reprodução de eventos e por fim são dadas a conhecer as tecnologias usadas para poder concretizar todos os objectivos e o espaço temporal em que eles foram concretizados.

No próximo capítulo iremos ver como todas as tecnologias foram combinadas e ficar a conhecer todos os detalhes da implementação.

4 Implementação

A aplicação desenvolvida está dividida então em dois grandes módulos: Launcher e RecReplay. O núcleo desta aplicação está situado no módulo RecReplay, sendo lá que estão definidos todos os métodos que irão possibilitar a gravação e reprodução. Esta separação foi pensada devido a uma futura integração com o TeStudio, mas também por certas impossibilidades impostas pela linguagem Java(que estão descritas no ponto 4.5). Ao separar o módulo Launcher do RecReplay foi necessário implementar um método para facilitar a comunicação entre eles em tempo de execução, opção essa que incidiu sobre Web Services para ,uma vez mais, facilitar a futura integração e comunicação entre esta aplicação em Java e o TeStudio (desenvolvido em C#).

No decorrer deste capítulo aprofundaremos a questão da integração, clarificar-se-à a implementação do módulo Launcher, bem como, do módulo RecReplay, mais concretamente de como foi implementado a gravação de eventos e sua reprodução.

A implementação de como se faz a validação da Domain Specific Language (DSL) também está aqui documentada.

4.1 Linguagem de script

Depois de definida a DSL era necessário validar todas as instruções recebidas. Foi necessário reconhecer se de facto a instrução estava bem construída e identificar de forma concisa cadeias, padrões de caracters e palavras. Expressões regulares são escritas numa linguagem formal que sendo depois passadas a um interpretador permitem identificar os padrões necessários.

Em Java dispomos de mecanismos muito bons para o tratamento de expressões regulares.

Para isso é criada a classe linguagem, onde todas as expressões regulares para reconhecimento de todas as instruções que a aplicação irá disponibilizar. Todas as expressões

estão num array de strings e , por exemplo, para reconhecer uma selecção de um elemento numa lista, a expressão regular correspondente é a seguinte:

- **select \"(.+)\" from (.+)**

onde (.+) significará que naquela posição poderá ocorrer qualquer tipo de caracter. Para instruções onde sabemos que só um inteiro poderá existir usaremos \\d+, que significa que poderemos encontrar um ou mais dígitos.

A classe Java.util.Scanner faz a leitura e a análise gramatical em strings e tipos primitivos utilizando expressões regulares, fazendo uso da classe Java.util.Pattern que contém a expressão regular desejada que será usada para a comparação. Assim criando um objecto da classe scanner, com o string da instrução e fazendo uso dos métodos:

- **findInLine(Pattern pattern)**
- **match()**

Iremos percorrer cada posição do array das expressões regulares e usando para cada uma os dois métodos anteriores, para tentar encontrar alguma que valide a instrução recebida. Caso exista alguma expressão que valide, a posição do array é passada a um “switch” para que então possa ser executado o código que corresponde aquela expressão. De notar que, o método match retorna um objecto da classe MatchResult, onde estão separados os valores que correspondem aos tokens (.+) e \\d+, para que então possam ser usados na execução. Caso não seja encontrada nenhuma expressão regular que valide a instrução recebida é emitido um erro indicando que a instrução está mal formada.

4.2 Integração

Tendo em conta que o TeStudio é desenvolvido em C# e esta aplicação em Java, era necessário desde já implementar uma forma de comunicação entre os dois módulos desta aplicação que permitisse facilidade na integração futura, ou seja, exigia-se interoperabilidade. A escolha de Web Services pareceu perfeita, uma vez que, permitem a comunicação entre aplicações enviando e recebendo dados em XML. Assim cada aplicação poderá utilizar a sua linguagem e quando existir necessidade de comunicação os dados são convertidos em XML e enviados. Para a representação e estruturação dos dados nas mensagens recebidas/enviadas é utilizado o XML (eXtensible Markup Language). As chamadas às operações, incluindo os parâmetros de entrada/saída, são codificadas no protocolo SOAP (Simple Object Access Protocol, baseado em XML). Os serviços (operações, mensagens, parâmetros) são descritos usando a linguagem WSDL (Web Services Description Language). O processo de publicação/pesquisa/descoberta de Web Services utiliza o protocolo UDDI (Universal Description, Discovery and Integration).[22]

Para então ser possível uma integração mais fácil, foi necessário utilizar uma estrutura de dados semelhante a existente na aplicação já desenvolvida pela empresa. A comunicação entre

os dois grandes módulos é então feita via Web Service e por objectos XML do tipo ControlItem e ControlAction.

4.2.1 JaxB

Era então necessária a criação das classes representativas dos objectos ControlItem e ControlAction. Para isso e, de forma a simplificar o processo, utiliza-se esta tecnologia (JaxB) que, através de um esquema em XML que lhe é passado, irá criar todas as classes dos objectos que nele estão definidos.

Um esquema de XML já existente foi alterado, de maneira, a que todos os campos achados necessários para minimamente identificar um componente estivessem presentes. Aceitando como input este ficheiro, a API faz o parse e cria todas as classes necessárias para a utilização dos objectos definidos no esquema. Este funcionamento pode ser ilustrado pelo seguinte diagrama:

Figura 7: Conversão automática de esquema de XML para classes Java

Neste caso as duas grandes classes mais relevantes serão o ControlItem e o ControlAction. Abaixo poderemos ver os diagramas representativos destes dois objectos:

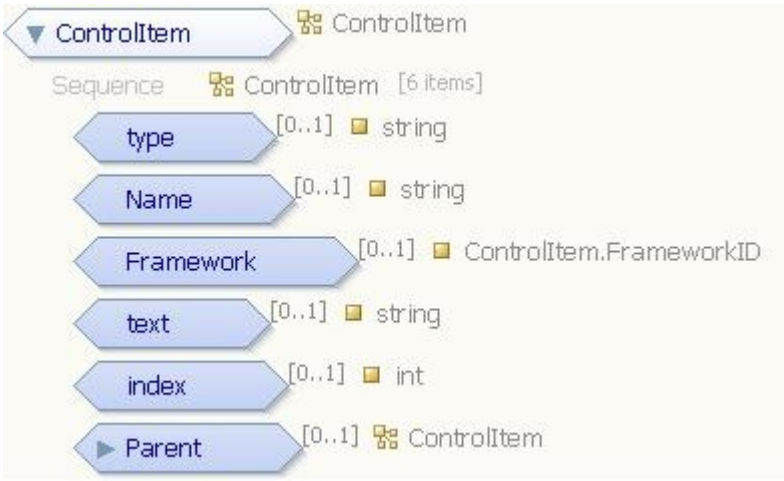

Figura 8: Estrutura de um ControlItem

No diagrama acima podemos ver todos os campos existentes num ControlItem, cada um detém uma informação relevante para a identificação do componente que representa. Cada ControlItem tem seis campos:

Type – Neste campo, do tipo string, estará guardada a classe do componente em questão (Ex: Javax.S.JButton);

Name – O name representa o nome atribuído ao componente por quem estará a desenvolver. Se este campo fosse sempre utilizado e único, nunca existiriam problemas na distinção entre dois componentes;

FrameWork – Este campo representa a framework a que o componente pertence, tendo em conta a integração, é necessário ter também este campo para distinguir a framework a que ele pertence. Este campo poderá tomar os valores: “JAVA_SWING”, “WINFORMS”, ”WPF”;

Text – Neste campo estará gravado o texto do componente, ou seja, caso o componente tenha algum texto associado, como por exemplo, o texto de um botão ou então a label associada a uma checkbox. No caso de botões que tenha associado, não texto, mas um icon, o nome do ficheiro desse icon será guardado neste campo;

Index – Após a análise da API de Java e dos métodos para devolver um componente dentro de um container, verificou-se que estão disponíveis métodos que aceitam um índice de componente. Este índice do componente ajudará a identificar o seu posicionamento dentro do container. Este índice é calculado pela posição do array retornado pelo método getComponents(), disponível para todas as classes derivadas de JComponent;

Parent – Este campo será um outro objecto ControlItem, representando o container que o componente está inserido. Neste campo estará toda a árvore de hierarquia em que o componente onde se verificou a acção está inserido.

Todos os campos têm possibilidade de ser nulos, no entanto, na realidade só no “Parent”, “Text” e “Name" é que isso irá acontecer.

No diagrama representado abaixo encontramos a estrutura de um ControlAction:

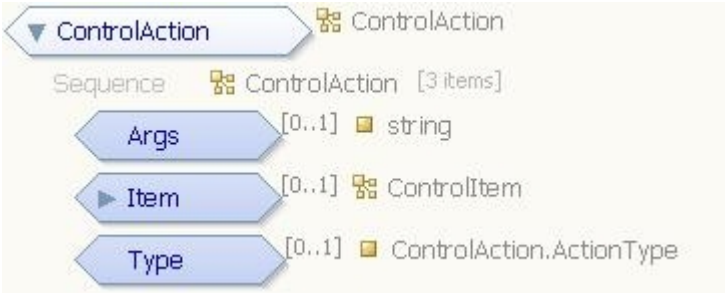

Figura 9: Estrutura de uma ControlAction

Este objecto será usado para representar uma acção. É construído apenas em tempo de execução, à medida que vão sendo pedidas as sucessivas acções que estão descritas no script.

Três campos estão definidos neste objecto:

Args – String com a linha de script que representa uma acção;

Item – Contém o ControlItem, ou seja, a informação relativa ao componente onde a acção ira ocorrer;

Type – É o tipo da acção sobre o componente. Este campo poderá tomar os valores de "NONE","CLICK","RIGHTCLICK","DRAGNDROP","ENTER_TEXT","INPUT_CHARKEY","INPUT_DESELECT","INPUT_SELECT","INPUT_SYSKEY","ASSERT_PROPERTY","ASSERT_ELEMENT","ASSERT_LISTCOUN","LOG_PROPERTY","LOG_SELECTED","LOG_LISTCOUNT","MAXIMIZE","MINIMIZE","CLOSE","WAIT","PAUSE".

Para comunicação com o Web Service é usado um “Marshler” para converter os objectos em XML para que então eles sejam dados como argumento aos métodos definidos no Web Service.

Após recebidos nos métodos do Web Service é feita a operação inversa, para então os objectos ficarem em variáveis no módulo launcher. Estes dois processos estão ilustrados nos diagramas abaixo:

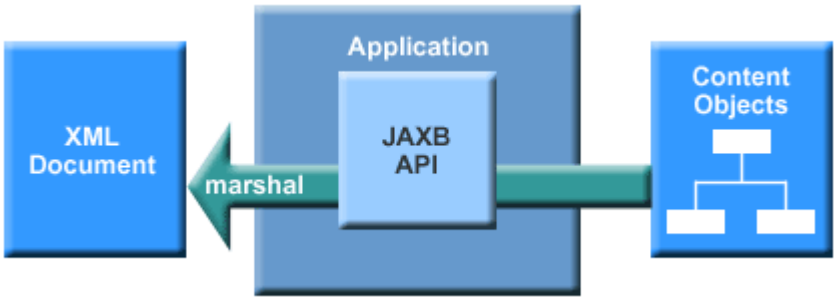

Figura 10: Conversão de objecto Java para XML (Operação de Marshal)

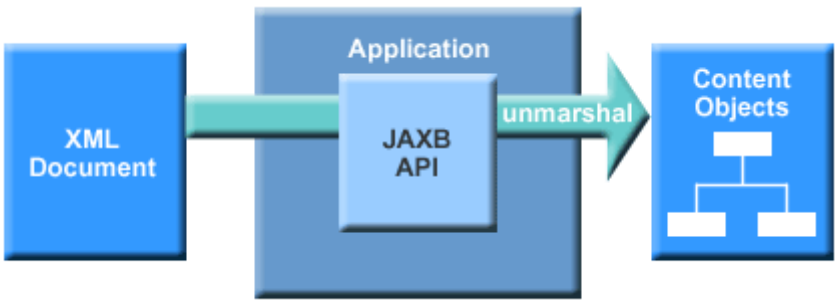

Figura 11: Conversão de XML para objecto Java (Operação de Unmarshal)

4.2.2 Web Service

A comunicação entre os dois módulos será efectuada através deste serviço. Nele são definidos vários métodos e tendo também a variável onde fica guardado todas as acções, ControlItems e *alias* respectivos. Abaixo temos o diagrama representativo desta classe:

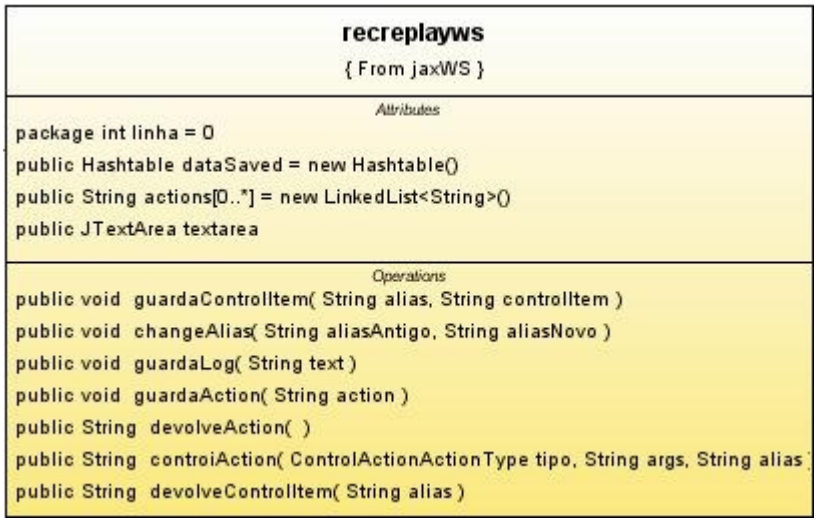

Figura 12: Diagrama da classe recreplayws

Como podemos ver, temos sete operações definidas:

- Três delas para gravar nas variáveis os dados, onde uma diz respeito ao LOG da aplicação quando é efectuada a reprodução, e as duas restantes para as instruções e ControlItem. É nesta ultima que é usado a API do JAXB para converter o XML recebido em objecto e guardar.

- Uma operação para alterar um *alias* que já foi definido. Só é possível fazê-lo durante a fase de gravação.

- Uma operação para disponibilizar um ControlAction. Por sua vez este método utiliza um outro que constrói um objecto daquele tipo devolvendo-o, para que então seja feito o "marshal" desse mesmo objecto. Este método é chamado sucessivamente retornando null quando todas as acções forem enviadas.

- Por último, foi definido também uma operação para que seja devolvido um ControlItem, através do seu *alias*.

Como atributos, os dois mais importantes serão a hashtable e a linkedlist. Foi optado por usar uma hashtable para guardar o *alias* e o ControlItem, pela rapidez de acesso aos dados que a estrutura permite e pela sua característica de associar dois valores, ou seja, com ela podemos

associar o *alias* ao controlItem. Por sua vez a linkedlist será usada para guardar as instruções do script.

4.3 Módulo Launcher

Este é um dos dois módulos da aplicação. A aplicação desenvolvida estará dividida em 2 JARs diferentes para que a aplicação a ser testada possa ser iniciada numa JVM diferente. O primeiro JAR será exactamente o que corresponde ao módulo Launcher que será responsável por manter o Web Service, janela de log e a interface para introdução dos dados da aplicação a ser testada. Abaixo podemos encontrar o diagrama com as dependências das 3 classes principais neste módulo,

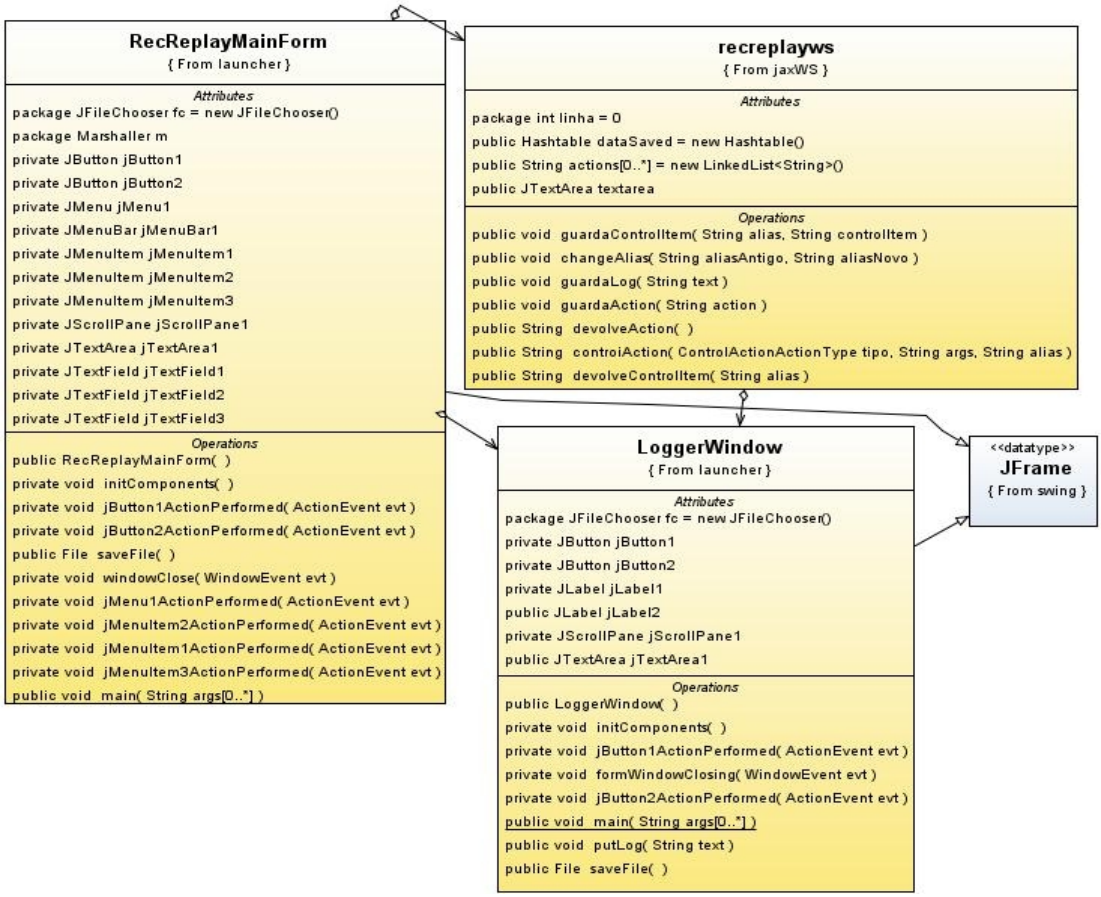

Figura 13: Diagrama de classes e dependências no módulo Launcher

Existe uma outra classe criada para efeitos da integração que servirá apenas para o lançamento da gravação ou de reprodução consoante seja passado ou 0 ou 1, respectivamente. A aplicação a ser testada irá ser lançada apenas quando forem executados estes métodos e portanto a sua descrição será feita nas próximas secções.

A razão da existência desta classe mesmo quando o produto for integrado, prende-se com o facto de ser conveniente a AUT ser lançada no directório onde está instalada. Foi chegada a esta conclusão após experiências com AUT que utilizavam ficheiros do disco rígido, que estariam na directoria onde a AUT estava instalada e através da base de dados de bugs da Sun. Este assunto será abordado na secção 4.5 que diz respeito a problemas encontrados durante a implementação.

Assim este módulo irá ser usado para lançar o método main das duas classes do segundo módulo (Classe gravação e Classe Reprodução) que serão responsáveis então por lançar a AUT e efectuar o resto das funções da aplicação.

Para o lançamento da gravação ou reprodução é criado um processo recorrendo a API de Java, sendo usado o método:

- **ProcessBuilder(List<String> command)**

A criação deste processo, é a criação de um novo processo Java e portanto será lançado numa JVM diferente da que o "Launcher" está a ser executado. Quando o processo é criado é definida a directoria onde este será lançado, sendo dado como argumentos a directoria da AUT, o seu classpath e a classe que contém o main da aplicação. Tudo isto é recolhido através de campos da interface gráfica da aplicação desenvolvida.

O lançamento da aplicação de gravação ou reprodução é o papel mais importante desta classe, no entanto, disponibiliza ainda a gravação de scripts e das informações de componentes recolhidas, para que mais tarde se possa fazer o carregamento do teste. Disponibiliza também a classe "LoggerWindow" que é onde será registado o desenrolar de acontecimentos da reprodução do teste.

4.4 Módulo Gravação/Reprodução

Este é o módulo principal da aplicação, é onde estão as funcionalidades mais importantes. Nas próximas secções iremos ficar a saber detalhadamente como todo este processo funciona, desde o lançamento da aplicação até a fase de reprodução de eventos.

4.4.1 Lançamento da AUT

O lançamento da AUT é feito através de reflexão. Conseguimos assim um grande controlo sobre a aplicação. Para conseguir um lançamento bem sucedido da aplicação é preciso utilizar um classloader disponibilizado pela API de Java para então definirmos o classpath da aplicação.

Para efectuar o lançamento teremos então a seguinte ordem de passos:

– Criar array de URLs (variavel do tipo URL definido na API de Java) com o classpath enviado pelo módulo laucher.

- Carregar para uma variável o ClassLoader do sistema, recorrendo para isso ao método getSystemClassLoader().
- Carregar para o ClassLoader os dados contidos na variável criada no primeiro passo.
- De seguida bastará usar o método LoadClass(string className) do classloader obtido para carregar a classe onde está contido o main da AUT
- Fazer uso da API de reflexão para invocar o método main da AUT.

No inicio é necessário também carregar a fila de eventos costumizada para que se possa então controlar os eventos da aplicação. Para isso é criada uma instância do tipo EventsManager que estende a classe Eventqueue. Mais detalhe sobre esta classe será dado na próxima subsecção, uma vez que ela representa um dos papéis principais da gravação.

Para carregarmos esta fila de eventos costumizada, usaremos a seguinte linha de código:

- **Toolkit.getDefaultToolkit().getSystemEventQueue().push(**instanciaASerCarregada**);**

A aplicação está neste momento pronta para iniciar a gravação dos eventos.

4.4.2 Gravação

A gravação englobará duas grandes classes. Nos diagramas abaixo podemos ver as classes representadas, algumas das operações definidas para elas.

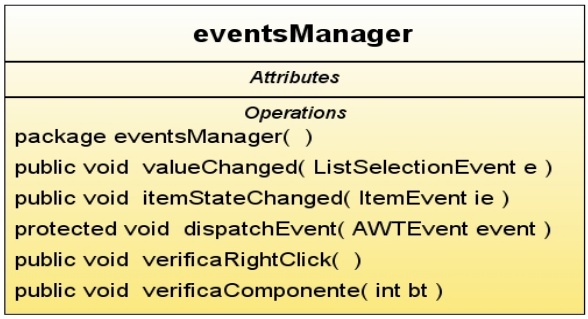

Figura 14: Diagrama da classe eventsManager

A classe eventsManager é uma das principais no processo de gravação. Ela vai substituir a fila de eventos do sistema e todos os eventos terão de passar por ela. Dentro do método dispathEvent está definido o tratamento dos eventos, em que irá ser feita a verificação do tipo de evento recebido (se é evento de rato ou teclado).

O tratamento do evento é feito consoante o botão do rato pressionado funcionando da seguinte forma:

Botão esquerdo → é guardado o evento, caso exista um evento semântico associado ao componente;

Botão centro → é aberto o formulário para guardar definir o *alias* do componente clicado;

Botão direito → Verifica se o componente clicado terá alguma função para este tipo de clique e se tiver guarda a informação.

Na imagem abaixo podemos ver um excerto de código onde se processa toda esta verificação:

```
197         if(event.getID()== MouseEvent.MOUSE_PRESSED)
198               {
199   window = mouse.getComponent();
200   target = SwingUtilities.getDeepestComponentAt(mouse.getComponent(), mouse.getX(), mouse.getY());
201   mouse = SwingUtilities.convertMouseEvent(mouse.getComponent(), mouse, target);
202   mousePre = mouse;
203   comp.pontoPress = mouse.getPoint();
204
205                   if(aliasS!= null)
206                       aliS = aliasS.isVisible();
207
208                    if(mouse.getButton() == 1 && aliS ==false)
209                      {
210                        botaoClick=1;
211                        verificaComponente(1);
212                      }
213
214                     if(mouse.getButton()==2)//abrir cena alias
215                       {
216                           try {
217                               verificaComponente(2);
218                               comp.mostraGravaAlias(aliasS, target);
219                           } catch (Exception ex) {ex.printStackTrace();}
220                       }
221
222                       if(mouse.getButton()==3)//rightclick
223                       {
224                           verificaRightClick();
225                       }
226           }
```

Figura 15: Exemplo das operações efectuadas no tratamento de um evento MOUSE_PRESSED

Como podemos ver, começamos por tratar o evento recebido de modo a que o componente e coordenadas estejam correctamente adicionadas ao mesmo e é então verificado qual o botão do rato clicado para o direccionar para o método de tratamento.

Dentro destes métodos o tratamento é bastante semelhante só que desta feita, o que se tenta identificar é o componente que está presente no evento e é conseguido isso da seguinte forma:

```
470   if(mouse.getComponent() instanceof JTable)
471   {
472       comp.gravaJTable(mouse);
473   }
```

Figura 16: Identificação do componente do evento

No método de verificação existem vários if´s semelhantes, um para cada tipo de componente. Quando algum é encontrado é então chamado um método da classe componentsSave. Nesta classe estão definidos diversos métodos para tratamento de cada um dos tipos de componentes existentes.

Todo o processo descrito acima é cíclico, parando apenas quando a AUT é fechada.

Abaixo podemos ver o diagrama da classe componentsSave, com algumas das operações definidas nela:

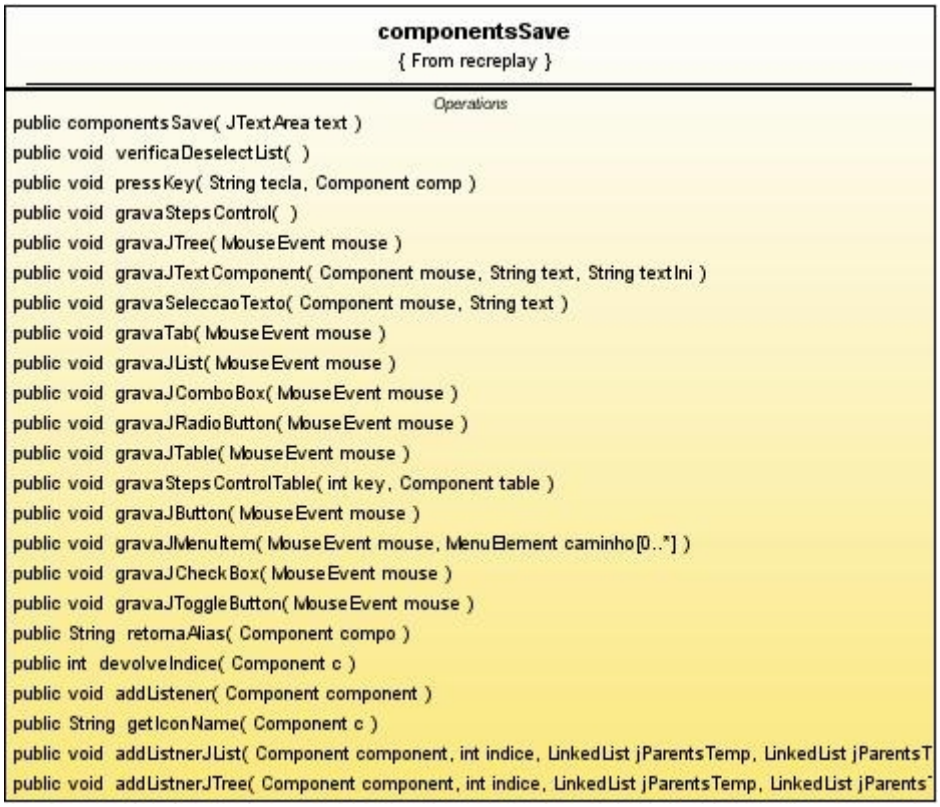

Figura 17: Diagrama da classe componentsSave com os métodos mais relevantes

O processo de gravação de eventos inclui também um tipo semelhante ao ControlItem que é guardado durante a gravação. Iremos ter uma LinkedList com vários elementos deste tipo em que cada um conterá a referência para o componente e o seu *alias*, para que quando seja pedido

o adicionar de um novo evento semântico, não seja enviado um ControlItem novamente caso o componente já tenha sido guardado, evitando assim duplicação de dados.

A decisão de manter uma estrutura com as referências para os componentes já enviados deve-se ao facto de que, com as referências, pode comparar-se facilmente dois componentes e saber se eles são o mesmo, o que não acontece com a estrutura ControlItem utilizada.

O processo de gravação de acção de uma maneira geral é efectuado sempre da mesma forma:

1. Invoca o método **addlistener(Componente)**
   - Testa se o componente já foi adicionado a lista, se não foi constrói o objecto com a hierarquia dos componentes até ao componente em questão, se já foi adicionado salta para o passo 2;
   - Testa o tipo de componente para saber que tipo de informações irá guardar no ControlItem, e constrói o objecto;
     - Consoante o tipo de objecto são adicionados listeners ao componente para que as acções sobre determinado componente sejam registadas por esse listener e enviadas pelo Web Service.
   - Envia o objecto através do Web Service;

2. Para o caso dos componentes que não tenha sido adicionado listener é construída uma string que respeita a linguagem que já foi apresentada e enviada através do Web Service.

A opção de adicionar listener aos componentes do tipo JcomboBox, Jlist, Jtree foi tomada para facilitar o acesso a alterações de selecção nos mesmos componentes. Usando o tratamento de eventos do Java é, por exemplo, mais simples identificar uma selecção num elemento de uma lista e construir assim a string da acção.

Este será todo o processo de gravação de componentes em que cada acção será enviada quando esta ocorre.

Para além deste processo de gravação existe também a possibilidade de adicionar os ControlItem sem que seja registada qualquer acção sendo para isso necessário apenas clicar sobre o componente com o botão do meio do rato. Quando é efectuado este clique o método addlistener(Componente) é invocado (o componente fica logo guardado e é atribuído um alias genérico) e abrirá um formulário onde se poderá alterar o alias. De notar que, neste ponto, o ControlItem já estará guardado e a única alteração que iremos fazer é sobre o alias genérico.

4.4.3 Reprodução

A reprodução é um processo cíclico em que estará sempre a pedir uma acção ao Web Service até que lhe seja retornado um valor nulo. Nesse ponto a reprodução é terminada. Na figura abaixo podemos ver o funcionamento do processo de gravação:

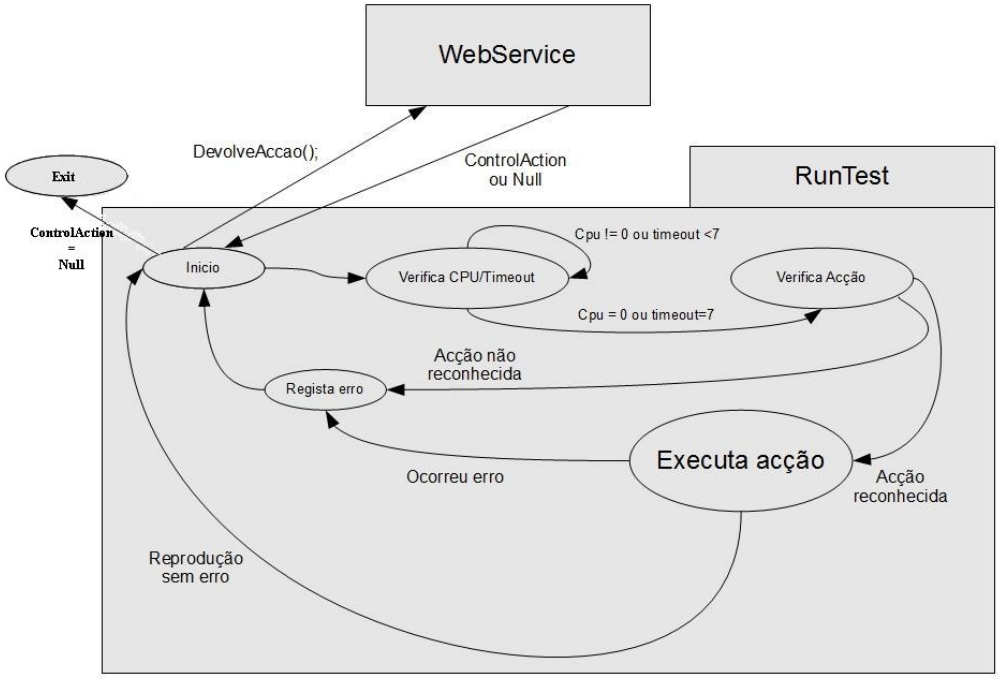

Figura 18: Estados existentes na operação de reprodução

Como podemos ver o processo é iniciado sempre com um pedido ao Web Service. No retorno irá ser devolvido um objecto do tipo controlAction para que então se possa executar a acção. Quando o controlAction é obtido é separado o argumento e o item, que nele está incluído, e verificado qual a acção que está descrita. Esta verificação é feita recorrendo a Classe Scanner da API de Java que permite efectuar o reconhecimento de uma string usando expressões regulares. Se de facto a expressão estiver definida na classe (que foi descrita no ponto 4.1) é devolvido o índice que se situa no array, para então depois este ser passado a um Switch e então direccionar a aplicação de reprodução para o excerto de código necessário para executar aquela acção. Existem diversas operações definidas para a reprodução como podemos ver no diagrama da classe RunTests que está abaixo, nele estão representadas as operações principais para a reprodução:

Figura 19: Diagrama da classe runTests

Como podemos ver existem métodos para todos os componentes, componentes como por exemplo uma CheckBox não está definido nenhum método, uma vez que, como este componente deriva da classe AbstractButton, ele é tratado como se fosse um botão normal.

Assim sendo, o processo de reprodução apesar das diferenças dos componentes respeita sempre uma ordem, ordem essa que podemos verificar no diagrama representativo da reprodução de uma acção genérica que esta abaixo:

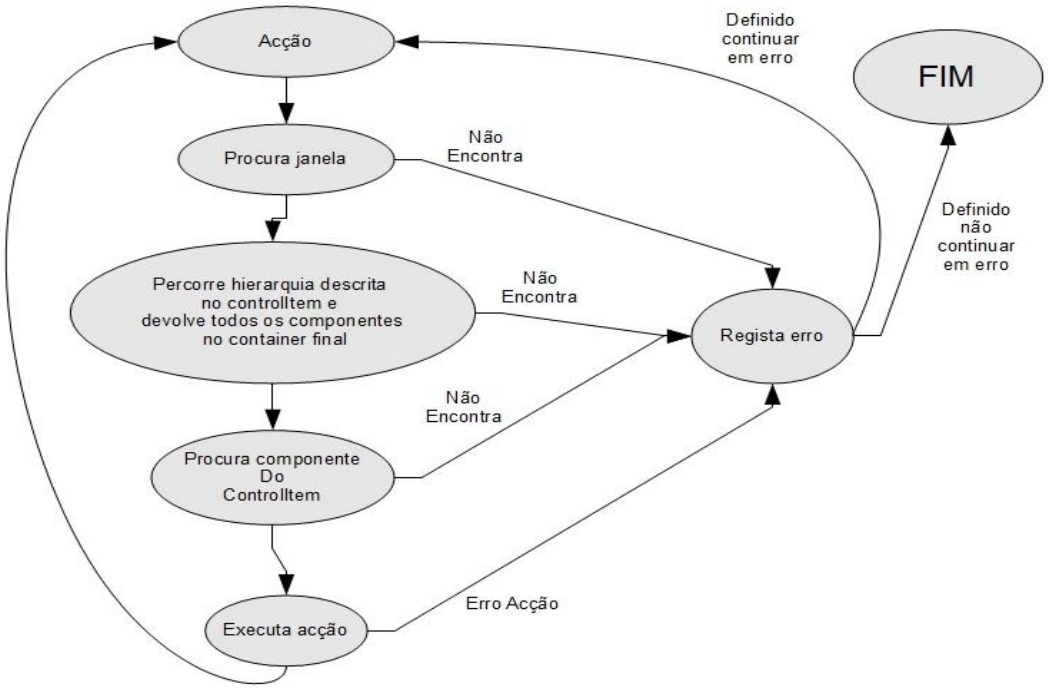

Figura 20: Operações realizadas na reprodução de uma instrução

Como podemos ver no processo de reprodução, quando ocorre algum erro estes são sempre registados, e enviados para a janela de erros através do Web Service.

Exemplos de erros que poderão ocorrer serão:

- Componente não estar localizado no ecrã (fora dos limites ou a janela estar minimizada);
- Não serem encontrados nenhum ou algum dos componentes descritos no ControlItem.

Para além de erros poderão ser também emitidos alguns warnigs que indicarão que o componente devolvido não terá sido identificado com todas as informações disponíveis, terá sido só identificado pela sua classe e pelo seu índice dentro do container.

4.5 Problemas durante a implementação

Durante a implementação foram encontrados alguns problemas até a chegada da solução. Abaixo descreve-se os mais relevantes, indicando também a solução adoptada quando foi chegada a uma.

Situação 1

→ Falha na obtenção da localização do botão de expansão de uma Jtree.

Para solucionar esta situação foi optado definir a localização deste botão no código, ou sejam, localização será "hardcoded" e sempre relativamente a sua posição dentro de cada JTree.

Situação 2

→ A obtenção dos filhos de um certo componente nem sempre é efectuada com o mesmo método. Um exemplo disso é um componente dentro de uma JViewPort em que o método para o obter difere de quando ele esta dentro de um JPanel.

Para contornar esta situação irá ser testado, ao realizar a procura do componente, se este tipo de situação se verifica e usar o método necessário.

Situação 3

→ Chamada ao system_exit() faz a JVM encerrar em qualquer momento da execução.

A solução para contornar este problema passa por instalar um Security Manager para que toda a informação possa ser tratada antes da JVM encerrar.

Situação 4

→ No caso de uma JCombobox manter sempre o índice zero, quando existe uma selecção na mesma (para passar um valor para um jtextarea por exemplo) são despoletados dois eventos o da selecção do índice e o da JCombobox a voltar a selecção antiga automaticamente.

Para solucionar o problema é necessário verificar se a JComboBox tem esta opção activa e em caso afirmativo, filtrar o segundo evento.

Situação 5

→ Numa Jlist, quando temos possibilidade de reordenar as linhas a cada mudança de índice (trocar a linha 3 pela 2), recebemos também um evento de mudança de índice seleccionado.

Contorna-se esta situação filtrando o evento de selecção, para que na reprodução não tenhamos passos desnecessários. Verifica-se se a lista foi clicada quando existiu a mudança de selecção antes de guardar o evento.

Situação 6

→ Propriedade de User.Dir é apenas de leitura, ou seja, sendo alterado em tempo de execução não terá qualquer efeito porque esta variável só é consultada quando a JVM é iniciada[referência ao bug]. Se a AUT utilizar caminhos relativos para localização dos seus ficheiros de configuração ocorrerão problemas pois a JVM irá ser iniciada com o caminho para a aplicação que irá testar.

Para solucionar esta situação e visto não se poder alterar nada na AUT, molda-se a aplicação em desenvolvimento para esta situação. Inicia-se um processo programaticamente com o directório da AUT e caso seja necessário o directório raiz da aplicação que está a ser desenvolvida trabalha-se com o classpath.

Situação 7

→ Componentes custom

Neste caso a solução nem sempre é eficaz, componentes custom são tratados da mesma maneira que os componentes Java Swing, é verificado se são instâncias de alguma classe e partindo desse ponto é dado o tratamento necessário a informação. Não derivando de uma classe que possa dar algum tipo de identificação sobre o componente (Exemplo: em vez ser uma instância de uma JList, o componente JListCustom foi desenvolvido derivando da classe Component) até ao momento não foi pensada nem implementada nenhuma estratégia para resolver estes casos.

Situação 8

→ Timing para a realização das acções poderá ser variável,sendo difícil saber se o componente descrito na acção está visível.

Para contornar este problema optou-se por verificar o uso de CPU, continuando para a próxima acção quando o tempo de CPU gasto pela aplicação for zero, existe também um timeout de 7 segundos para a espera sobre o uso do CPU para prever alguma operação que esteja a ser realizada em BACKGROUND, sendo portanto possível passar a próxima acção. Existe também a instrução "wait on alias", que servirá para parar os testes enquanto uma JProgressBar não mostra o final de uma operação.

4.6 Avaliação da solução

Para avaliar a robustez da aplicação criada foram usadas duas aplicações. Procurou-se utilizar aplicações que mostrassem algum desafio e que se pudesse verificar se realmente os diversos objectivos foram cumpridos.

No entanto, a avaliação da aplicação é de facto algo difícil, uma vez que, para poder testar correctamente era necessário ter acesso total as aplicações para alterar o código e verificar se o comportamento seria o esperado. É por isso difícil saber se a aplicação reagirá bem a qualquer situação que lhe seja imposta mas, na teoria, o resultado esperado é positivo. Apesar de uma das aplicações utilizadas ser open source, uma AUT foi criada, para efeitos de testes, para que então se tivesse acesso total ao seu código e fosse simples a sua alteração e manipulação. Esta AUT serviu na ajuda do desenvolvimento da ferramenta mas, no entanto, a sua simplicidade não

permite avaliar com toda a certeza a robustez da ferramenta. Assim, os resultados obtidos com esta AUT foram positivos mas excluídos desta secção.

No entanto, de uma maneira geral, é seguro afirmar que a aplicação desenvolvida comporta-se como o esperado. Existem, no entanto, situações de erro que poderão ocorrer. Estes erros, que já são usuais em ferramentas deste tipo foram minimizados mas não eliminados totalmente. Exemplo disso será uma identificação de um componente em que, do seu ControlItem, só conseguiremos saber a classe a que pertence, condicionando a sua identificação.

4.6.1 Aplicações testadas

Duas aplicações foram testadas para comprovar a robustez da ferramenta:

Jabref - Aplicação para gerir referências bibliográficas. Ferramenta open source desenvolvida em Java. Esta aplicação foi inicialmente escolhida devido a utilização de componentes costumizados e também devido a utilização de componentes em que a sua informação era mínima, nomeadamente, botões em que não existia sequer texto para os identificar e só teriam uma imagem associada.

Centaur - É um sistema de gestão de fraude (SGF), especialmente concebido para operadoras de telecomunicações que exigem rápida integração, dos seus actuais, novos serviços e plataformas com o SGF. Esta já é uma aplicação bastante complexa, com utilização de componentes costumizados, frames flutuantes, componentes com pouca informação e aos seus tempos de carregamento variáveis devido a acessos a bases de dados.

Com o uso destas duas aplicações foi então possível verificar se os objectivos foram cumpridos. Mas como referido anteriormente não é possível afirmar com toda a certeza que erros não ocorrerão, seria necessário despender de muito tempo e efectuar centenas de testes de modo a que todas as situações possíveis fossem cobertas e que então pudéssemos afirmar que a aplicação irá reagir bem a qualquer tipo de situação.

4.6.2 Exemplo de teste

Sendo impossível reproduzir aqui todos os testes efectuados, foram escolhidas duas situações para demonstrar um teste bem sucedido e um teste em que alguns erros iriam ser impressos no log.

Foi gravado um pequeno script de teste para demonstrar o funcionamento da aplicação, neste caso o script foi gravado interagindo-se com o GUI e o resultado foi o seguinte:

click on Jbutton0

click on JButton1

write "a" to JTextComponent2

write "b" to JTextComponent3

write "c" to JTextComponent4

select 1 from JTabbedPane5

write "d" to JTextComponent6

write "e" to JTextComponent7

write "f" to JTextComponent8

close Window9

click on Jbutton10

Neste caso a aplicação usada para testes foi o jabref, abaixo teremos o exemplo do log registado e foram introduzidas duas instruções para demonstrar dois dos tipos de erro que poderão ocorrer durante a execução dos testes.

- **Situação 1**

Na execução dos testes esperava-se uma execução correcta e sem erros e como podemos ver pela imagem do log, abaixo, toda a execução ocorreu sem problemas:

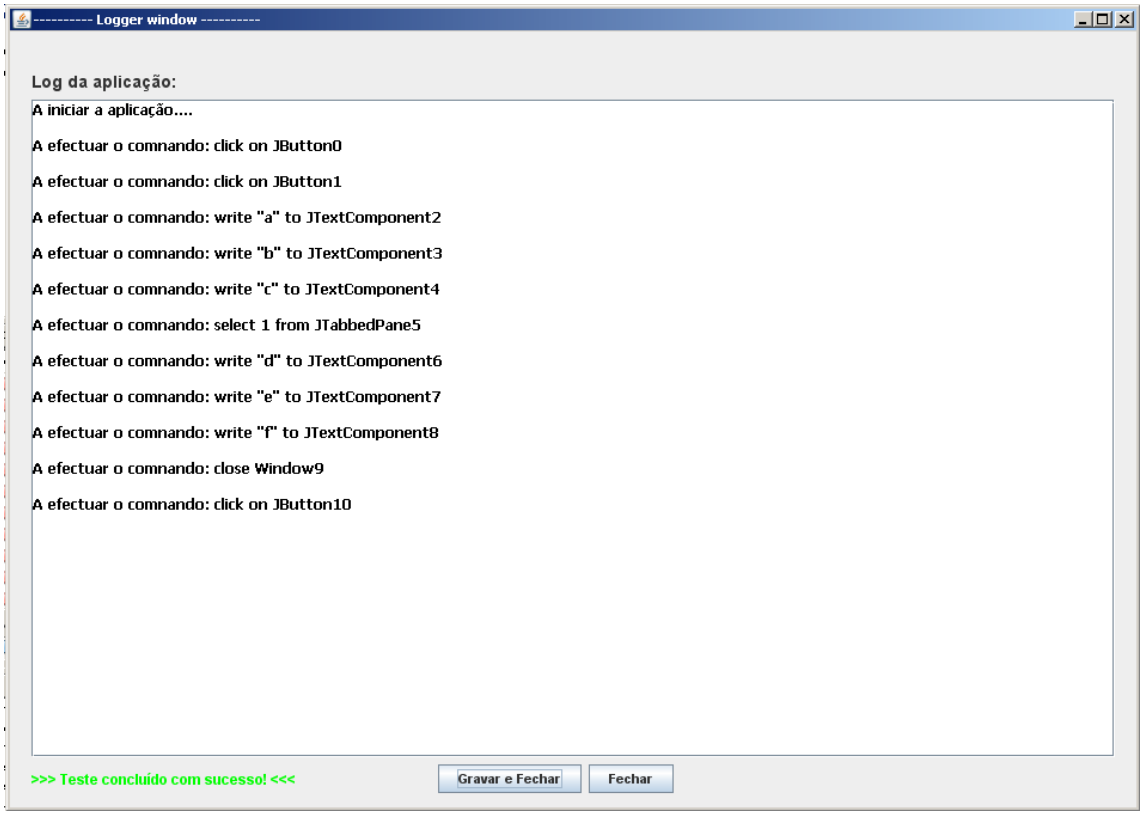

Figura 21: Log de uma reprodução bem sucedida

Este será então o aspecto da janela de log da aplicação e como podemos ver não aparece nenhuma mensagem de erro no log que indica que a reprodução ocorreu sem erros nenhuns. Aparece uma mensagem no canto inferior esquerdo, que fará o status geral da reprodução indicando se o teste foi bem sucedido ou não. Abaixo teremos uma outra situação que nos mostrará um exemplo de uma execução com erros.

- **Situação 2**

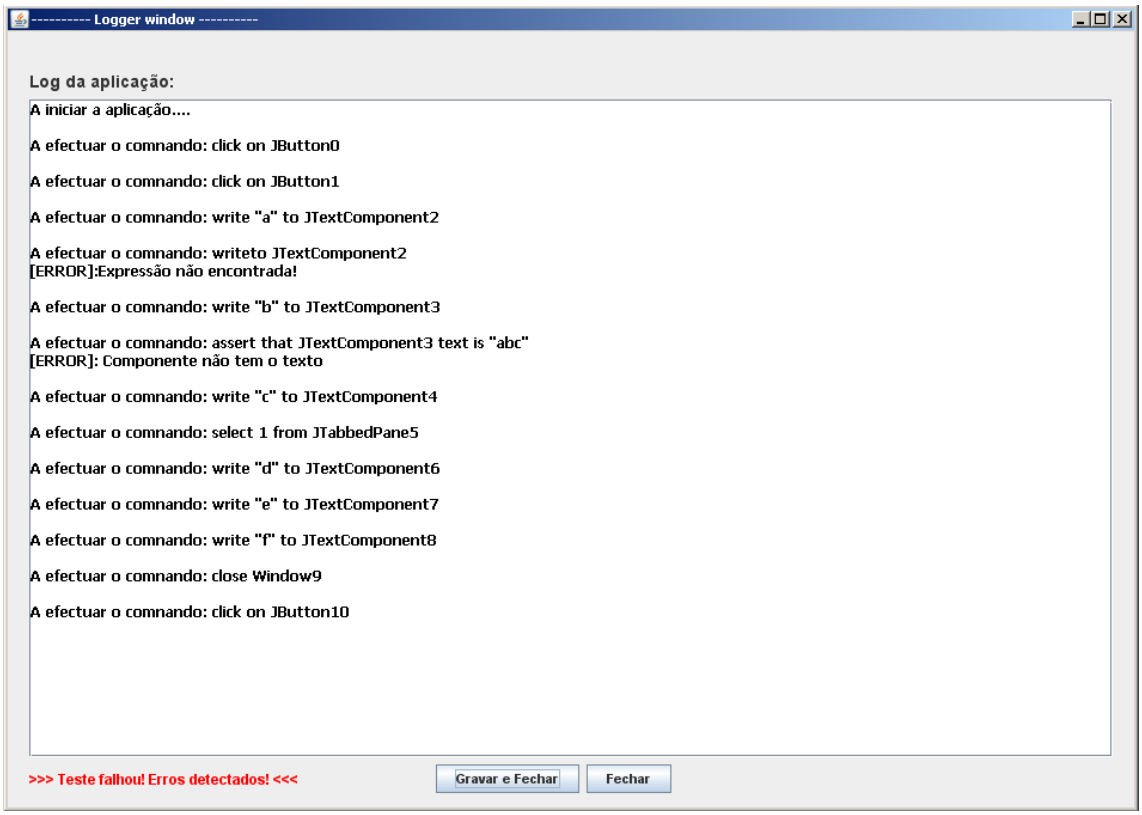

Figura 22: Log de uma execução com erros

Nesta situação, foram introduzidas duas instruções ao script anterior, uma com erro e outra de assert. Como podemos ver ambas falharam, mas mais importante será a instrução de assert que na tentativa de verificar se o componente continha o texto indicado produz um erro. Isto poderia indicar uma situação em que algo na execução não correu como esperado, portanto o teste teria de falhar.

Os testes da aplicação Centaur, são executados da mesma forma, no entanto, não se poderá comprovar visualmente que um dos grandes desafios desta aplicação, os tempos de loading. Desafios deste género não são comprováveis textualmente, nem por imagens, seria necessário uma demonstração prática para se ver realmente que a aplicação minimizou um dos grandes problemas deste tipo de aplicações, a sincronia de acções. Assim sendo mostrou-se

apenas o exemplo do script gerado para o JabRef, uma vez que, para o Centaur iria ser gerado um semelhante e não traria nada de novo disponibiliza-lo aqui.

4.7 Resumo

Neste capítulo foi dado a conhecer o log da aplicação de testes, sendo esta, provavelmente, uma das partes mais importantes. Ao testador só interessará o resultado final dos testes descrito pelo log. A intenção desta aplicação é mesmo tornar o processo de interacção com o GUI secundária interessando só mesmo o log da aplicação que irá dar indícios se algo correu mal.

Existem vários erros que podem ser detectados para além dos demonstrados que passam, por exemplo, por erros produzidos devido a não localização de um componente (as razões que levam a este tipo de erro poderão ser várias, uma mudança de localização no GUI, de um componente, sobre o qual as informações guardadas sejam insuficientes para o detectar após essa mudança ou então um erro qualquer no GUI que impeça o componente referido na instrução de ficar visível) ou então componentes que não se encontram nos limites do ecrã.

De uma maneira geral qualquer situação de erro que possa acontecer será marcada no log, durante a execução, para posterior análise.

5 Conclusões e Trabalho Futuro

Todos os objectivos foram cumpridos e apenas um deles tem uma pequena anotação como poderemos ver no decorrer desta secção. Os testes efectuados a aplicação desenvolvida foram efectuados com sucesso usando-se para isso aplicações diferentes que pudessem oferecer algum desafio a esta aplicação para que se pudesse apurar o grau de cumprimento dos objectivos.

São apresentados neste capitulo também algumas ideias para um trabalho futuro no que diz respeito a continuação deste projecto.

5.1 Satisfação dos Objectivos

Os objectivos apresentados no capítulo 1 são aqui novamente referidos, para que se possa fazer uma avaliação, ponto a ponto, do grau de cumprimento dos mesmos.

- *"Especificação e implementação de um motor de gravação e execução de testes a aplicações desenvolvidas em Java Swing"*, esta meta foi concluída com sucesso. É o resultado final do produto e como mostrado no ponto anterior a aplicação está funcional.

- *"Permitir a execução automática de testes a aplicações via instrumentação do GUI"*, objectivo concluído com sucesso, sendo criado um motor para a gravação e reprodução de interacções com o GUI.

- *"Facilitar a execução de testes de regressão minimização do custo de desenho dos testes"*, objectivo também concluído com sucesso, sendo possível guardar todas as interacções que existirem com um GUI, para mais tarde se poderem efectuar os testes de regressão sobre novas versões da AUT.

- *"Maximização da "tolerância" dos testes a alterações no GUI entre várias versões da mesma aplicação"*, para concretizar este objectivo tentou tira-se o máximo de informação possível sobre determinado componente, informação essa que permita identificar um componente mesmo sendo ele mudado de sitio. Este objectivo pode considerar-se completo mas no entanto as alterações no GUI continuam a mostrar-se um problema e em certos casos não será possível encontrar um componente que tenha sido alterado.

- *"Maximização da "tolerância" dos testes a alterações nos dados tratados pela aplicação"*, este ponto está directamente relacionado com o anterior, tendo portanto, as mesmas limitações.

- *"Gravação selectiva de eventos, filtrando eventos que não produzem qualquer efeito/alteração no GUI"*, objectivo concluído com sucesso, sendo verificado se um componente tem atribuído um evento semântico e caso não o tenha essa acção não será gravada.
- *"Interoperabilidade"*, este objectivo também foi concluído com sucesso com o uso de web services.

5.2 Trabalho Futuro

No que diz respeito a uma possível continuação deste projecto existem alguns objectivos adicionais que deveriam ser abordados com mais profundidade.

Existem diversos pacotes com componentes costumizados para aplicações em Java e em certos casos a identificação de um componente pode estar comprometida, já que um botão costumizado poderá derivar directamente da classe Component, não passando pela hierarquia de classes que um JButton passará, nestes casos a identificação deste componente como sendo um botão estará comprometida. Dotar a aplicação de robustez necessária para lidar com situações destas seria uma mais valia.

Relativamente ao método utilizado para sincronização de acções, melhoramentos poderiam ser feitos com a combinação de ainda mais informações sobre a aplicação. Utilizar a ocupação de memória, juntamente com a ocupação de CPU e utilização de rede no caso de aplicações que o façam seriam possíveis opções e conjugar estas e outras opções iria certamente trazer melhorias à sincronização.

Na aplicação é sempre procurado cada componente referenciado numa instrução, mesmo que esse componente seja usado em duas acções consecutivas. Durante a reprodução, um mecanismo de cache poderia ser implementado onde se guardaria um histórico de componentes já procurados, para que não tenha de existir uma nova procura na hierarquia de componentes do GUI, para referências ao mesmo componente.

Referências

[1]: Atif M. Memon Computer, GUI Testing: Pitfalls and Process, 2002

[14]: J. Watkins, Testing IT: an off-the-shelf software testing process, 2000

[17]: I. K. El-Far and J. A. Whittaker, Model-Based Software Testing. Encyclopedia of Software Engineering, 2001

[11]: British Computer Society Specialist Interest Group in Software Testing, Standard for Software Component Testing, Working Draft 3.4, 27 April 2001

[9]: Yoonsik Cheon, Myoung Yee Kim, Ashaveena Perumandla, A Complete Automation of Unit Testing for Java Programs, February 2005; revised July 2005

[12]: Ian Sommerville, Software Engineering, 8th edition, 2006

[16]: FEUP CIQS, Software Test Management, 2008

[13]: P. C. Jorgensen, Software Testing – a Craftsman Approach, 1995

[8]: Jessica Chen, Suganthan Subramaniam, Specification-based Testing for GUI-based Applications, 2002

[18]: IEEE, ANSI/IEEE Standard 1008-1987, IEEE Standard for Software Unit Testing, 1987

[15]: laurie williams, Testing Overview and Black-Box Testing Techniques, 2006

[19]: FEWSTER, M., GRAHAM D., Software Test Automation, 1999

[20]: PETTICHORD, Bret, Success with Test Automation, 2001

[21]: KANER, Cem., Improving the Maintainability of Automated Test Suites, 1997

[7]: Lee White,Husain Almezen, Generating Test Cases for GUI Responsibilities Using Complete Interaction Sequences, 2000

[2]: Michael Silverstein, , Nov/Dec 2003

[23]: Arie van Deursen - Paul Klint - Joost Visser , Domain-Specific Languages:An Annotated Bibliography,

[24]: Mark Dalgarno, Matthew Fowler, UML vs. Domain-Specific Languages, 2008

[22]: Davi Cunha, Web Services, SOAP e Aplicações Web, 10 dez 2002

[25]: S.P Ng, T. Murnane, K. Reed, D. Grant and T.Y Chen , A preliminary survey on software testing practices in Australia, 2004

Anexo A: Diagrama de gantt

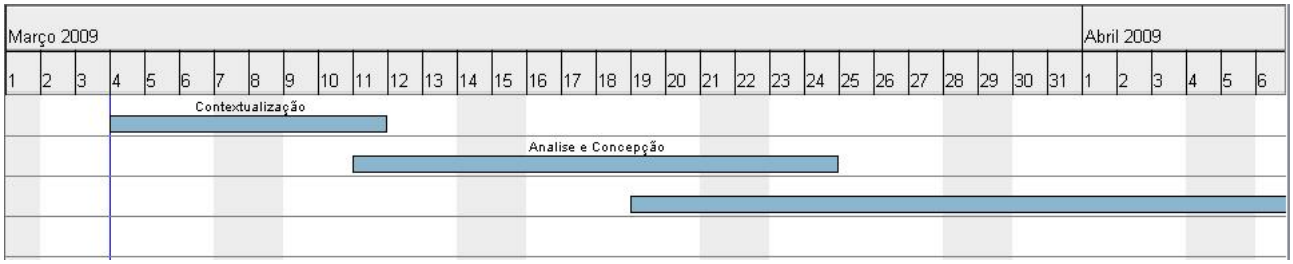

Figura 23: Duas primeiras tarefas e sua duração

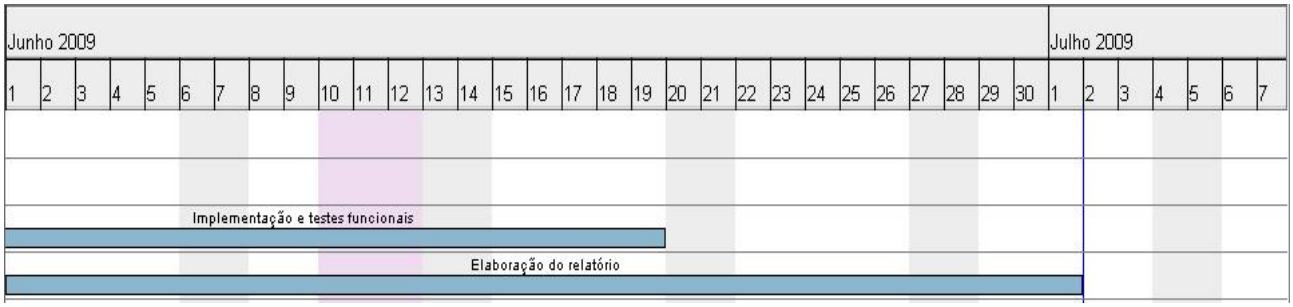

Figura 24: Duas ultimas tarefas do projecto

O mês de Abril e Maio foi omitido porque no decorrer desses dois meses só existiu a fase de implementação e testes funcionais.

Anexo B: Peso dos testes no orçamento de um projecto

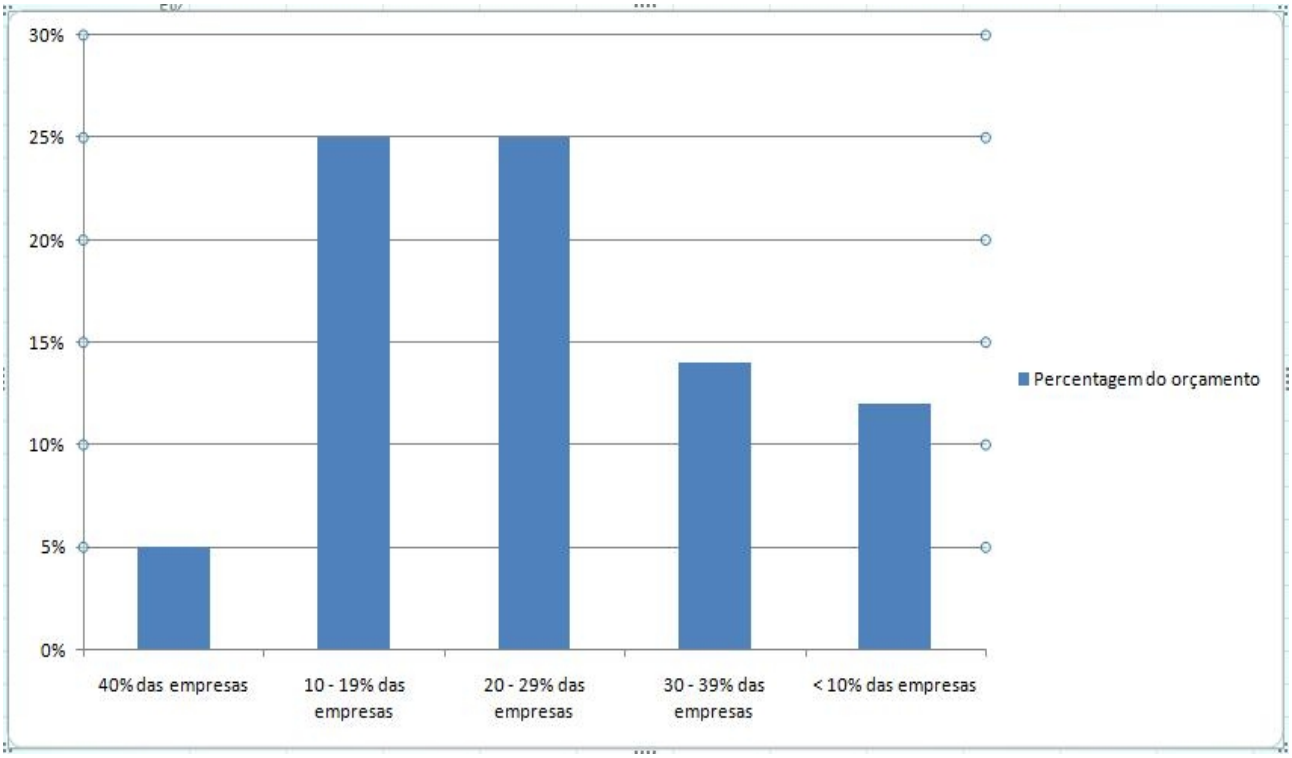

Figura 25: Percentagem do Orçamento inicial atribuído à fase de testes

Dados obtidos de um inquérito[25] efectuado na Austrália, encontrou os valores demonstrados no gráfico para a alocação do orçamento inicial para o teste de software.

Anexo C: Instruções e diferenças com as disponíveis anteriormente

#	Instrução	Definição
1	**click on** alias	Já definida em C#, foi usado o mesmo conceito em Java
2	**click on** "item"**@**alias	Já definida em C#, foi usado o mesmo conceito em Java
3	**select** "nome" **from** alias	Já definida em C#, foi usado o mesmo conceito em Java, acrescentando a possibilidade de múltiplas selecções separando por espaços os nomes
4	**deselect** "nome" **from** alias	Criada apenas na parte Java que servirá para desseleccionar itens seleccionados com a instrução anterior. Existe a possibilidade de múltiplas selecções separando por espaços os nomes
5	**select** 234 **from** alias	Já definida em C#, foi usado o mesmo conceito em Java, acrescentando a possibilidade de múltiplas selecções separando por vírgulas os índices
6	**Deselect** 234 **from** alias	Criada apenas na parte Java que servirá para desseleccionar itens seleccionados com a

		instrução anterior. Existe a possibilidade de múltiplas desselecções separando por espaços os índices
7	**select** (rowIdx,colIdx) **from** alias	Já definida em C#, foi usado o mesmo conceito em Java, acrescentando a possibilidade de múltiplas selecções separando por espaços os pares índice/coluna
8	**Deselect** (rowIdx,colIdx) **from** alias	Criada apenas na parte Java que servirá para desseleccionar itens seleccionados com a instrução anterior. Existe a possibilidade de múltiplas desselecções separando por espaços os pares índice/coluna
9	**select** text **in** alias	Já definida em C#, foi usado o mesmo conceito em Java
10	**write** "text" **to** alias	Já definida em C#, foi usado o mesmo conceito em Java
11	**press** ENTER(@2)? **in** alias	Já definida em C#, foi usado o mesmo conceito em Java
12	**press** 'c' **in** alias	Já definida em C#, foi usado o mesmo conceito em Java
13	**press** TAB	Já definida em C#, foi usado o mesmo conceito em Java
14	**assert that** alias **text is** "whatever"	Já definida em C#, foi usado o mesmo conceito em Java
15	**assert that** alias **has** 34 **elements**	Já definida em C#, foi usado o mesmo conceito em Java
16	**assert that** alias **has element** "sed"	Já definida em C#, foi usado o mesmo conceito em Java
17	**log text of** alias	Já definida em C#, foi usado o mesmo conceito em Java
18	**log selected item in** alias	Já definida em C#, foi usado o mesmo conceito em Java
19	**log number of items in** alias	Já definida em C#, foi usado o mesmo conceito em Java
20	**capture**	Já definida em C#, foi usado o mesmo conceito em Java
21	**drag (**“item”**@)?**sourceAlias **to (**“item”**@)?** targetAlias	Já definida em C#, foi usado o mesmo conceito em Java
22	**rightclick on** alias	Já definida em C#, foi usado o mesmo conceito em Java
23	**rightclick on** "item"**@**alias	Já definida em C#, foi usado o mesmo conceito em Java
24	**close**	Já definida em C#, foi usado o mesmo conceito em Java
25	**maximize**	Já definida em C#, foi usado o mesmo

		conceito em Java
26	**minimize**	Já definida em C#, foi usado o mesmo conceito em Java
27	**assert that** chkBox **is not? selected**	Já definida em C#, foi usado o mesmo conceito em Java
28	**wait** 12345	Já definida em C#, foi usado o mesmo conceito em Java
29	**wait on** alias	Já definida em C#, foi usado o mesmo conceito em Java

Anexo D: Esquema em XML das estruturas

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
xmlns:tns="http://schemas.datacontract.org/2004/07/Telbit.TeStudio.GUITesting.Common.Objs"
elementFormDefault="qualified"
targetNamespace="http://schemas.datacontract.org/2004/07/Telbit.TeStudio.GUITesting.Common.Objs"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:import namespace="http://schemas.microsoft.com/2003/10/Serialization/" />
 <xs:complexType name="ControlItem">
  <xs:sequence>
   <xs:element minOccurs="0" name="type" nillable="true" type="xs:string" />
   <xs:element minOccurs="0" name="Name" nillable="true" type="xs:string" />
   <xs:element minOccurs="0" name="Framework" type="tns:ControlItem.FrameworkID" />
   <xs:element minOccurs="0" name="text" nillable="true" type="xs:string" />
   <xs:element minOccurs="0" name="index" nillable="true" type="xs:int" />
   <xs:element minOccurs="0" name="Parent" nillable="true" type="tns:ControlItem" />
  </xs:sequence>
 </xs:complexType>
 <xs:element name="ControlItem" nillable="true" type="tns:ControlItem" />
 <xs:simpleType name="ControlItem.FrameworkID">
  <xs:restriction base="xs:string">
   <xs:enumeration value="NONE" />
   <xs:enumeration value="WINFORMS" />
   <xs:enumeration value="WPF" />
   <xs:enumeration value="JAVA_SWING" />
  </xs:restriction>
 </xs:simpleType>
                    <xs:element name="ControlItem.FrameworkID" nillable="true"
type="tns:ControlItem.FrameworkID" />
 <xs:complexType name="IdentifierConfig.ConfigIdentifier">
  <xs:annotation>
   <xs:appinfo>
    <IsValueType xmlns="http://schemas.microsoft.com/2003/10/Serialization/">true</IsValueType>
   </xs:appinfo>
  </xs:annotation>
  <xs:sequence>
   <xs:element minOccurs="0" name="method" type="tns:IdentifierConfig.Method" />
   <xs:element minOccurs="0" name="property" type="tns:IdentifierConfig.Property" />
```

```
      <xs:element minOccurs="0" name="type" nillable="true" type="xs:string" />
     </xs:sequence>
    </xs:complexType>
                    <xs:element name="IdentifierConfig.ConfigIdentifier" nillable="true"
type="tns:IdentifierConfig.ConfigIdentifier" />
    <xs:simpleType name="IdentifierConfig.Method">
     <xs:restriction base="xs:string">
      <xs:enumeration value="Invalid" />
      <xs:enumeration value="ToString" />
      <xs:enumeration value="GetHashCode" />
     </xs:restriction>
    </xs:simpleType>
    <xs:element name="IdentifierConfig.Method" nillable="true" type="tns:IdentifierConfig.Method" />
    <xs:simpleType name="IdentifierConfig.Property">
     <xs:restriction base="xs:string">
      <xs:enumeration value="Invalid" />
      <xs:enumeration value="Tag" />
      <xs:enumeration value="Text" />
      <xs:enumeration value="Name" />
     </xs:restriction>
    </xs:simpleType>
    <xs:element name="IdentifierConfig.Property" nillable="true" type="tns:IdentifierConfig.Property" />
    <xs:complexType name="ControlAction">
     <xs:sequence>
      <xs:element minOccurs="0" name="Args" nillable="true" type="xs:string" />
      <xs:element minOccurs="0" name="Item" nillable="true" type="tns:ControlItem" />
      <xs:element minOccurs="0" name="Type" type="tns:ControlAction.ActionType" />
     </xs:sequence>
    </xs:complexType>
    <xs:element name="ControlAction" nillable="true" type="tns:ControlAction" />
    <xs:simpleType name="ControlAction.ActionType">
     <xs:restriction base="xs:string">
      <xs:enumeration value="NONE" />
      <xs:enumeration value="CLICK" />
      <xs:enumeration value="RIGHTCLICK" />
      <xs:enumeration value="DRAGNDROP" />
      <xs:enumeration value="ENTER_TEXT" />
      <xs:enumeration value="INPUT_CHARKEY" />
      <xs:enumeration value="INPUT_SELECT" />
      <xs:enumeration value="INPUT_DESELECT" />
      <xs:enumeration value="INPUT_SYSKEY" />
      <xs:enumeration value="ASSERT_PROPERTY" />
      <xs:enumeration value="ASSERT_ELEMENT" />
      <xs:enumeration value="ASSERT_LISTCOUNT" />
      <xs:enumeration value="LOG_PROPERTY" />
      <xs:enumeration value="LOG_SELECTED" />
```

```
            <xs:enumeration value="LOG_LISTCOUNT" />
            <xs:enumeration value="MAXIMIZE" />
            <xs:enumeration value="MINIMIZE" />
            <xs:enumeration value="CLOSE" />
            <xs:enumeration value="WAIT" />
            <xs:enumeration value="PAUSE" />
          </xs:restriction>
        </xs:simpleType>
                                <xs:element name="ControlAction.ActionType" nillable="true"
type="tns:ControlAction.ActionType" />
      </xs:schema>
```