UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

Suporte à Construção e Execução Automática de Testes Funcionais Baseados em Interfaces Gráficas

Simone Antunes Correia

(Licenciada)

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e de Computadores

Orientador: Doutor Alberto Manuel Rodrigues da Silva

Júri

Presidente: Doutor António Manuel Ferreira Rito da Silva

Vogais: Doutor João Carlos Pascoal Faria

Doutor Alberto Manuel Rodrigues da Silva

Novembro 2005

Resumo

Este trabalho de investigação aborda a temática de testes automáticos, realizados sobre sistemas com interface gráfica, através de ferramentas do tipo *capture-replay*. Nesta dissertação, propomos um método para geração e execução automática de testes funcionais que se baseia no desenho da interface gráfica do sistema a ser exercitado e permite a redução do esforço de construção e manutenção de testes automáticos.

Para tal, foi desenvolvido o sistema AGEAT (Ambiente para a Geração e Execução Automática de Testes) com base na estrutura da *framework* WRAFS (*WinRunner Automation Framework Support*), permitindo que os testes que são realizados manualmente por um testador, através de acções sobre a interface gráfica, possam ser automaticamente gerados e executados, sem que seja necessário conhecimento na *framework* ou na linguagem de programação da ferramenta *capture-replay* que realizará a execução dos testes. O conceito de cenário, criado especificamente para o sistema AGEAT, agrega, para um conjunto de classes de objectos gráficos, funções de componentes que serão sugeridas durante a geração dos testes.

O sistema AGEAT ultrapassa as principais limitações da *framework* WRAFS e pode ser utilizado por qualquer sistema que possua uma interface gráfica suportada pela ferramenta Winrunner. Como valor adicional pode ainda ser utilizado como meio pedagógico para aqueles que queiram conhecer em pormenor a *framework* WRAFS.

Foram desenvolvidos dois casos de estudo com o objectivo de avaliar, sob cenários reais, a aplicabilidade do sistema AGEAT. O primeiro foi realizado sobre o sistema Fénix, com interface *web*, que suporta a gestão de processos de negócio duma universidade. O segundo foi realizado sobre o sistema L'Avocat, com interface *windows*, que suporta a gestão de processos de negócio de escritórios de advocacia. Os sistemas estão desenhados com características de usabilidade distintas o que permitiu verificar, com satisfação, a flexibilidade do sistema AGEAT para a criação, execução e manutenção de testes funcionais automáticos.

Palavras Chave: testes funcionais, testes automáticos, ferramentas *capture-replay*, testes *data-driven*, testes *keyword-driven, frameworks* de teste.

Abstract

This research work is about automatic tests, performed over systems with graphical user interface (GUI), using capture-replay tools. In this dissertation, we propose a method for the automatic generation and execution of functional tests which is based on the GUI design of the target system and allows for the reduction of the efforts on building and maintenance of automatic tests.

To accomplish that, we've developed the AGEAT system (stands for "Environment for the Automatic Generation and Execution of Tests) on top of the WRAFS (WinRunner Automation Framework Support) framework, allowing for the tests otherwise performed manually by a tester through actions over the system's GUI, to be generated and executed automatically, without needing previous knowledge of the framework itself nor of the capture-replay tool's programming language. The scenario concept, created specifically for the AGEAT system, aggregates, for a set of graphical object classes, component functions that will be suggested during test generation.

The AGEAT system overcomes the main limitations of the WRAFS framework and can be used by any system with a GUI supported by the Winrunner tool. It can also be used as a pedagogic tool for those who want to get more familiar to the WRAFS framework.

Two case studies were performed with the goal of evaluating, in a real world scenario, the applicability of the AGEAT system. The first case study was performed over the Fénix system, which is web-based and supports the management of business processes in a university. The second one was performed over the L'Avocat System, a windows application, which supports the management of business processes in a law office. The applications were designed with distinct usability features which allowed us to verify, satisfactorily, the flexibility of the AGEAT system for the creation, execution and maintenance of automatic functional tests.

Keywords: functional tests, automatic tests, capture-replay tools, data-driven tests, keyword-driven tests, test frameworks.

Agradecimentos

Agradeço a meus pais, Fernando e Lourdes, e a meu esposo Sérgio, por todo amor e apoio que me dão.

Agradeço a meu Professor Alberto Silva, pelas advertências, correcções e boa orientação ao longo deste trabalho.

Agradeço a meus amigos por todo o incentivo.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Numa aproximação tradicional da Engenharia de Software o processo de desenvolvimento de software mais comum era constituído de **longos** períodos dedicados ao desenho e a construção do sistema, seguidos de **curtos** e desorganizados períodos de teste, realizados imediatamente antes de sua disponibilização ao mercado ou utilização dentro da empresa. A actual tendência de períodos de desenvolvimento reduzidos e de desenvolvimento de aplicações cada vez mais complexas vem dar maior importância à realização de **Testes** para a obtenção de **Qualidade de Software**.

Várias definições do conceito de testes estão disponíveis por exemplo:

"*Testing is a process of planning, preparation, and measuring aimed at establishing the characteristics of a information system and demostrating the difference between the actual and the required status*" [PolVeenendaal1998].

Por outro lado, de acordo com a ISO (*International Standards Organization*) a definição de qualidade é:

"*The totality of features and characteristics of a product or a service that bear on its ability to satisfy stated or implied needs*" [ISO8402].

Analisando estas duas definições podemos dizer que os testes fornecem um **instrumento** para o aumento da **qualidade** dos sistemas, uma vez que ao identificar as diferenças entre o estado actual e o estado desejado, estas diferenças poderão ser eliminadas. Sem a realização de testes pouco podemos afirmar sobre a qualidade do sistema [KoomenPol1999]. No entanto, apenas as diferenças encontradas serão corrigidas.

Um conjunto de factores influencia o sucesso de um processo de testes: a escolha de profissionais experientes; regras bem definidas acerca do nível de qualidade pretendido; objectivos alinhados entre programadores e testadores que visem a qualidade do sistema; conhecimento das técnicas e abordagens existentes e aplicáveis a cada cenário de testes; e apoio pelos gestores. Por outro lado, os custos envolvidos num processo de testes são em geral altos, o que nalguns projectos o torna impraticável.

Considerando as seguintes tarefas do processo de desenvolvimento de software: Planeamento, Desenho, Programação, Testes, Operação e Manutenção [KanerFalkNguyen1999], os custos relativos a cada tarefa são sumarizados na Tabela 1.1. A Tabela 1.1 mostra que a realização de Testes constitui a segunda actividade de maior custo do processo. Segundo Kaner, o custo de realização de testes corresponde a 45% (15/33) do custo de desenvolvimento inicial de um produto de software [KanerFalkNguyen1999]. Salientamos que parte do custo de manutenção apresentado também inclue custos de Testes, visto que a cada alteração de manutenção, alguns testes serão refeitos.

Tarefa	**Custo Total %**	**Custo de Desenvolvimento %**
Análise de Requisitos	3	9
Especificação	3	9
Desenho	5	15
Codificação	7	21
Testes	15	45
Manutenção e Operação	**67**	

Tabela 1.1 - Custo de cada estágio em percentual do total e do custo de desenvolvimento (adaptado de [KanerFalkNguyen1999])

Referências apontam para percentagens de 25 a 50% relativos ao custo da tarefa de testes em relação ao custo total do projecto [Kit1995], [KoomenPol1999].

Neste âmbito, novas abordagens de Testes são procuradas para garantir a qualidade dos sistemas, mas a menores custos. Uma delas passa pela **Automatização de Testes**, que corresponde a "utilização de software ou ferramentas de hardware (robôs de teste) que permitam a preparação, execução, avaliação de resultados ou realização de tarefas de pós-processamento de casos de testes, de forma não assistida" [BerezaJarocinski2001]

Esta definição evidencia que automatização de testes pode ser feita em contextos variados e com objectivos distintos. Por exemplo, cada actividade de testes (identificação, desenho, construção, comparação) pode ser automatizada de forma diferente e através de diferentes ferramentas [FewsterGrahm1999].

Aos testes realizados sobre a interface gráfica de um sistema, a opção de automatização é realizada através de ferramentas chamadas *capture-replay* [BerezaJarocinski2001]. Tais testes realizados manualmente requerem tempo e podem ser bastante beneficiados através da automatização. Por outro lado, a funcionalidade de *capture-replay* que caracteriza tais ferramentas não é suficiente para o sucesso do projecto de testes e para a redução dos custos [FewsterGrahm1999], [Hendricson1998], [Zambelich1999]. Os testes criados mas baseados apenas na funcionalidade *capture-replay* são vulneráveis e apresentam menor custo-benefício a longo prazo. Isto porque os

scripts capturados são pobremente estruturados o que dificulta a sua reutilização. Por isto, outras alternativas para o uso das ferramentas *capture-replay* têm sido propostas. Técnicas de programação podem ser aplicadas durante a construção dos testes automatizados. (Notar que o desenvolvimento de testes automáticos é, de facto, desenvolvimento de software [Kaner1997]). A utilização de testes *data-driven* que garantem a separação entre as acções e os dados de entrada que são processados constitui recurso fundamental para o sucesso da automação de testes [Pettichord2001]. A criação de testes apenas através da utilização de *capture-replay* deve ser feita com cuidado, em cenários específicos, como por exemplo para testes de sistemas que sejam pouco alterados ao longo do tempo.

O esforço necessário à **construção** de testes automáticos que tragam benefícios duradouros ao projecto, é maior que o esforço necessário à construção manual do mesmo teste. Por outro lado a **execução** de tais testes, requerem pouco esforço, comparativamente ao esforço manual de sua execução. Ao depender do nível de automatização feita, o esforço da execução pode ser resumido ao lançamento de um *script* de teste que realizará uma sequência de acções.

Algumas *frameworks* entretanto foram construídas com o objectivo de diminuir o esforço de **construção** e viabilizar economicamente a aplicação de tal automatização [Nagle2000], [ArcherCSDDT], [BuwaldaJanssenPinkster2002]. Algumas experiências em testes automáticos foram publicadas e constituem referências para análise [Akervold2004], [Momeni2001]. A maioria das *frameworks* existentes definem formatos específicos para entrada de dados de testes, podendo incluir também acções de testes. Um componente de *software* desenvolvido à parte faz o processamento de tal informação, inserida normalmente em ficheiros de texto. O objectivo principal de tais *frameworks* é fornecer um ambiente para a criação dos testes, sem que seja necessário um conhecimento profundo da utilização da ferramenta de testes que realiza a execução automática. Além disto, estas estratégias facilitam a manutenção dos testes uma vez que a inclusão de novos testes passa muitas vezes pela alteração dos ficheiros de texto com novos valores de entrada. Em contrapartida, estes *frameworks* exigem o conhecimento de sua estrutura interna e da linguagem e formato utilizados para contrução de tais ficheiros.

1.1. Motivação

Actualmente existe uma forte necessidade pela criação de métodos que reduzam o esforço da **construção** e **manutenção** de testes automáticos realizados sobre a interface gráfica. Enquanto que a execução dos testes é totalmente automática, através das ferramentas *capture-replay,* a sua construção e manutenção requer ainda um esforço significativo de programação. As expectativas criadas pelas ferramentas *capture-replay* são frequentemente frustadas quando se constroem *scripts* de testes apenas através da funcionalidade de captura.

As *frameworks* comerciais ou desenvolvidas à medida que suportam a construção e a execução de testes automáticos surgem como uma opção ao cenário de *scripts* capturados. No entanto, para além

do investimento necessário à compra da ferramenta *capture-replay* e à *framework,* a necessidade de formação especializada (na *framework* e em programação) pode influenciar a uma não adesão deste tipo de *frameworks*, mantendo-se a utilização de *scripts* capturados com as limitações referidas atrás.

Por conseguinte, faz-se necessário a concepção de novas abordagens que permitam a criação de testes automáticos que possam ser não apenas facilmente executados, mas principalmente facilmente construídos e mantidos. Desta forma, testadores experientes no desenho de testes (identificação de condições de teste e desenho de casos de teste) poderão utilizar a automatização como ferramenta para o aumento da produtividade no processo de testes. O tempo que é ganho através da facilitação das tarefas de **construção** e **manutenção** de testes automáticos pode ser utilizado para a **concepção** de testes mais eficazes na detecção de defeitos. Além disto, tais testes poderão ser reutilizados a cada nova *release* do sistema, com pouco ou nenhum esforço adicional.

Uma forma de facilitar a construção e a manutenção de testes automáticos funcionais passa por tornar a realização da **concepção** do teste independente da tarefa de **construção** (que no caso dos testes automáticos corresponde à implementação de *scripts* para uma execução automática). Isto significa que a concepção e o desenho dos casos de testes tendem a ser feitos de formas semelhantes, quer para uma execução manual ou automática.

Uma abordagem mais inovadora para a crição de testes automáticos funcionais não deveria exigir ao testador, necessariamente, conhecimento de linguagens de *scripts* de ferramentas *capture-replay* ou de *frameworks* particulares para a criação de testes funcionais que serão executados, automaticamente, através da interface gráfica do sistema em particular.

É neste sentido que surge este trabalho. Pretendemos desenvolver uma abordagem para construção de testes funcionais automáticos que permita a disassociação entre as actividades de concepção e de construção de testes e que não requeira do testador conhecimentos em linguagens de *scripts* ou novas *frameworks*. Foi desenvolvido um sistema que permite a **geração automática** de testes, baseado na informação da interface gráfica do sistema que será exercitado. Através deste sistema pretende-se uma diminuição do esforço de construção e manutenção dos testes, com vista à obtenção dos benefícios, já mencionados, aos projectos de testes.

1.2. Enquadramento

O *ProjectIT* [ProjectIT2005] é um projecto de I&D que fornece uma infraestrutura para desenvolvimento de *software* com suporte à engenharia de requisitos, às actividades de análise e desenho, a aspectos de gestão de projectos e de geração de código. O projecto está dividido em componentes ou sub-sistemas, nomeadamente: *ProjectIT-Workbench*, *ProjectIT-Time*, *ProjectIT-Requirements*, *ProjectIT-MDD* e *ProjectIT-Tests*. O *ProjectIT-Tests* enquadra-se na área da engenharia de testes e envolve vários tópicos de investigação entre eles, **testes automáticos.**

Um dos objectivos do *ProjectIT-Tests* é de desenvolver e gerir arquitecturas de testes e conjuntos de testes num contexto de um determinado projecto. Neste âmbito, e como trabalho principal desta tese de mestrado, foi feita a concepção e o desenvolvimento do sistema **AGEAT** (Ambiente de Geração e Execução Automática de Testes) com o intuito de permitir a **construção** e **manutenção** de testes funcionais automáticos baseado na configuração gráfica do sistema alvo e na navegação desejada pelo testador. Os testes gerados pelo sistema serão automaticamente executados sobre a ferramanta comercial *Winrunner* [RunnerManual76].

O principal factor de inovação existente no AGEAT está relacionado no facto de que os utilizadores não precisam conhecer linguagens de programação ou *frameworks* particulares para que seus testes sejam executados automaticamente.

No âmbito desta dissertação foi ainda realizada a publicação e a apresentação do artigo científico, "**Técnicas para Construção de Testes Funcionais Automáticos**", na 5ª Conferência para a Qualidade nas Tecnologias da Informação e Comunicações (Quatic 2004) [CorreiaSilva2004].

1.3. Objectivos

Os principais objectivos no âmbito deste trabalho de investigação são os seguintes:

- Exploração de *frameworks* existentes para construção de testes automáticos *data-driven* e de ferramentas *capture-replay*;
- Identificação de uma abordagem para construção de testes automáticos que não requeira conhecimento de *frameworks* específicas ou em linguagens de programação de ferramentas *capture-replay;*
- Criação de sistema que permita a redução do esforço de construção e manutenção de testes automáticos. Tal sistema deverá utilizar uma ferramenta *capture-replay* e uma *framework opensource* existente;
- Validação da utilidade do sistema desenvolvido para construção de testes sobre sistemas informáticos reais com interfaces distintas, web e windows;

1.4. Organização da Dissertação

Esta tese é composta por sete capítulos e dois apêndices. De seguida descreve-se de forma sucinta o conteúdo de cada capítulo.

Capítulo 1 – Introdução. Neste capítulo é feito uma contextualização do âmbito e problema de investigação. Para tal, são introduzidos alguns conceitos básicos na área de testes e, em particular, de testes automáticos. As principais dificuldades no desenvolvimento de testes automáticos são

apresentadas e desta forma apresenta-se a motivação para o desenvolvimento da tese. Os objectivos a serem alcançados com o desenvolvimento do trabalho são por fim listados.

Capítulo 2 – Estado da Arte. Neste capítulo são introduzidos conceitos no âmbito da Engenharia de Software relacionados aos testes, manuais ou automáticos. São apresentadas as características e recursos das ferramentas *capture-replay* que permitem a execução de testes sobre interfaces gráficas. Por fim, as técnicas mais conhecidas para a criação de testes automáticos são descritas em detalhe.

Capítulo 3 – Frameworks WRAFS. Este capítulo resulta da investigação sobre a *framework WRAFS,* que suporta a criação de testes automáticos executados sobre a ferramenta *capture-replay Winrunner* [RunnerManual76]. O capítulo descreve em detalhe o comportamento e estrutura da *framework* com o objectivo de permitir uma análise crítica e entendimento da solução que será proposta pela tese.

Capítulo 4 - Problema. Este capítulo apresenta uma análise das principais limitações da *framework WRAFS*, sobre os quais uma alteração à abordagem traria evidentes benefícios à construção e manutenção dos testes. Dos cinco aspectos identificados como limitações, esta dissertação fornecerá uma solução para os três com impacto mais negativo à criação de testes automáticos. Os dois outros aspectos constituirão alvo de trabalho futuro.

Capítulo 5 – AGEAT: Análise e Concepção. Este capítulo e o próximo constituem em conjunto o corpo mais inovador desta tese. Este capítulo descreve o sistema AGEAT desenvolvido. O sistema é apresentado através de um Modelo de Casos de Uso, que apresenta os perfis de utilizadores e as principais funcionalidades do sistema, e um Modelo de Domínio onde são descritas as principais classes existentes. É também apresentada a relação entre o sistema AGEAT e os componentes de software externos relacionados, nomeadamente: *Winrunner* e a *framework* WRAFS.

Capítulo 6 – Casos de Estudo. Este capítulo apresenta uma validação do sistema AGEAT, através de dois casos de estudo. O capítulo apresenta o processo de criação de testes para dois sistemas reais, com interfaces *Windows* e *Web*, através do sistema AGEAT. As principais situações ocorridas durante o processo são apresentadas de modo a que seja possível perceber a aplicabilidade do sistema em situações reais. Alguns dos produtos decorrentes deste processo como a configuração dos objectos da interface gráfica, a composição dos testes e os resultados obtidos são apresentados nos Anexos A e B relativos aos sistemas alvo exercitados respectivamente, *Fénix* e *L'Avocat*.

Capítulo 7 – Conclusão. Este capítulo apresenta a conclusão do trabalho de investigação e ainda apresenta as principais contribuições e dificuldades obtidas e referências para trabalho futuro.

Apêndice A – Artefactos de Teste do Sistema *Fenix*. Este Apêndice apresenta alguns dos produtos gerados durante o processo de teste ao sistema ***Fenix***. A Secção A1 apresenta o ficheiro *(guimap)* de configuração dos objectos gráficos da aplicação alvo. A Secção A2 apresenta a

composição das baterias de teste criadas e a Secção A3 apresenta o resultado da execução do caso de estudo ao sistema web escolhido.

Apêndice B – Artefactos de Teste do Sistema *L'Avocat*. Este Apêndice apresenta alguns dos produtos gerados durante o processo de teste ao sistema *L'Avocat*. A Secção A1 apresenta o ficheiro *(guimap)* de configuração dos objectos gráficos da aplicação alvo. A Secção A2 apresenta a composição das baterias de teste criadas e a Secção A3 apresenta o resultado da execução do caso de estudo ao sistema *windows* escolhido.

Capítulo 2

Estado da Arte

A crescente complexidade nos sistemas informáticos juntamente com os métodos de desenvolvimento rápido e incremental – como por exemplo, *Rapid Application Development* [Mcconnell1996] e *Extreme Programming* [Beck2000], que prometem intervalos de entrega mais frequentes – requerem testes de qualidade que possam ser rapidamente executados sempre que necessário. Em tal cenário os testes manuais são pouco vantajosos, visto que muitos testes são re-executados a cada *release* do sistema.

Os testes automáticos fornecem uma solução neste sentido pois, quando desenvolvidos de forma adequada, serão facilmente executados. Muitos projectos revelam que o conhecimento na área de automação e experiência nos métodos e ferramentas disponíveis, bem como o uso de técnicas que promovam a reutilização e facilidade de manutenção dos testes automáticos, são fundamentais para se obter êxito nesta área [DoughertyHaber2002], [Linz1998], [Hendricson1998].

Este capítulo apresenta uma análise geral à temática dos testes automáticos organizado da seguinte forma: a Secção 2.1 introduz os conceitos básicos sobre testes manuais e testes automáticos usados no âmbito da engenharia de software; a Secção 2.2 apresenta as características das actuais ferramentas para construção e execução automática de testes funcionais; a Secção 2.3 apresenta diferentes técnicas para automatização de testes funcionais e finalmente, as considerações finais e de síntese do capítulo são apresentadas na Secção 2.4.

2.1. Testes em Engenharia de Software

Podemos definir os testes como uma actividade que tem como objectivo verificar se o software construído está de acordo com sua especificação e se satisfaz as expectativas do cliente e ou utilizador do sistema. Esta definição é uma conclusão a partir do reconhecimento de que a actividade dos testes é parte integrante do processo de Validação e Verificação (V & V) da Engenharia de Software, sendo considerada a técnica dinâmica que exercita a implementação [Sommerville2000].

Nesta secção apresentam-se os conceitos básicos sobre os testes manuais e testes automáticos usados no âmbito da engenharia de software. Introduz-se nomeadamente os tópicos referentes a (1) planeamento de testes; (2) granularidade de testes; (3) abordagens de testes; (4) testes manuais vs automáticos; e (5) automatização da actividade de teste.

2.1.1. Planeamento de Testes

Tal como referido anteriormente, os testes "exercitam uma implementação", mas é importante salientar que previamente a este exercício os testes devem ser planeados de tal modo que satisfaçam seu objectivo principal que, na opinião de Myers, é de "revelar a existência de defeitos" [Myers1979]. Um caso de teste é constituído por um conjunto de **dados de entrada, condições de execução** de uma ou mais operações e **resultados esperados ou dados de saída**, desenvolvidos com um objectivo particular. O desenho dos **casos de teste** e a preparação dos **dados de teste** constituem actividades fundamentais do planeamento dos testes realizadas por um testador.

Visto que o número de casos de teste normalmente é elevado, na maioria das vezes apenas é executado um subconjunto destes casos de testes. Faz parte também da actividade de planeamento determinar quais são os testes mais importantes e que deverão ser executados. Por exemplo, pode ser decidido que, os testes às funcionalidades já existentes em outra versão do sistema devem ser prioritários aos testes às funcionalidades novas oferecidas na nova versão do sistema. Neste caso, as funcionalidades já existentes que forem escolhidas para serem testadas serão exercitadas pelos chamados Testes de Regressão [Pfleeger2001].

Testes de Regressão são definidos como sendo testes aplicados a uma nova versão ou *release* de um sistema a fim de verificar que ele continua a realizar as mesmas funções e da mesma maneira que na versão anterior [Pfleeger2001]. Os Testes de Regressão são reutilizados entre diferentes versões do sistema, sofrendo pouca ou nenhuma alteração.

2.1.2. Granularidade dos Testes

Devido à complexidade dos sistemas a actividade de testes deve ser feita ao longo de diferentes estágios. O processo de testes mais utilizado é composto por cinco estágios, como mostramos na Figura 2.1.

O processo é iterativo, sendo que a informação, produzida em cada estágio, poderá fluir entre estágios adjacentes. Os estágios do processo de testes segundo definições em [Sommerville2000] são:

Testes de Unidade: Componentes individuais são testados independentemente de outros componentes para certificação de que operam correctamente.

Testes de Módulo: Um módulo é uma colecção de Componentes relacionados tais como classes, tipos abstractos de dados ou um conjunto de procedimentos ou funções. Durante este estágio cada módulo é testado individualmente.

Testes de unidade e de módulo fazem parte do processo de implementação e são da responsabilidade dos programadores que estiveram a desenvolver o componente alvo e ou o componente para testar o componente alvo.

Testes de Sub-Sistema ou Testes de Integração: Esta fase envolve o teste de colecções de módulos integrados em sub-sistemas. Sub-sistemas podem ser desenhados e implementados independentemente. Um problema comum em grandes sistemas está na integração das interfaces entre os módulos dos sub-sistemas. Estes testes devem portanto concentrar-se no exercício rigoroso de tais interfaces para detecção de possíveis erros.

Testes de Sistema: Os sub-sistemas são integrados para formarem um único sistema. O processo deste tipo de teste irá detectar falhas resultantes das interacções entre subsistemas e componentes de sistema.

Testes de Aceitação: Este é o estágio final do processo de testes antes do sistema ser aceite para uso operacional. O sistema é testado com dados fornecidos pelo utilizador final, ao contrário de dados fictícios ou simulados. Os testes de aceitação revelam principalmente erros e omissões na definição dos requisitos, pois através do uso de dados reais o sistema é exercitado de formas variadas.

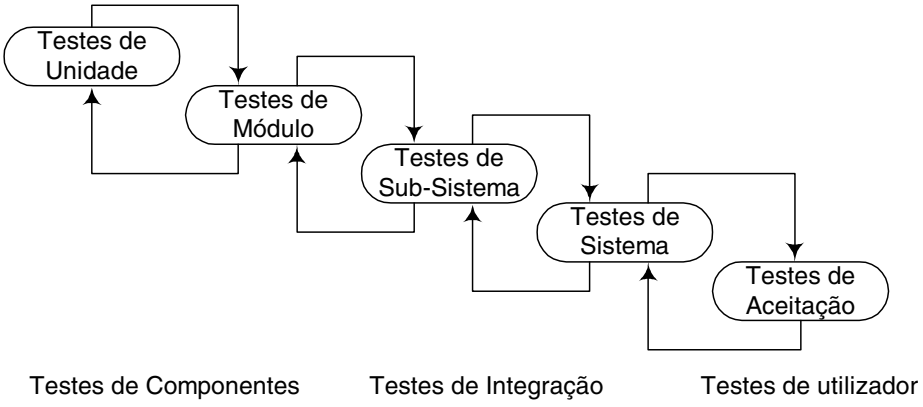

Figura 2.1 - Visão geral do processo de testes (adaptada de [Sommerville2000])

2.1.3. Abordagens de Testes

Testes *Black Box* ou Testes Funcionais e Testes *White Box* ou Testes Estruturais representam as principais abordagens existentes de teste segundo [Kit1995]. Qualquer destas abordagens podem ser aplicadas, em princípio, durante qualquer estágio do processo de testes, contudo cada uma delas é

preferencialmente aplicável a determinados tipos de componentes e realizáveis por equipas distintas.

Segundo [Sommerville2000], a abordagem funcional por exemplo, é melhor aplicada sobre componentes de sistema e realizada por uma equipa de testes, enquanto a abordagem estrutural é melhor aplicada a componentes individuais ou a colecções de componentes dependentes e realizada pela equipa de desenvolvimento.

Através da abordagem de Testes Funcionais, os casos de testes são derivados a partir da especificação do sistema ou componente a ser testado. O sistema é visto como uma caixa fechada e o seu comportamento apenas pode ser derivado através do estudo dos possíveis valores de entrada e dos valores de saída relacionados. Por outro lado, através da abordagem de Testes Estruturais, o testador pode analisar o código e usar o conhecimento da estrutura do componente para derivar os casos de testes. No restante deste capítulo, quando não explicitada a abordagem de testes, deverá ser assumida como a de testes funcionais.

2.1.4. Testar versus Automatizar Testes

A qualidade de um caso de teste é descrita através de quatro atributos [FewsterGrahm1999]. O primeiro consiste na capacidade de encontrar defeitos. O segundo refere a capacidade em exercitar mais de um aspecto reduzindo assim a quantidade de casos de teste requeridos. O terceiro e quarto fazem considerações de custo. O terceiro é inferido baseado no custo necessário para a realização do caso de teste incluindo o esforço de desenho, execução e análise dos resultados de teste. O quarto atributo refere o esforço de manutenção necessário sobre o caso de teste a cada alteração do sistema. Estes quatro atributos devem ser balanceados de forma a se ter casos de teste de boa qualidade.

A forma manual ou automática de realização de um teste, não interfere nos dois primeiros atributos citados. A automatização de um caso de teste interfere apenas em quão económico o caso de teste será e que esforço de manutenção será necessário. O esforço de construção e de manutenção requerido para um teste automático é normalmente maior do que para um teste manual equivalente. Mas uma vez construído, o teste automático tende a ser mais económico que o teste manual, o esforço de execução e de verificação de resultados será uma pequena fracção do esforço de construção. Para testes funcionais sobre sistemas com interface gráfica foi concluído por Linz e Daigl [LinzDaigl1998] que, após investimentos iniciais de criação de infra-estrutura, o gasto em testes automáticos representará 40% do gasto com testes manuais.

Visto que o custo original da implementação e o custo da manutenção de um caso de teste automático será diluído a cada execução que seja necessária, os Testes de Regressão são fortes candidatos a serem automatizados.

No entanto, os testes automáticos não são garantia da existência de testes eficazes. A identificação, selecção e desenho dos casos de testes funcionais devem ser feitos por um testador que domine o domínio da aplicação. O responsável que constrói testes automáticos e mantém os artefactos relacionados com o uso de uma ferramenta de execução de testes é chamado de *Test Automator* ou Automatizador de Testes. O automatizador de testes pode ser ou não um testador, devendo sempre ter habilidades em programação. A habilidade do automatizador ditará a qualidade da automação, mas será a habilidade do testador quem ditará a eficácia dos testes realizados.

2.1.5. Automatização da Actividade de Teste

Nesta secção descrevemos as actividades de teste e suas possíveis formas de automatização. As actividades de teste são normalmente realizadas na seguinte sequência: (1) Identificação, (2) Desenho, (3) Construção, (4) Execução e (5) Comparação, como mostra a Figura 2.2. As três primeiras actividades fazem parte do planeamento dos testes. Falaremos de cada uma individualmente.

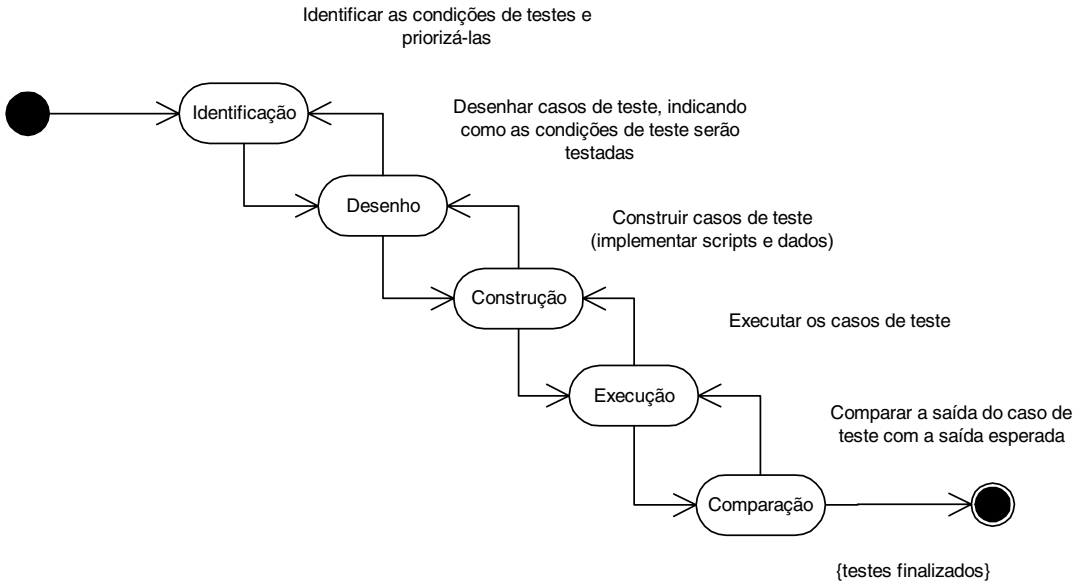

Figura 2.2 - Actividades de teste (adaptada de [FewsterGrahm1999])

Identificação

Nesta primeira etapa, o testador determinará o que será testado identificando as condições de teste (itens ou eventos) que precisam ser verificados por cada teste. Condições de teste são descrições de circunstâncias que devem ser examinadas. Existem diversas técnicas para uma derivação rigorosa de condições de teste nomeadamente [Kit1995]: *equivalence particioning, boundary value analysis, cause-effect graphing*, todas indicadas para uso em uma abordagem de Testes *Black Box* ou

Funcional. Algumas técnicas para derivação de condições de teste para uso sob uma abordagem *White Box* é apresentada em [Kit1995], como *statement coverage* e *path coverage*.

Desenho

O desenho dos casos de teste determinará como as condições de testes serão testadas. Um caso de teste é um conjunto de testes realizados em sequência e que possuem um objectivo comum que é a razão ou propósito do teste.

Cada caso de teste possui dados de entrada, a informação necessária para a execução do teste e a saída esperada. Os pré-requisitos para realização dos testes, tais como variáveis do ambiente de execução ou estado da base de dados devem ser explicitados para cada caso de teste. A saída ou resultado esperado inclui a criação ou visualização de itens, a modificação ou actualização de itens (em base de dados, por exemplo) ou a remoção de itens. O resultado esperado deve indicar portanto o estado do sistema após a realização da operação de teste. A actividade de desenho de casos de testes exige um esforço intelectual por parte do testador.

Algumas ferramentas fazem a geração automática dos casos de teste baseadas em código fonte [IPL], [LDRA], [Testwell] ou especificação formal [ADL].

Construção

Nesta actividade os casos de teste são transformados em scripts de teste que quando utilizados durante uma execução manual do teste, é também chamado de procedimento de teste. Um procedimento de teste detalha a informação descrita no caso de teste de modo que o testador possa, seguindo as instruções do procedimento, executar e validar a execução de cada teste. Um *script* de teste é normalmente armazenado em ficheiro e escrito em linguagem específica, usada por uma ferramenta de automação da execução do teste. Um *script* de teste pode implementar um ou mais casos de teste. A ferramenta irá processar o script executando as acções por ele descritas.

A preparação dos dados de entrada do teste e do resultado de saída esperado é tarefa fundamental da fase de construção do teste. Quando os resultados de saída forem utilizados para comparação automática através de uma ferramenta, estes devem ser minuciosamente descritos. Uma comparação manual não exige tanto rigor na descrição do resultado esperado que normalmente está incluso dentro do procedimento de teste.

Execução

Nesta actividade o sistema é executado utilizando os *scripts* de teste. Para testes manuais, esta fase pode consistir de testadores a seguirem as instruções existentes em um procedimento de teste. Para testes automáticos as tarefas desta fase podem ser resumidas apenas no lançamento do *script* de

teste correspondente. Os dados de entrada podem estar no *script* de teste ou separado em ficheiros ou em base de dados. A ferramenta irá executar o *script*, efectuando as acções que o testador efectuaria manualmente. Existem diferentes ferramentas para execução automática de testes e diferentes técnicas para criação de tais *scripts* como veremos apresentadas na Secção 2.3.

Comparação

Os resultados obtidos do sistema em testes são usados para determinação do resultado do teste. Existem dois possíveis resultados: positivo ou negativo. A verificação do resultado obtido pode ser feita com confirmação informal do que o testador espera ver ou pode ser um comparação rigorosa e detalhada entre o resultado obtido e a resultado esperado (determinado durante a fase de construção do teste). Alguns resultados podem ser comparados ainda durante a execução do teste (uma mensagem que deve ser enviada ao ecrã), mas, por exemplo, resultados que fazem modificações de registos em base de dados podem apenas ser comparados depois que a execução do teste é finalizada. Um teste automático pode utilizar os dois métodos de comparação.

2.2. Ferramentas para construção e execução de testes automáticos funcionais

As ferramentas para construção de testes automáticos funcionais permitem a realização dos testes de uma forma não assistida.

Nesta secção iremos destacar as ferramentas do tipo *capture-replay* fazendo uma descrição de suas funcionalidades, recursos, limitações e contextos de utilização. A ferramenta comercial WinRunner [RunnerInfo] será usada para a exemplificação de alguns aspectos descritos.

2.2.1. Funcionalidades

As ferramentas *capture-replay* ou gravação-execução que daqui por diante abreviaremos por ferramentas GE, são ferramentas que permitem a criação, execução e a verificação dos resultados de testes automáticos para sistemas com interface gráfica standard. Na indústria informática, são as ferramentas mais conhecidas para a realização de testes automáticos.

As ferramentas GE possuem duas funções principais de utilização, a função de gravação e a função de repetição. Através da primeira, todos os objectos visualizados no sistema a ser testado são registados e toda a sequência de interacções realizadas sobre estes objectos são registados/gravados em ficheiro. Este ficheiro ou *script* de teste, como passa a ser chamado, contém todos os movimentos do rato e teclado realizados, todas as entradas inseridas, as opções escolhidas e o resultado obtido. Quando interrompido o modo de gravação, a ferramenta pára de registar as acções no *script*.

Através da função de repetição, qualquer *script* escrito na linguagem da ferramenta poderá ser executado, incluindo aquele criado pela função de gravação. A execução de um *script* que tenha sido gravado será uma repetição fiel de todas as acções realizadas aquando da gravação.

O resultado obtido durante a gravação pode ser considerado o resultado esperado do teste e será comparado com os resultados obtidos durante as subsequentes execuções automáticas como forma de determinar o resultado final do teste.

2.2.2. Recursos Utilizados

Existem duas principais formas usadas pelas ferramentas GE para o reconhecimento dos elementos de uma interface gráfica da aplicação sob testes. A primeira é orientada pela posição das coordenadas do elemento gráfico no ecrã. A segunda, mais flexível, reconhece os elementos da interface como objectos gráficos, possuidores de propriedades que determinam o seu aspecto e comportamento.

Para a ferramenta WinRunner [RunnerInfo], que realiza testes sobre aplicações com interface Windows ou Web, a lista das propriedades físicas juntamente com os respectivos valores de atribuição, usados para identificação de cada objecto gráfico, formam o que é chamado de "**descrição física**" do objecto [RunnerManual76]. No entanto, os *scripts* de teste referenciam os objectos utilizando outro identificador, chamado de "**nome lógico"**. A relação entre o descritor físico e o nome lógico é mantida em ficheiros chamados de GuiMap[RunnerManual76]. Esta solução, permite que os *scripts* mantenham-se sintaticamente válidos mesmo quando há alteração nas propriedades físicas dos objectos, passando a ser necessário apenas alteração no ficheiro GuiMap.

Existem duas formas para a geração dos ficheiros GuiMap, oferecidas pela ferramenta Winrunner. Durante a função de gravação a ferramenta faz o reconhecimento de todos os objectos sob os quais houveram interacções e regista-os num ficheiro GuiMap. Uma outra função disponível pelo Winrunner, chamada de *Learn*, faz o reconhecimento de cada objecto gráfico que é seleccionado pelo utilizador e o regista no ficheiro GuiMap.

A maioria das ferramentas GE permitem a criação de testes que interajam com objectos gráficos e janelas criados com o standard *Microsoft Foundation Class Library* (MFC). Objectos e janelas criados usando tecnologias diferentes por exemplo, *Java Foundation Class Library* podem ou não ser suportados pela ferramenta envolvida [Hendrickson1999]. No caso da ferramenta WinRunner existem *add-ins* para o tratamento de aplicações Java, Power Builder, Visual Basic, Oracle, Delphi, Siebel e aplicações Web [HorwathGreenLawler2000], entre outras. Caso uma ferramenta GE não consiga identificar o tipo do objecto gráfico, como último recurso a ferramenta GE pode sempre reconhecer objecto pelas suas coordenadas espaciais do ecrã.

Muitas ferramentas GE, incluindo WinRunner, possuem funções próprias para a realização de *queries* SQL sobre qualquer base de dados que suporte a interface ODBC. Tal funcionalidade permite que a verificação do resultado não seja feita apenas baseado no que é visualizado no ecrã, mas sim no que foi realmente alterado em base de dados.

A possibilidade de permitir a criação de bibliotecas de funções reutilizáveis é uma funcionalidade bastante desejável e existente em muitas ferramentas. Além de permitir a criação de bibliotecas próprias algumas ferramentas permitem o acesso a bibliotecas externas através de chamadas a ficheiros .dll.

As linguagens de scripts fornecidas pelas ferramentas GE normalmente são proprietárias e interpretadas. A ferramenta WinRunner fornece uma linguagem estruturada, semelhante a C, chamada TSL (*Test Script Language*) [RunnerManual76].

2.2.3. Limitações

As actuais ferramentas comerciais GE são vendidas como sendo uma solução fácil e auto-suficiente para a realização de testes automáticos, no entanto a experiência mostra que os *scripts* gravados possuem algumas limitações que inviabilizam a sua utilização como única forma de criação de testes. Abaixo mostramos algumas das fragilidades dos *scripts* gravados, designadamente:

- Os *scripts* são pobremente estruturados. Os *scripts* gerados pelo modo de gravação resultam em *scripts* extensos onde todas as acções efectuadas são registadas. Muitas destas acções poderiam ser reutilizadas por outros testes mas são repetidamente gravadas em cada novo *script* de teste gravado.

- Inexistência de mecanismos de reutilização de código. As acções (código), os dados de entrada e de resultado esperado ficam todos juntos no *script* criado pelo modo de gravação. Isto não permite que os *scripts* sejam escaláveis e reutilizados pois os dados estão hard-coded no *script*. Muitas vezes o que difere um caso de teste de outro são os dados por ele processados, enquanto as acções são as mesmas.

Estas limitações podem ser ultrapassadas se, tal como acontece na programação, utilizarmos as técnicas apropriadas para construção dos *scripts*. Na Secção 2.3 veremos algumas formas de se desenvolver *scripts* com baixo custo de manutenção e mais vantajosos num desenvolvimento a longo prazo.

2.3. Técnicas para construção de scripts de testes automáticos

A actividade de automação de testes inclui tarefas de programação idênticas as utilizadas no desenvolvimento de software [DustinRashka2001]. As técnicas para construção de *scripts* de testes

automáticos são similares às técnicas de programação. Os *scripts* de testes automáticos contêm dados e instruções para a ferramenta de teste. A minimização do esforço de manutenção dos *scripts* só é conseguida através de um investimento na construção dos *scripts*. Em [FewsterGrahm1999], os autores consideram a existência de cinco técnicas para criação de *scripts*, que se refere de seguida.

Scripts Lineares

Um *script* linear é aquele obtido a partir da gravação feita por uma ferramenta GE. É uma rápida forma de começar a construir *scripts* de testes automáticos, pois não requer conhecimento da linguagem oferecida pela ferramenta. No entanto, estes *scripts* não são úteis num plano a longo prazo. Normalmente, possuem informação excessiva e repetida, dados registados junto às acções (hard-coded) e estão muito associados a particularidades do sistema na altura da gravação o que os torna bastante vulneráveis a mudanças no sistema a ser testado. A criação de testes automáticos através de *scripts* gravados não é portanto uma boa prática.

Scripts Estruturados

Tal como as linguagens de programação estruturadas estes *scripts* usam estruturas de controlo como "If" e "Loop", o que garante uma flexibilidade não existente nos *scripts* lineares. Na escolha da ferramenta convém verificar, entre outras, a capacidade das suas linguagens no que se refere às instruções de controlo.

Scripts Partilhados

São *scripts* que podem ser usados por mais de um caso de teste. Os *scripts* partilhados podem ser específicos a uma aplicação ou independente de aplicação. Uma vez identificado um conjunto de acções úteis para mais de um teste, um *script* partilhado deve ser criado e disponibilizado para invocação a partir de qualquer outro teste. As informações variáveis devem ser passadas como parâmetro, tal como acontece na invocação de uma função na programação estruturada.

Scripts *Data-driven*

São *scripts* mais abrangentes que lêem entradas de testes ou resultados esperados a partir de um ficheiro de dados ou tabela de dados evitando termos dados *hard-coded* no próprio *script*.

Além disto esta técnica permite que novos testes sejam adicionados mais facilmente, uma vez que em alguns casos a existência de novos testes pode ser expressa pela inclusão de novas entradas na tabela de dados, sem nenhuma alteração no *script* de controlo. Os testes podem ser adicionados sem a necessidade de alteração no código do *script*. Em adição à entrada de teste, o resultado esperado também pode ser removido do *script* e colocado no ficheiro de dados, uma vez que o resultado esperado está directamente associado com a entrada do teste.

Muitas ferramentas GE encorajam a utilização desta técnica fornecendo mecanismos para armazenamento de dados de entrada em ficheiro texto e atribuição dos mesmos, durante a execução, a variáveis do *script*.

Como uma limitação desta técnica podemos citar que os *scripts data-driven* requerem que o desenho do teste seja feito na linguagem de automação da ferramenta. Desta forma, todos os envolvidos com o desenvolvimento de testes automáticos ou execução automática de testes precisam ser conhecedores do ambiente e da linguagem de programação da ferramenta de automação. A Figura 2.3 apresenta um exemplo de utilização desta técnica para testes a um sistema de facturação. Os testes apresentados fazem a inserção de clientes, onde os dados requeridos são: nome, telemóvel e número de contribuinte.

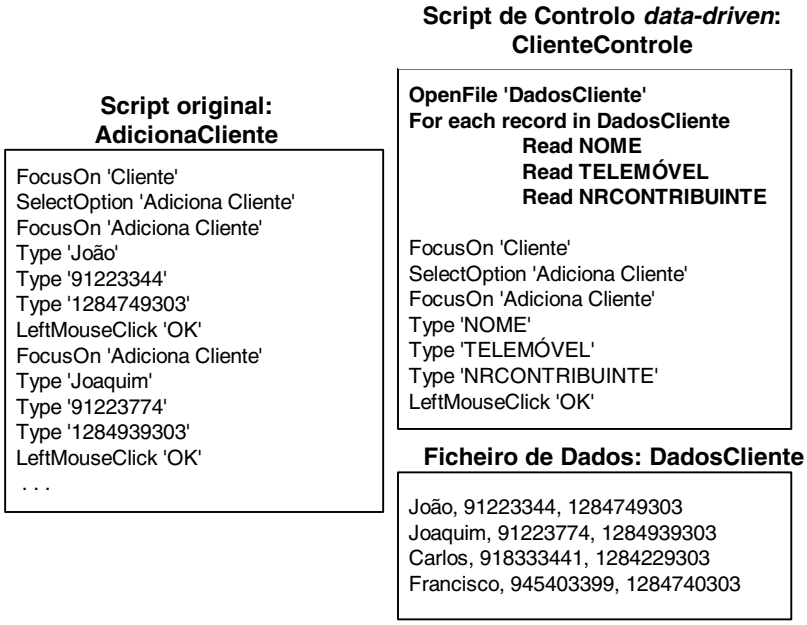

Figura 2.3 - O script original AdicionaCliente, implementado pela técnica data-driven (adaptada de [FewsterGrahm1999])

Scripts *Keyword-driven*

Na técnica *data-driven* a navegação e acções realizadas são as mesmas para cada caso de teste e o conhecimento lógico sobre os testes está distribuído no ficheiro de dados e no *script* de controlo e precisam ser sincronizados. A técnica *keyword-driven* combina a técnica *data-driven* com a possibilidade de especificar os casos de teste de forma menos detalhada, tal como é feito quando estamos a descrever um caso de teste manual.

Esta técnica expande o ficheiro de dados da técnica anterior de forma a torná-lo uma descrição dos casos de teste que queremos automatizar, utilizando um conjunto de palavras chave (*keywords*). Este novo ficheiro que é chamado de ficheiro de testes descreve apenas o que o caso de testes faz, mas não a forma como é feito. O *script* de controlo tem habilidade de interpretar estas *keywords*, as

quais estão implementadas fora do *script* de controlo. Esta separação na implementação das *keywords* requer um nível adicional de implementação técnica, que é feito através dos chamados *scripts* de suporte. Temos portanto três estruturas básicas que são o *script* de controlo, os ficheiros de teste e os *scripts* de suporte.

Esta técnica permite usar o conhecimento de um testador experiente no domínio da aplicação para o desenho ou construção dos casos de testes em ficheiros de teste e o conhecimento técnico de um automatizador nas ferramentas e linguagens existentes para a construção dos scripts de suporte, utilizando assim perfis distintos de testadores para a construção de testes automáticos mais efectivos e robustos. A Figura 2.4 apresenta a utilização desta técnica para o exemplo de um sistema de facturação, onde os testes pretendem verificar a realização de operações básicas, como adicionar/remover factura e adicionar/remover cliente. As palavras chave AdicionaFactura e RemoveFactura, são invocadas em diferentes testes ou ficheiros de teste, mas estão implementadas apenas nos scripts de suporte correspondentes.

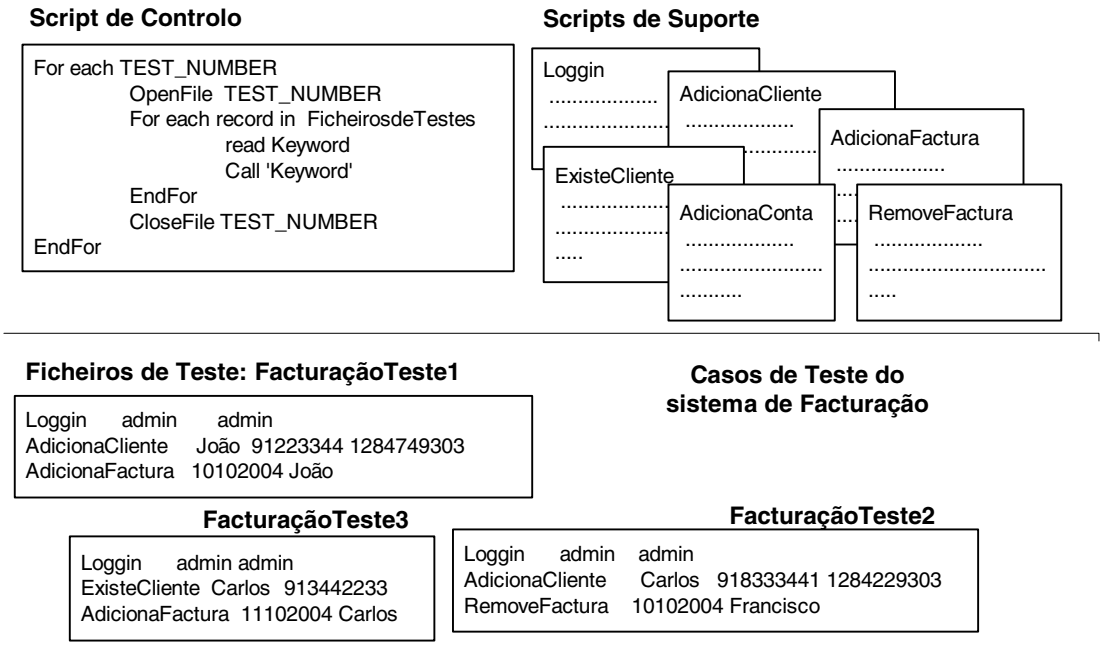

Figura 2.4 - Aplicação da técnica keyword-driven (adaptada de [FewsterGrahm1999])

Alguns produtos de suporte a testes automáticos foram desenvolvidos com base nesta técnica. O produto comercial *TestFrame*, fornece um *script* de navegação, cuja função principal é processar conjuntos de ficheiros em formato Excel, contendo referências às chamadas "action words"[BuwaldaJanssenPinkster2002]. No capítulo 3, uma *framework* de domínio público baseada nesta técnica, será apresentada em detalhe.

2.4. Resumo

Concluímos este capítulo com a apresentação da Figura 2.5 que sintetiza os conceitos e classificações apresentados sobre os testes de sistemas informáticos.

Vista como actividade que promove o aumento na qualidade do software, a realização de testes de acordo com uma metodologia tem vindo a se tornar cada vez mais frequente na indústria informática. Vimos neste capítulo que esta actividade deve ser realizada segundo um processo bem definido que valorize a qualidade do desenho dos testes independente do tipo de suporte possível: manual ou automático.

Neste capítulo, apresentamos os diferentes tipos de testes segundo sua granularidade e as principais abordagens de teste existentes. Indicamos como cada uma das actividades de testes pode ser realizada automaticamente, salientando o contexto onde a automatização é mais indicada.

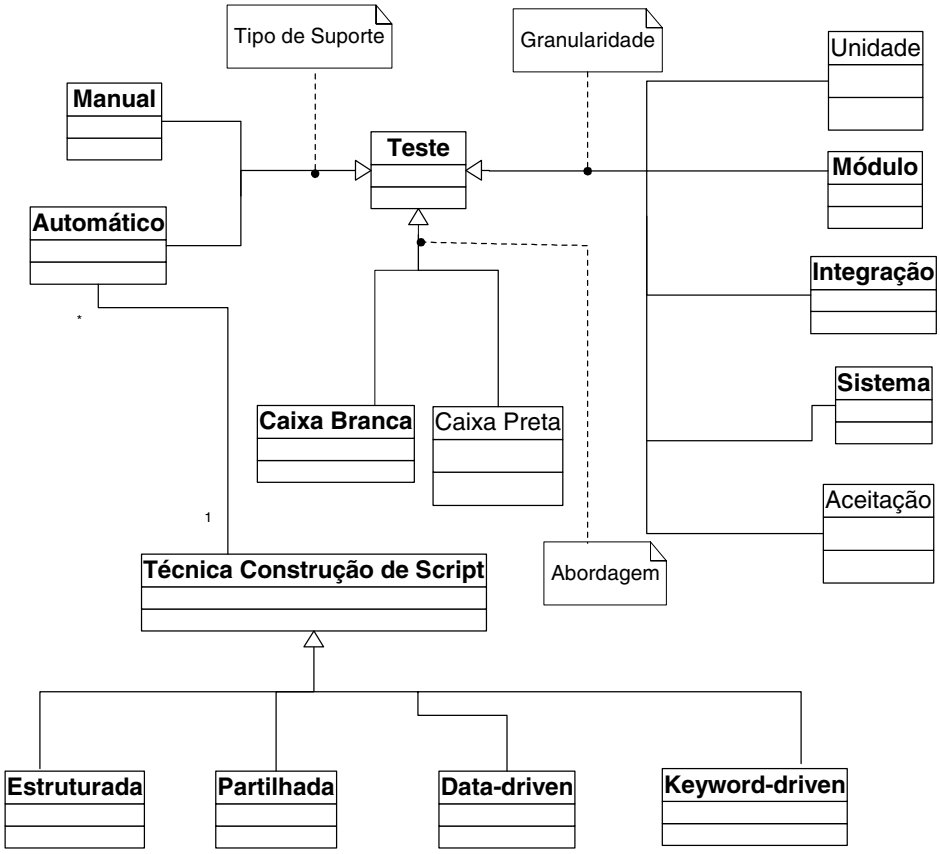

Figura 2.5 - Visão geral dos tipos de testes

Apresentamos as funcionalidades das ferramentas que permitem a execução e comparação automática dos testes, bem como suas limitações. Diferentes técnicas para a construção de testes automáticos foram apresentadas, tendo sido salientadas as mais evoluídas *data-driven* e *keyword-driven*, que permitem a criação de testes automáticos reutilizáveis e flexíveis.

Um procedimento favorável ao sucesso da automação é tratá-lo como um projecto de desenvolvimento de software [Pettichord2001]. Neste sentido, sugere-se a criação de uma infra-

estrutura de testes automáticos reutilizáveis e de fácil manutenção, com um objectivo de sucesso a longo prazo. Apenas desta forma poderão ser constatadas as vantagens da automação.

Capítulo 3

Framework WRAFS

Este capítulo apresenta o projecto SAFS (*Software Automation Framework Support*) que possibilita a implementação de *frameworks* para automação de testes do tipo *keyword-driven*, a serem executados por uma ferramenta *Capture-Replay*. Após uma visão geral do projecto, a implementação WRAFS, específica para a ferramenta WinRunner[RunnerInfo], é apresentada em detalhe.

O projecto SAFS surgiu a partir do trabalho desenvolvido por Carl Nagle para o instituto SAS, para a criação de uma *framework* para construção de testes automáticos de interfaces gráficas [MosleyPosey2002]. O instituto SAS detém os direitos intelectuais, mas todos os desenvolvimentos feitos com base neste projecto estão disponíveis para uso público. Os primeiros desenvolvimentos foram iniciados em 1999 tendo continuado até a data actual.

O capítulo está organizado da seguinte forma: a Secção 3.1 descreve o projecto SAFS e suas potencialidades; a Secção 3.2 descreve a especialização do projecto na implementação de uma *framework* para uso através da ferramenta WinRunner; a Secção 3.3 apresenta o fluxo de execução dos testes conforme suportado pela *framework*, a Secção 3.4. apresenta como a verificação dos resultados de teste é feita no âmbito do uso da *framework*.

3.1. O projecto SAFS

Tal como vimos no capítulo 2, a técnica para a construção de testes *keyword-driven* possui inúmeras vantagens. A possibilidade de se fornecer uma interface que não exija conhecimento nas ferramentas *Capture-Replay*, nomeadamente a interface de ficheiros texto, para construção de testes em uma linguagem de alto nível, traz bastantes vantagens [Zambelich1999].

Baseado numa abordagem *keyword-driven*, Carl Nagle, do instituto SAS, desenvolveu inicialmente a *framework* RRAFS (*Rational Robot Automation Framework Support*) para automação de testes funcionais sobre sistemas com interface gráfica, específica para uso sobre a ferramenta *Rational Robot* [RobotInfo]. Posteriormente, utilizando o mesmo modelo usado para o projecto RRAFS,

desenvolveu uma nova *framework* sobre a ferramenta WinRunner [RunnerInfo] chamada de WRAFS (*WinRunner Automation Framework Support*). RRAFS e WRAFS constituem as *frameworks* mais desenvolvidas dentro do projecto SAFS, no entanto novos motores SAFS podem ser desenvolvidos para diferentes ferramentas *Capture-Replay*, levando a existência de uma família de *frameworks,* para este tipo de ferramentas.

O projecto SAFS encontra-se disponibilizado no *SourceForge (http://sourceforge.net/projects/safsdev/)* direccionado para a criação de *frameworks* para automação de testes funcionais *Keyword-Driven,* utilizando ferramentas *Capture-Replay*. Este projecto está em constante evolução e constitui uma grande ajuda àqueles que pretendam automatizar testes sobre aplicações gráficas de forma planeada e estruturada.

Actualmente, o projecto SAFS tem como objectivo principal a criação de uma única *framework keyword-driven* genérica que permita a criação de testes independentes da ferramenta de execução *Capture-Replay* a ser utilizada. Através desta *framework*, os testes criados poderão ser executados sobre diferentes ferramentas *Capture-Replay*, sem que seja necessário alteração nos mesmos. A construção de uma *framework* independente de ferramenta *Capture-Replay* trará grandes benefícios à comunidade dedicada a automação de testes.

Apesar do desenvolvimento actual na construção desta *framework,* independente de ferramenta *Capture-Replay*, as *frameworks* RRAFS e WRAFS continuam a ser frequentemente actualizadas. Por um lado, para correcção de defeitos e inclusão de melhorias e, por outro lado, para que possam ser futuramente integradas à *framework* genérica. O trabalho desta tese focalizou-se sobre a *framework* WRAFS, que constitui uma das *frameworks* mais evoluídas [WRAFS2005] do projecto SAFS.

3.2. Framework WRAFS

No capítulo 2 vimos que, segundo a técnica *keyword-driven,* a cada *keyword* pode estar associado um *script* de suporte que contém o código para a execução da acção referida pela *keyword.* Este *script,* no entanto, é construído através de programação na linguagem oferecida pela ferramenta *Capture-Replay*.

A *framework* WRAFS permite que os testes *keyword-driven* sejam criados ainda mais facilmente, sem que seja necessário a programação de *scripts* de teste na linguagem da ferramenta WinRunner. A partir dos comandos disponibilizados através da linguagem da ferramenta WinRunner, a *framework* criou novos comandos ou *keywords* para serem usadas na construção dos testes. Os novos comandos fornecidos pela *framework* incluem funcionalidades para tratamento de erros e escondem detalhes da sintaxe do comando original fornecido pela linguagem do WinRunner.

A Figura 3.1 apresenta trecho do código da *framework* que corresponde a implementação da *keyword* "*Select*" disponível para objectos do tipo *ComboBox*. A implementação verifica a geração de qualquer erro na execução da função WinRunner disponível *list_select_item.* O testador apenas precisa utilizar a *keyword "Select*" para utilização desta funcionalidade.

Os comandos são referenciados em ficheiros de texto e o agrupamento de comandos no formato determinado pela *framework* leva a criação dos testes automáticos. O testador precisa apenas dominar os comandos disponibilizados pela *framework*, e não tem, em geral, de conhecer a linguagem de programação do WinRunner. A criação e alteração de testes podem ser feitos com maior facilidade, visto que a sua alteração passa a ser realizada ao nível dos ficheiros texto e não ao nível de *scripts* longos e complexos na linguagem da ferramenta. Deste modo, a *framework*, esconde os detalhes e complexidades da ferramenta WinRunner e fornece uma interface mais simples e consistente para criação de testes.

```
status = list_select_item(StepDriverState["compGUIID"], selection, LEFT,0);

if (status != E_OK)
      {
      status = edit_set(StepDriverState["compGUIID"], selection);
      if (status != E_OK)
            LogMessage ("Unable to set " & StepDriverState["compname"] & " to value " & selection & ".");
      }

if (status == E_OK)
      {
      StepDriverState["statuscode"] = SDNoScriptFailure;
      LogMessage (StepDriverState["compname"] & " set to value " & selection & " successfully.");
      }
```

Figura 3.1 - Trecho do código da Framework WRAFS para a keyword Select disponível para objectos do tipo combobox

A Figura 3.2 apresenta a arquitetura da framework WRAFS. Os módulos centrais da *framework* são: *WRAFS Driver e Engine*. Os testes a serem realizados encontram-se estruturados segundo um formato definido (dentro das chamadas "*Test Tables": CycleTable, SuiteTable e StepTable).* A implementação de todos os componentes da *framework* WRAFS foi feita utilizando a linguagem disponibilizada pelo WinRunner a linguagem TSL (Test Script Language) [RunnerManual76].

Nas subsecções seguintes falaremos em maior pormenor sobre cada componente da *framework WRAFS.*

3.2.1. Funções de Componentes (Component Functions)

Funções de Componente ou *Component Functions* são funções da *framework* que realizam acções sobre os componentes gráficos da aplicação de teste.

A *framework WRAFS* criou um módulo de funções para cada tipo de objecto gráfico existente e reconhecível pela ferramenta WinRunner, como por exemplo, Botões, Caixa de Texto, Listas. Estas

funções foram criadas com base, principalmente, nas funções disponíveis na linguagem do WinRunner, sendo acrescentado algum código para tratamento de erros e de excepções. Por exemplo, para a função existente no WinRunner chamada de "*Click_Button"* foi adicionada a funcionalidade que verifica se o objecto recebedor da acção está actualmente activo. Se não estiver, tenta activar o objecto invocando a função "*SetFocus"* fornecida pelo WinRunner. Esta funcionalidade acrescida, evita que tenhamos execuções de testes complexos que falham, pelo simples facto de uma janela não estar activa no momento em que iria receber a acção, quando este facto não deveria influenciar o resultado final do teste.

Figura 3.2 - Arquitectura Framework WRAFS (adaptada de [MosleyPosey2002])

As funções de componente e seus argumentos definem o vocabulário de baixo nível da *framework* WRAFS. Cada módulo define as *keywords* que são válidas para o tipo de componente que ele trata. Por exemplo, para um componente do tipo "*TextBox"* existirão as *keywords: InputText,*

VerifyValue. Tais *keywords* são utilizadas na criação das tabelas de teste de baixo nível como veremos mais adiante.

A Tabela 3.1 ilustra parte do conteúdo de um ficheiro Excel, disponibilizado pela *framework* WRAFS, mostrando as funções existentes para o componente do tipo *ComboEditBox* e os respectivos argumentos. Através deste ficheiro o testador pode consultar as funções que pode utilizar para manipular os componentes gráficos na aplicação alvo e assim criar os seus testes.

Nome Função	Descrição	Objecto alvo	Var 1	Var N
Click	Attempts to perform a standard Click on the ComboEditBox	ComboEditBox		
Select	Select an item by its text value from a ComboEditBox.	ComboEditBox	^TextValue	
SelectIndex	Select an item by its index from a ComboEditBox (ComboBox).	ComboEditBox	^Indexalue	
SelectPartialMatch	Select an item via a partial text match.	ComboEditBox	^TextValue	
SetTextValue	Set the text of the ComboEditBox	ComboEditBox	^TextValue	

Tabela 3.1 - Funções WRAFS disponíveis para componentes do tipo ComboEditBox

3.2.2. Bibliotecas de Suporte (Support Libraries)

A **Biblioteca de suporte** ou *Support library* fornece rotinas e utilitários de propósito geral que são utilizadas internamente pela *framework* WRAFS para realização de suas tarefas. As funcionalidades principais fornecidas são as seguintes: tratamento de ficheiros, *strings*, *buffers*, variáveis; funções de acesso à base de dados, utilitários para registo de actividades em log, entre outros.

3.2.3. Mapa da Aplicação (Application Map)

Application map ou **mapa da aplicação** mantém a informação dos componentes gráficos existentes na interface da aplicação de teste. Serão sobre estes componentes que as funções de componente irão actuar. Normalmente, para cada elemento gráfico existente na aplicação alvo deve existir uma entrada no mapa da aplicação, com as propriedades do elemento necessárias para a ferramenta WinRunner o reconhecer. Todas estas entradas ficam registadas no mapa da aplicação. Para o WinRunner, o mapa da aplicação corresponde ao ficheiro guimap, introduzido no Capítulo 2.

O mapa da aplicação é um elemento fundamental para que a ferramenta WinRunner consiga interagir com a aplicação alvo e para a criação dos testes através da *framework* WRAFS. Não será possível definir os testes da aplicação em sua totalidade enquanto não se tiver construído o mapa da aplicação, dado que, as operações contidas nas tabelas de baixo nível actuam sobre os objectos do mapa da aplicação.

Através do uso da função *Learn* fornecido pela ferramenta WinRunner é possível construir automaticamente por reflexão um mapa da aplicação. Esta funcionalidade serve para captar todas as propriedades e valores de propriedades do objecto gráfico que se queira registar no mapa da aplicação. Todos os objectos que estão envolvidos durante a realização dos testes são registados no mapa. Para cada elemento gráfico incluído no mapa é associado automaticamente um identificador baseado no valor da propriedade *label ou name*, atribuída pelo programador. É frequentemente necessário que este identificador, que será o nome usado para referenciar os objectos a partir das tabelas de teste, seja alterado para um nome mais sugestivo. Outro propósito desta alteração é introduzir uma camada de protecção aos testes criados, que continuarão inalterados caso haja alteração no nome dado pelo programador ao objecto numa nova *release* do sistema.

A Figura 3.3 ilustra o conteúdo de um mapa da aplicação com referências aos objectos gráficos de uma janela de *Login* simples. A Figura mostra a referência a quatro objectos gráficos correspondendo a quatro entradas no mapa da aplicação. As palavras em negrito correspondem aos identificadores de objecto, que serão usados nas tabelas de teste.

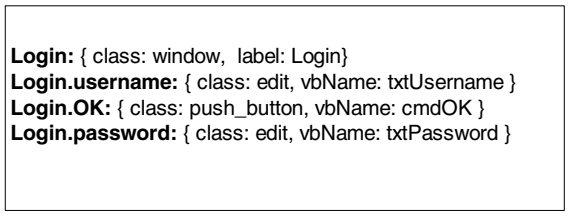

Figura 3.3 - Conteúdo do mapa da aplicação para o exemplo de janela de Login

O primeiro objecto, corresponde a própria janela de Login, tendo a propriedade *class* com valor igual a *Window*. O identificador deste objecto é igual ao conteúdo de sua propriedade *label*, facto que corresponde ao comportamento *default* nos mapas de aplicação criados pelo WinRunner. A seguir são referidos dois objectos gráficos do tipo *edit*, com identificadores iguais a *username* e *password* respectivamente. Por fim, há a referência ao objecto do tipo *push_button* que tem identificador igual a OK. Na Figura 3.3, o identificador *username* é precedido do identificador *Login*. Isto significa que, o objecto gráfico *username* se encontra sobre a janela *Login*. Na construção dos testes este prefixo será o conteúdo informado na coluna *Context* e na maioria das vezes corresponderá a um identificador dum objecto do tipo *Window*.

3.2.4. WRAFS Driver e WRAFS Engine

Dado a existência de diferentes formatos de tabelas de teste, contidas em ficheiros de texto, foram criados pela *framework* diferentes interpretadores: *CycleDriver, SuiteDriver, StepDriver.* Cada interpretador corresponde a um *script* WinRunner, os quais em conjunto realizam o processamento das tabelas de teste.

Para além da tarefa de processamento de tabelas de teste, cada tipo de interpretador fornece um conjunto particular de comandos, chamados de comandos *Driver*. Estes comandos realizam diferentes tarefas durante o processamento das tabelas de teste. São comandos complementares que auxiliam no bom funcionamento de toda a *framework*. *SetApplicationMap* é um exemplo de comando *Driver* que faz o carregamento o mapa de aplicação a ser usado. Ao conjunto dos três interpretadores e seus respectivos comandos *Driver* dá-se o nome de ***WRAFS Driver*** ou ***Core DDE*** (*Data-Driven Engine*) [WRAFS2005].

O ***WRAFS Engine,*** por sua vez, é composto pelo conjunto das **funções de componentes** e da **biblioteca de suporte**.

3.2.5. Tabelas de Teste

Tabelas de Teste (*Test Tables*) correspondem a ficheiros de texto que são processados pela *framework WRAFS* para a realização dos testes. Em seu conteúdo existem instruções para a realização de acções de teste. Tais tabelas podem ser criadas a partir de um editor de texto ou qualquer outro programa que exporte registos delimitados, tal como *Microsoft Excel* ou *Access*.

Existem três tipos de tabelas de teste: Tabelas de Nível Alto, Intermediário ou Baixo também conhecidas como *Cycle Tables, Suite Tables, Step Tables*, respectivamente. Cada tipo de tabela possui um formato interno e propósito particular que deve ser conhecido pelo testador para a criação dos testes.

3.2.6. Tabelas de nível baixo

As tabelas de baixo nível contêm instruções detalhadas para a realização das operações de teste. Cada registo de tabela de nível baixo, normalmente, contém uma chamada a uma **função de componente** para a realização da acção sobre um objecto gráfico.

Em cada linha da tabela de nível baixo são indicados o comando, o documento, o componente, a acção a ser realizada e os argumentos da operação, caso existam.

O documento representa o contexto em que o componente gráfico se encontra e indica o nome da janela que contém o componente. O nome do componente deve ser o nome de qualquer identificador de objecto existente no mapa da aplicação, sobre o qual se quer realizar a acção. A acção é uma função de componente permitida para o tipo de componente gráfico indicado. A Figura 3.4 ilustra as posições dos campos dum registo numa tabela de nível baixo e seus respectivos conteúdos.

Posição	Nome	Conteúdo
1	Comando	Contém o tipo do registo
2	Documento	O nome da janela onde as acções serão realizadas
3	Componente	O nome do componente gráfico que receberá a acção
4	Acção	A acção a ser realizada ou função de componente
> 4	Argumentos	Parâmetros passados à função de componente

Figura 3.4 - Campos dum registo de tabela de nível baixo (adaptada de [WRAFS2005])

A Figura 3.5 ilustra o conteúdo de uma tabela de nível baixo que descreve uma operação exemplo de inserção de cliente, chamada de **Inserir_Cliente**. Para o exemplo desta operação considera-se que as únicas informações requeridas são: nome, nif e endereço do cliente.

As variáveis "^nome" , "^nif" e ^endereco irão conter os valores a serem inseridos nos componentes gráficos *label_nome, label_nif* e *label_end* respectivamente. Estes componentes devem pertencer ao mapa da aplicação. A atribuição dos valores a estas variáveis normalmente acontece na tabela de nível intermediário que invocou esta tabela de nível baixo. Este exemplo será desenvolvido ao longo deste capítulo para melhor entendimento.

Inserir_Cliente.STD				
Comando	**Documento**	**Componente**	**Acção**	**Argumento**
T	W_cliente	Label_nome	SetTextValue	^nome
T	W_cliente	Label_nif	SetTextValue	^nif
T	W_cliente	Label_end	SetTextValue	^endereco
	W_cliente	Bt_inserir	Click	

Figura 3.5 - Conteúdo de tabela de nível baixo para o exemplo da operação Inserir_Cliente

Em resumo, uma tabela de teste de nível baixo equivale a um *script* de teste, dado que ela contém toda a informação necessária, acções e dados, para que *framework*, através da ferramenta WinRunner, realize o conjunto de acções nela descritas. O módulo *StepDriver* é o módulo responsável por processar todas as instruções encontradas nas tabelas de nível baixo.

3.2.7. Tabelas de Nível Intermediário

As tabelas de nível intermediário, não contêm instruções detalhadas para a realização do teste como as encontradas na tabela de nível baixo. As tabelas de nível intermediário contêm referências a um conjunto de tabelas de teste de nível baixo, agrupadas com um objectivo particular. Esta hierarquização facilita a criação de testes reutilizáveis, visto que muitas tabelas de nível baixo poderão ser utilizadas em diferentes tabelas de nível intermediário. As tabelas de nível intermediário são tratadas pelo módulo *SuiteDriver* que passa o controle para o *StepDriver* a cada

tabela de nível baixo encontrada. A Figura 3.6 ilustra as posições dos campos dum registo numa tabela de nível intermediário e seus respectivos conteúdos.

Posição	Nome	Conteúdo
1	Comando	Contém o tipo do registo
2	Acção	Nome de uma tabela de nível baixo ou um comando *Driver*
> 2	Argumentos	Parâmetros passados à tabela de nível baixo ou comando *Driver*

Figura 3.6 - Campos dum registo de tabela de nível intermediário

A Figura 3.7 ilustra uma tabela de teste de nível intermediário chamada **Clientes.SDD**. Esta tabela contém referências a tabelas de nível baixo, entre as quais, a Inserir_cliente.STD, ilustrada na Figura 3.5. A acção ***SetVariableValues*** é um exemplo de **comando *Driver*** que atribui valores às variáveis nome, endereco e nif que serão usadas na tabela Inserir_Cliente.SDD de nível inferior.

Clientes.SDD				
Comando	**Acção**	**Argumento1**	**Argumento2**	**Argumento3**
C	SetVariableValues	^nome = "Joaquim"	^endereco = "Rua do Alentejo"	^nif = "338382848"
T	Inserir_Cliente			
T	Consultar_Cliente	^nif = "338382848"		

Figura 3.7 - Conteúdo de tabela de nível intermediário para o exemplo de operações genéricas sobre a entidade clientes

3.2.8. Tabelas de Nível Alto

Tabelas de nível alto agrupam tabelas de nível intermediário em ciclos de teste. Este terceiro nível existente na *framework* permite que os testes sejam agrupados por tipos, como por exemplo: testes de regressão, aceitação, performance, etc. A utilização deste terceiro nível pode ter uma motivação específica a cada projecto de teste e testador. Por exemplo, cada ciclo ou tabela de nível alto poderá agrupar testes de uma determinada funcionalidade ou conjunto de funcionalidades da aplicação alvo. Este último nível permite que se tenha mais um nível para classificação e organização dos testes segundo a estrutura que for mais apropriada.

As tabelas de nível alto são processadas pelo *CycleDriver* que encaminha ao *SuiteDriver* cada tabela de nível intermediário que encontra. A Figura 3.8 ilustra as posições dos campos dum registo numa tabela de nível alto e seus respectivos conteúdos.

Posição	Nome	Conteúdo
1	Comando	Contém o tipo do registo
2	Acção	Nome de uma tabela de nível intermediário ou um comando *Driver*
> 2	Argumentos	Parâmetros passados à tabela de nível intermediário ou ao comando *Driver*

Figura 3.8 - Campos dum registo de tabela de nível alto

A Figura 3.9 ilustra o conteúdo de uma tabela de nível alto que agrupa os testes de aceitação de uma aplicação que faz a gestão de clientes. As acções ***SetApplicationMap*** e ***Pause*** são ***comandos Driver*** da *framework*. *SetApplicationMap* carrega o mapa da aplicação descrito no campo de argumento. A acção **IniciaAplicacaoAlvo** corresponde a uma tabela de nível intermediária que invoca o executável e efectua o *login* na aplicação alvo. A acção **Clientes** corresponde a tabela de nível intermediário mostrada na Figura 3.7.

TestesAceitacao.CDD		
Comando	**Acção**	**Argumento1**
C	SetApplicationMap	C:\mapa.gui
T	IniciaAplicacaoAlvo	
C	Pause	20
T	Clientes	

Figura 3.9 - Conteúdo de tabela de nível alto para os testes de aceitação de uma aplicação exemplo

3.3. Fluxo de Execução

Nesta secção é apresentado o fluxo de execução dos testes construídos sobre a *framework* WRAFS. A Figura 3.10 apresenta o diagrama de actividades correspondente ao fluxo. Neste diagrama cada actividade está associada ao módulo responsável por sua execução.

A execução dos testes é iniciada através do *script* ***StartCycleTest***. Este *script* carrega toda a biblioteca de *scripts* formada pelos módulos *Driver* e *Engine*. Depois de ter carregado os módulos em memória, o *sript StartCycleTest* invoca o módulo *CycleDriver* que irá ler o conteúdo da primeira **tabela de teste de alto nível**. O primeiro campo de cada registo (cada linha do ficheiro texto) indica o tipo do registo. O fluxo de execução será determinado sempre a partir do valor contido neste campo.

Se o tipo do registo lido em qualquer tabela de testes for "C", significa que o registo contém um **comando *Driver*** e o *script DDriverCommand* é chamado. Este *script* irá executar o comando *Driver* que estiver informado no segundo campo do registo. A execução volta para o controlo do módulo responsável *(CycleDriver, SuiteDriver ou StepDriver)* que irá ler o registo a seguir e assim por diante até que todos os registos tenham sido lidos.

Se o tipo do registo é igual a "T" e o módulo responsável pela execução é o *CycleDriver,* significa que o registo tem referência a uma tabela de nível intermediário e o controle é enviado para o *SuiteDriver*.

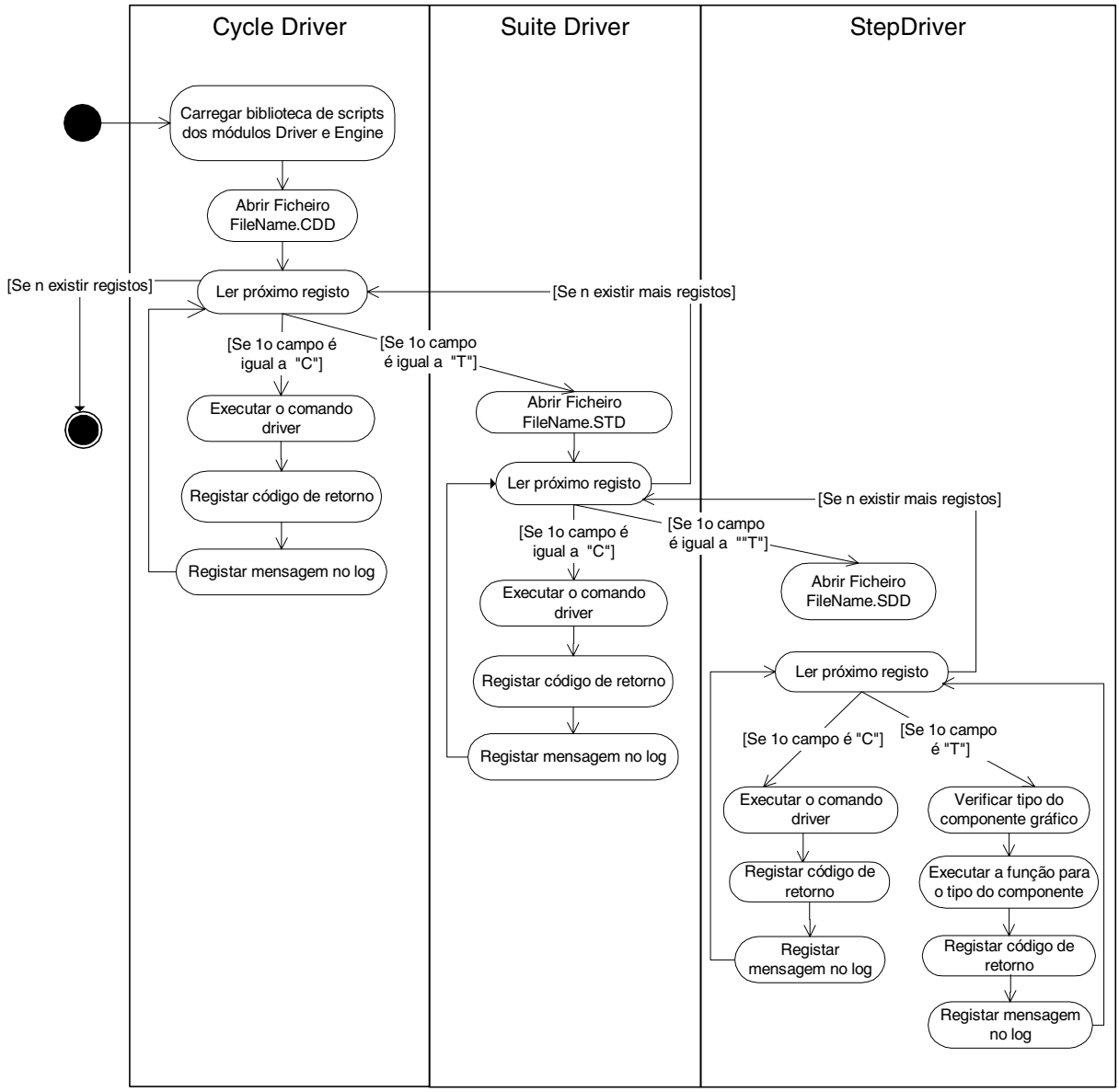

Figura 3.10 - Fluxo de execução pelos módulos CycleDriver, SuiteDriver e StepDriver

Se o tipo do registo é igual a "T" e o módulo responsável pela execução é o *SuiteDriver*, significa que o registo tem referência a uma tabela de nível baixo e o controle é enviado para o *StepDriver*.

Se o tipo do registo é igual a "T" e o módulo responsável pela execução é o *StepDriver,* significa que o registo contém uma **função de componente**. Por isto, irá obter os outros campos que compõem o registo nomeadamente, o nome da janela, o nome do componente e o nome da acção a ser executada, bem como os parâmetros. A partir do nome do componente o *script* irá obter o tipo do componente, através da informação contida no mapa da aplicação, e, finalmente invocará a função de componente sobre o respectivo componente gráfico. De seguida o controle é retornado

para o *StepDriver*. O *StepDriver* irá ler os registos seguintes da tabela de baixo nível e o processo continuará até que todos os registos da tabela sejam processados.

Depois de terminado o trabalho do *StepDriver*, o controle volta ao *SuiteDriver* que irá processar todos os registos restantes da tabela de nível intermediário e, finalmente, o controle voltará para o *CycleDriver* que irá processar todos os registos restantes da tabela de nível alto.

Após a execução de cada função de componente, o resultado é registado através das funções específicas para registo de histórico encontradas no *script LogUtilities* pertencente ao módulo ***WRAFS Engine.***

3.4. Verificação do Resultado da Execução

A *framework* WRFAS regista o resultado dos testes em ficheiros de Log. Como pode ser visto na Figura 3.9 da secção anterior, a cada comando *Driver* ou função de componente que se executa, o resultado é registado no ficheiro de Log correspondente. Existe um ficheiro de Log específico para cada tipo de tabela de teste que se esteja a executar nomeadamente "*StepLog.txt", "SuiteLog.txt"* e *"CycleLog.txt"*.

Caso a execução do comando ou função tenha sido realizada com sucesso, é registada uma nova linha no respectivo ficheiro de Log iniciada com a palavra OK, seguida de uma pequena descrição do que foi realizado. Por exemplo, as linhas da Figura 3.11 mostram o que seria registado no ficheiro StepLog.txt, para o caso de uma execução com sucesso da tabela de teste Inserir_Cliente.SDD da Figura 3.5. Supondo-se aqui que tenham sido passados os seguintes valores para as variáveis nome, nif e endereço respectivamente: "João", "33", "Rua da Beneficência".

```
OK   Label_nome      SetTextValue to João successful.
OK   Label_nif       SetTextValue to 33 successful.
OK   Label_end       SetTextValue to Rua da Beneficência successful.
OK   W_cliente:Bt_inserir CLICKED.
```

Figura 3.11 - Exemplo de conteúdo do Log para uma execução de tabela de teste com sucesso

Neste exemplo as palavras 'OK' indicam que: cada função de componente executada era válida para o tipo de objecto gráfico alvo, os dados de entrada eram válidos para o tipo de componente gráfico alvo e que a ordem de execução dos comandos é coerente com a ordem de navegação imposta pelo sistema alvo na janela w_cliente. No entanto, o teste Inserir_Cliente pode ter falhado mesmo tendo sido registadas as palavras OK no Log se, por exemplo, a aplicação alvo não inseriu de facto o cliente João na base de dados.

Isto significa que a tabela Inserir_Cliente.SDD não contém as acções para uma verificação do resultado do teste automática e sim, apenas as acções para a execução automática do teste. As acções necessárias poderiam ser acções de verificação de conteúdo da base de dados, através dos

comandos *Driver* existentes para este propósito ou acções para consulta da informação através da própria interface gráfica da aplicação alvo. De qualquer destas formas seria possível confirmar se o cliente "João" tinha sido inserido com sucesso em base de dados, caso este fosse o objectivo do teste Inserir_Cliente.SDD.

A inclusão destas novas acções na tabela de baixo nível para a verificação do resultado, implicaria na existência de mais informação a ser analisada pelo testador. Os ficheiros de Log constituem o único recurso disponibilizado pela *framework* para a verificação dos resultados de teste e, requerem sempre um esforço manual do testador para análise de seu conteúdo. O conteúdo de um ficheiro StepLog.txt pode tornar-se muito extenso e comprometer o trabalho de verificação dos casos de teste. Também a correspondência entre os casos de teste inseridos nas tabelas de teste e seu resultado final não é facilmente obtido e requer uma análise cuidadosa do testador sobre os ficheiros de Log.

3.5. Resumo

Este capítulo abordou o funcionamento da *framework WRAFS*, que possibilita a construção de testes automáticos a serem executados pela ferramenta Winrunner. Apresentamos a funcionalidade de cada componente da *framework*, a hierarquia a ser seguida para a construção dos testes bem como o formato dos ficheiros texto que são processados. Apresentamos o fluxo seguido no processamento de cada tipo de tabela de teste existente.

Capítulo 4

Problemas da Framework WRAFS

Como analisado no Capítulo 3, a *framework* WRAFS fornece um ambiente consistente para a construção de testes automáticos, através da utilização da técnica *keyword-driven* e da disponibilização duma linguagem para o desenvolvimento de testes, que abstrai os detalhes de uma linguagem de programação, como a TSL, fornecida pela ferramenta WinRunner.

Este capítulo, faz uma análise crítica aos aspectos menos conseguidos da *framework* WRAFS, os quais podem ser melhorados ou totalmente reestruturados. A análise feita foi baseada na experiência da autora no uso da ferramenta Winrunner e na utilização da *framework* para a construção de testes automáticos. A bibliografia existente sobre a *framework* foi explorada, incluindo-se nomeadamente contactos com a equipa de desenvolvimento da própria *framework.*

A análise feita neste capítulo não põe em causa o mérito da *framework* e sua importância, mas identifica seus pontos menos positivos para a seguir propor um caminho alternativo de trabalho para construção de testes automáticos.

Neste capítulo cada secção aborda uma limitação ou problema da *framework.* A Secção 4.1 apresenta as restrições decorrentes do formato de construção manual dos testes; a Secção 4.2 apresenta os problemas decorrentes do formato dos ficheiros de texto para construção dos testes; a Secção 4.3 apresenta o formato existente para visualização da hierarquia dos testes; a Secção 4.4. analisa o esforço necessário para a verificação dos resultados de teste e a Secção 4.5 aborda a falta de recursos disponíveis para registo dos resultados das execuções.

4.1. Construção Manual de Testes

Vimos no Capítulo 3 que a *framework* WRAFS fornece um ambiente para a construção de testes que são automaticamente executados, no entanto a construção de tais testes, embora facilitada, continua a ser feita manualmente. O testador precisa conhecer, não a linguagem TSL do WinRunner, mas a linguagem interna da *framework*. Isto inclui o conhecimento das funções de

componentes para cada tipo de componente gráfico, dos nomes dos identificadores dos objectos do mapa da aplicação, dos comandos dos *Drivers* disponíveis, do formato para criação de cada tipo de tabela de teste e da relação entre estes conceitos e estrutura interna da *framework*. Através deste conhecimento, as tabelas de teste são criadas e modificadas manualmente até que os testes necessários tenham sido criados sobre a *framework*. Finalmente, os testes são executados automaticamente pela ferramenta WinRunner.

A assinatura das funções de cada componente (que inclui o nome da função, o tipo de componente sobre o qual a função pode ser aplicada e os parâmetros que são passados) são disponibilizados pela *framework* segundo um formato tabular de ficheiro Excel (como mostrado na Tabela 3.1 do Capítulo 3). O testador precisa de consultar manualmente este ficheiro e seguir as suas instruções para a criação de testes.

Para os testadores com conhecimento prévio em testes automáticos e uso de ferramentas *Capture-Replay* a tarefa de construção dos testes através da *framework* não é difícil. Isto porque muitos dos conceitos existentes na *framework* são mapeados directamente na automação de testes através de *scripts* WinRunner. O formato de construção dos testes é diferente, mas está suportado por conceitos comuns às duas abordagens de construção: *scripts* WinRunner e *framework* WRAFS.

A Figura 4.1 mostra o conteúdo de um ***script* WinRunner** com as acções necessárias para a inserção de um registo de cliente através de uma determinada interface gráfica. O exemplo mostra uma versão simplificada, onde apenas o nome, endereço e tipo do cliente são informados. A Figura 4.2 mostra o conteúdo de uma **tabela de nível baixo** para execução das acções para inserção de um registo através da mesma interface gráfica da Figura 4.1.

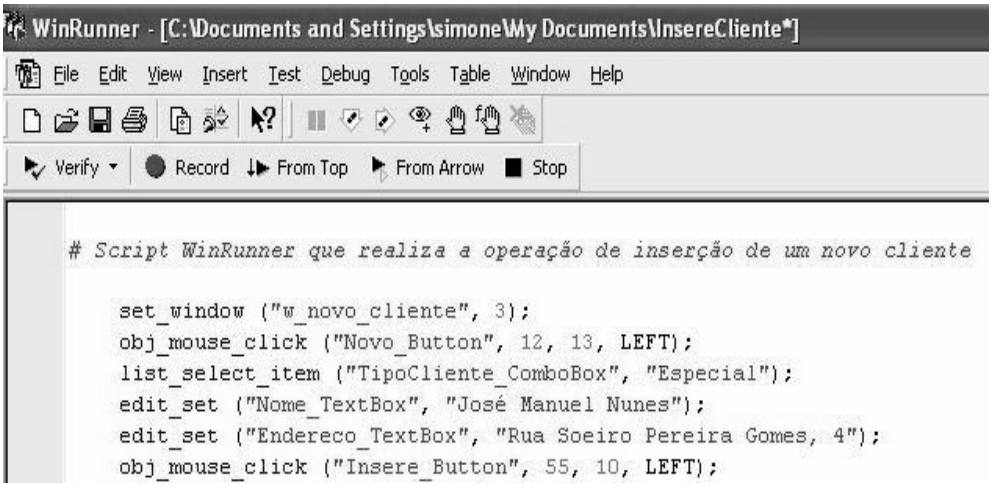

Figura 4.1 - Script WinRunner que corresponde a acção de inserção de novo cliente

O conceito de identificadores de objectos gráficos existentes num mapa de aplicação é comum a ambas abordagens. Na Figura 4.1, o componente Nome_TextBox corresponde ao mesmo

componente gráfico que na Figura 4.2 está referenciado pelo identificador Nome_Cliente. Em ambas abordagens é utilizado o conceito de mapa da aplicação.

As **funções de componente**, existentes na *framework,* correspondem às funções WinRunner que são utilizadas para interacção com os objectos gráficos. Na Figura 4.2, a função de componente *SetTextValue* corresponde às funções WinRunner *edit_set* da Figura 4.1. Apesar de ser identificado conceitos correspondentes, como pode ser comparado entre as figuras 4.1 e 4.2, a linguagem e formato oferecidos pela *framework* WRAFS faz com que a construção dos testes automáticos seja mais simples do que construção correspondente utilizando a linguagem TSL disponível. Ao argumento que possa dizer que a linguagem TSL fornece mais recursos de implementação que os permitidos através da linguagem e estrutura da *framework*, pode ser contraposto que a *framework* WRAFS encontra-se bastante evoluída e continua a ser desenvolvida de modo a ganhar ainda mais flexibilidade.

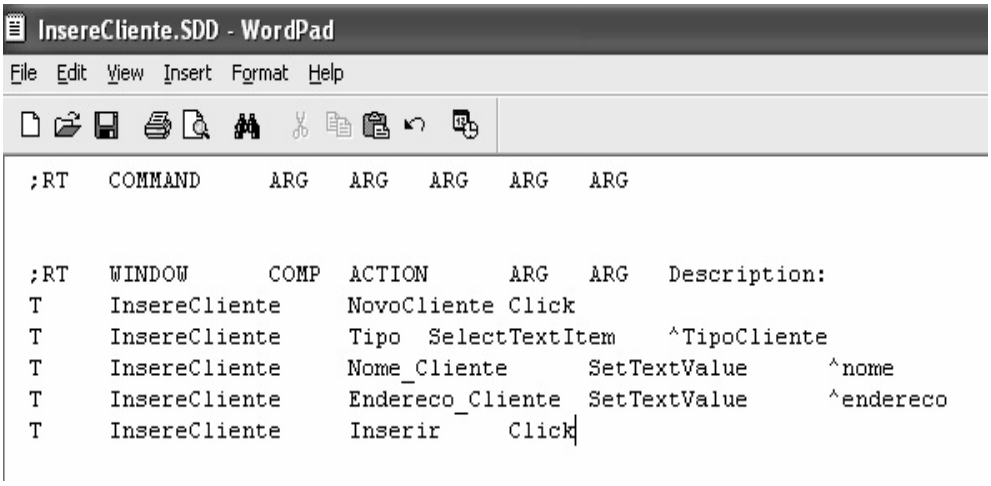

Figura 4.2 - Tabela de Baixo Nível da Framework WRAFS com acções que realizam a inserção de novo cliente

Para testadores sem conhecimento prévio em testes automáticos e ferramentas *Capture-Replay*, todos estes conceitos são novos e, apesar da *framework* abstrair detalhes e complexidades da ferramenta WinRunner, a construção dos testes lhes exigirá um esforço de aprendizado relativamente alto.

Estes testadores, apesar de não dominarem a tecnologia das ferramentas de automação de testes, muitas vezes possuem um conhecimento especializado na área do negócio da aplicação alvo e experiência na construção de testes manuais efectivos. Testadores com este perfil, peritos na identificação e desenho de casos de testes de qualidade, mas sem conhecimento em automação de testes através de ferramentas *Capture-Replay*, estão impossibilitados de utilizar a *framework* WRAFS para a construção de testes automáticos.

Uma abordagem alternativa, para uso da *framework*, seria de utilizar dois perfis de testadores para criação dos testes. Aos testadores com conhecimento em automação de ferramentas *Capture-Replay*

seria atribuída a tarefa de construção das **tabelas de nível baixo**, isto porque é neste nível normalmente que os conceitos de funções de componentes e mapa da aplicação são utilizados. Aos testadores sem conhecimento em automação, mas com conhecimento nos requisitos da aplicação alvo e experiência na área dos testes, seria atribuída a tarefa de construção das **tabelas de nível intermediário** e **alto**. Mesmo com esta abordagem, estaria sendo exigido aos testadores o conhecimento mínimo do formato de construção das tabelas de teste, segundo as regras da *framework*.

O trabalho decorrente desta tese propõe uma solução mais efectiva a este cenário, não suportado pela *framework* WRAFS.

4.2. Formato para a construção dos testes

Os ficheiros do tipo de texto que contém as **tabelas de teste** da *framework* WRAFS, tal como o visualizado na Figura 4.2, oferecem uma interface muito acessível para construção dos testes, no entanto possui limitações decorrentes de um formato texto.

Apesar de ser um formato suportado por ferramentas amplamente difundidas - os editores de texto – e, não exigir conhecimento especializado do utilizador, muitos problemas para a execução dos testes podem ser introduzidos caso as tabelas não sejam criadas conforme o formato exigido pela *framework.*

Para minimizar potenciais erros de edição o testador precisa conhecer em pormenor o formato dos três tipos de tabelas de teste. Embora não pareça evidente a dificuldade para a utilização do formato exigido, é de salientar que são os pequenos erros de digitação, a falta de um separador entre os campos de um registo que podem fazer com que o teste não possa ser executado e o testador passe bastante tempo a tentar perceber qual a falha na tabela de teste.

No exemplo da Figura 4.3 são destacados dois exemplos que mostram como pequenas falhas (erros de edição) na formatação dos registos de teste implicarão erros de execução nos testes planeados.

Na primeira linha de comando desta tabela, três espaços separam o nome do objecto gráfico e o nome da função a ser aplicada sob o mesmo. O nome do componente em questão é “NovoCliente” e o nome da função é “Click”. No entanto, uma vez que o separador entre campos da tabela de teste está definido para ser um “tab”, durante a execução desta linha, a *framework* considera que o nome do objecto é então “NovoCliente Click” e não apenas “NovoCliente”, como deveria. Este engano interromperá a execução do teste, sendo a seguinte mensagem de erro enviada ao ecran: ***Can Not Find the Object: “NewFlight Click”***. O teste fica interrompido neste ponto e nenhuma acção posterior é executada até que se corrija o ficheiro de teste e substitua-se os três espaços por um “tab”.

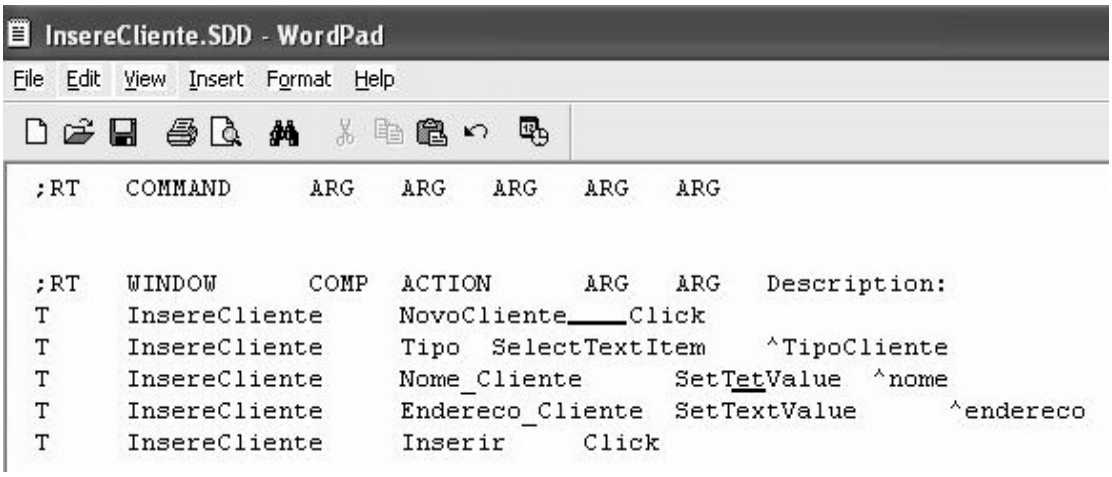

Figura 4.3 - Tabela de nível baixo construída com pequenos erros de edição

Na terceira linha mostra-se um erro de digitação no nome da **função de componente**: *SetTextValue*. Ao executar esta linha, a *framework* não reconhece o nome da função mas continua a execução das linhas seguintes sem nenhum alerta ser enviado ao Log. O nome do cliente fica por ser preenchido e a seguinte mensagem de erro interrompe a execução do teste: ***Window: "InsereCliente", Object: "Inserir", Error: "Object is currently disabled".*** Este erro é uma consequência do engano no nome da função. Como a aplicação alvo apenas habilita o botão de inserção após todos os campos terem sido preenchidos, este botão está desabilitado quando o WinRunner tenta executar o último comando da tabela de teste, que faria o *click* no botão Inserir.

Estes são exemplos de situações que ocorrem devido à construção manual dos testes em ficheiros de texto, requeridos pela *framework* WRAFS. Num cenário de testes complexos e com grande quantidade de acções, maior será a probabilidade de se ter mais erros na formatação, que levarão a erros de execução dos testes e esforço do testador, até que os possa detectar e corrigir.

Em suma, o formato texto, embora acessível para qualquer utilizador, apresenta os problemas descritos nesta secção. Esta tese fornecerá uma solução para os problemas decorrentes do formato texto para a construção dos testes através da *framework* WRAFS.

4.3. Visão desorganizada dos testes criados

A existência de três níveis de tabelas de testes tem o objectivo principal de oferecer mecanismos para reutilização de testes. Cada tabela de teste pode ser utilizada em diferentes tabelas de nível imediatamente superior.

Apesar das vantagens da hierarquia das tabelas, verifica-se que a gestão dos testes criados através da *framework* torna-se uma tarefa complexa. Cada tabela criada corresponde a um ficheiro de texto que existe numa certa directoria. Para que um testador, durante a criação de um novo teste, saiba quais os testes ou tabelas de nível inferior tem disponível para inclusão, precisa entrar na directoria

correcta e verificar pelo nome e conteúdo dos ficheiros existentes para encontrar aquele teste que necessita incluir em sua nova tabela de teste.

Um problema semelhante coloca-se quando os testes já estão criados e é necessário realizar alguma alteração. Para se determinar, a partir dos testes criados os pontos nas tabelas de teste que sofrerão alteração, será necessário navegar-se através da hierarquia de testes, abrindo-se e fechando-se ficheiros texto correspondentes aos testes.

Mesmo no cenário mais simples, onde os testes já estão criados e deseja-se apenas consultá-los, é difícil visualizar a relação entre as tabelas de testes criadas. Através da Figura 4.4 é possível visualizar o conteúdo das três directorias existentes para cada tipo de tabela de teste esperada pela *framework* WRAFS. Em cada directoria, os nomes dados aos ficheiros podem sugerir os testes que poderiam ser criados, mas não indicam como efectivamente estão criados. A relação entre os testes e a ordem de execução não é conhecida a menos que se consulte o conteúdo dos ficheiros.

Figura 4.4 - Tabelas de teste de nível alto, médio e baixo, existentes em cada directoria

Em secção quis ressaltar que, apesar de existir uma hierarquia lógica entre as tabelas de testes criadas sobre a *framework* WRAFS, esta hierarquia não é visualizada facilmente. Esta tese fornecerá uma solução para este problema.

4.4. Verificação dos Resultados

A *framework* disponibiliza para verificação dos resultados dos testes ficheiros de Log, onde é registada o sucesso ou falha de qualquer acção. Este formato requer algum esforço de análise do testador, que tem que realizar uma análise cuidadosa sobre os três níveis de Log que correspondem aos três tipos de tabelas de teste.

A relação entre os testes criados e o resultado dos mesmos não é facilmente analisada. O testador deve fazer uma análise, linha a linha, nos ficheiros de Log e identificar a que testes correspondem aqueles resultados de execução, sendo pois que esta informação não é facilmente visualizada.

4.5. Histórico de Resultados

Actualmente a *framework* regista nos ficheiros de Log apenas o resultado da última execução realizada. Por conseguinte, não existe um recurso para manutenção do histórico dos resultados das execuções dos testes.

Com esta limitação, não é possível reportar-se a natural evolução da estabilidade do sistema através da fase de testes.

Se existisse um histórico da execução de testes, o diagnóstico das possíveis causas dos problemas poderia ser feito mais facilmente.

Capítulo 5

AGEAT: Análise e Concepção

A proposta desta dissertação é de expandir as funcionalidades da *framework* WRAFS, criando um sistema para **geração automática de testes** que poderão ser **automaticamente executados** sobre a **ferramenta WinRunner**. O sistema funciona com base na estrutura da *framework* WRAFS, mas fornece funcionalidades acrescidas que solucionam os principais problemas identificados no Capítulo 4.

Este capítulo apresenta a definição das funcionalidades, os aspectos de análise, concepção e desenho do sistema AGEAT. A Secção 5.1 apresenta a ideia central que motivou a construção do sistema, a Secção 5.2 apresenta os requisitos não funcionais, a Secção 5.3 apresenta os requisitos funcionais, a Secção 5.4 apresenta o modelo de domínio, a Secção 5.5 apresenta o modelo de casos de uso do sistema desenvolvido e por fim a Secção 5.6 apresenta a arquitectura de componentes do sistema AGEAT.

5.1. Concepção

Na área dos testes de sistema e de utilizador constata-se que diferentes aplicações são testadas através da realização de acções simples que manipulam a informação dos objectos de negócio, através do uso de uma interface gráfica particular oferecida. Para um *Sistema de Facturação*, por exemplo, os objectos ou entidades de negócio como Cliente, Produto, Factura, são manipulados através da interface gráfica do respectivo sistema.

O testador tem conhecimento das regras de negócio que estão a ser exercitadas durante a realização das operações de teste, mas cada operação é composta por um conjunto de acções que individualmente constitui uma interacção sobre um componente gráfico, enquadrado na interface do sistema alvo.

As operações **CRUD** (***C**reate, **R**ead, **U**pdate, **D**elete*) são frequentemente realizadas durante a realização dos testes através da interface gráfica do sistema. A combinação destas operações

juntamente com as variações dos dados de entrada fornecidos, permitem a criação de variadas baterias de teste.

Cada operação é realizada normalmente sobre uma ou mais janelas da interface gráfica. Cada janela, por sua vez, é composta de vários elementos gráficos que são manipulados durante a realização da operação. As formas de interacção em cada tipo de componente gráfico são, em último caso, limitadas pelas funções disponíveis na linguagem TSL [RunnerManual76] do WinRunner. Caso o testador esteja a usar a *framework* WRAFS, as possibilidades de interacção são definidas pelas **Funções de Componentes** disponíveis. Embora existam várias possibilidades de realizar a interacção, é possível identificar um subconjunto restrito de acções, para cada tipo de elemento gráfico, que são frequentemente utilizadas na realização de cada operação. Por exemplo, durante uma operação de criação, a acção frequentemente realizada pelo testador sobre um componente do tipo *ComboBox* é a acção que corresponde à selecção de um texto da lista de itens apresentada no *ComboBox.* Esta acção corresponde à **função de componente** "*Select"* da *framework* WRAFS ou à **função WinRunner** "*list_select_item"*. No entanto, existem outras acções disponíveis para este tipo de objecto gráfico, como pode ser visto na Tabela 3.1 do Capítulo 3, mas na maioria das vezes apenas a função *Select* é usada pelo testador, sendo as outras funções utilizadas de forma menos habitual.

Com base nesta análise, após diversas experiências com a ferramenta WinRunner e a *framework* WRAFS, foi desenvolvido como trabalho prático desta dissertação, um sistema que permite não apenas a execução automática de testes, mas, principalmente, a geração automática de testes *data-driven*. Desta forma, pretendemos manter as vantagens inerentes à execução automática, mas **minimizar o esforço de construção e manutenção dos testes**.

Todos os testes que são realizados manualmente por um testador através de acções sobre a interface gráfica podem ser automaticamente gerados e executados pelo sistema. Por exemplo, o sistema pode ser utilizado para testes que realizem uma combinação de operações CRUD com variações de dados de entrada, para qualquer sistema alvo com interface gráfica que seja suportada pela ferramenta Winrunner.

Este sistema, denominado **AGEAT** (**A**mbiente de **G**eração e **E**xecução **A**utomática de **T**estes), utiliza, como plataforma de suporte, a *framework* WRAFS, mas atende às necessidades daqueles que não queiram enfrentar os problemas apresentados no Capítulo 4, inerentes da utilização directa da *framework.* Através do AGEAT, os testadores não precisam conhecer a *framework* WRAFS nem a linguagem WinRunner para a construção de suas baterias de teste. Como valor adicional, o sistema pode ser ainda utilizado como meio pedagógico para aqueles que queiram conhecer em pormenor a *framework* WRAFS.

A informação central utilizada pelo sistema AGEAT para a geração automática dos testes é baseada na informação obtida pela experiência na área de testes e na *framework* WRAFS, capaz de

relacionar os **tipos de objectos gráficos** existentes às **funções de componentes WRAFS** aplicáveis. Durante a fase de concepção do sistema, pensamos que a relação que associasse *Tipos de Objectos Gráficos* a *Funções WRAFS,* poderia ser mais flexível e depender de uma terceira variante a quem chamamos de Cenário. Um **Cenário** é composto de um conjunto de tipos de objectos gráficos e suas respectivas funções WRAFS.

O sistema AGEAT fornece toda a informação de configuração de um único cenário, chamado de cenário **Genérico**. No entanto o utilizador pode estender a funcionalidade do sistema e criar novos cenários caso seja necessário. O cenário genérico sugere as funções WRAFS mais frequentemente utilizadas durante a realização de operações CRUD em cada tipo de objecto gráfico.

Por exemplo, para a maioria das operações realizadas, a interacção feita sobre um objecto gráfico do tipo *edibox* será feita pela função *SetTextValue,* que insere o texto passado como parâmetro da função no objecto gráfico. Esta portanto será a função sugerida pelo cenário Genérico ou seja, a função, por omissão, utilizada pelo sistema para a geração automática dos testes.

Para outras situações, quando o testador já possui conhecimento em automação de testes e nas funções WRAFS ou deseja criar interacções diferentes das sugeridas, o sistema oferece a possibilidade de personalização das interacções, passando a estar disponíveis para utilização todas as funções de componentes disponíveis na *framework* WRAFS, na interface gráfica do sistema AGEAT.

A Figura 5.1 ilustra as dependências entre o sistema AGEAT e os componentes de software externos com quem se relaciona, nomeadamente: *Framework* WRAFS e Ferramenta Winrunner.

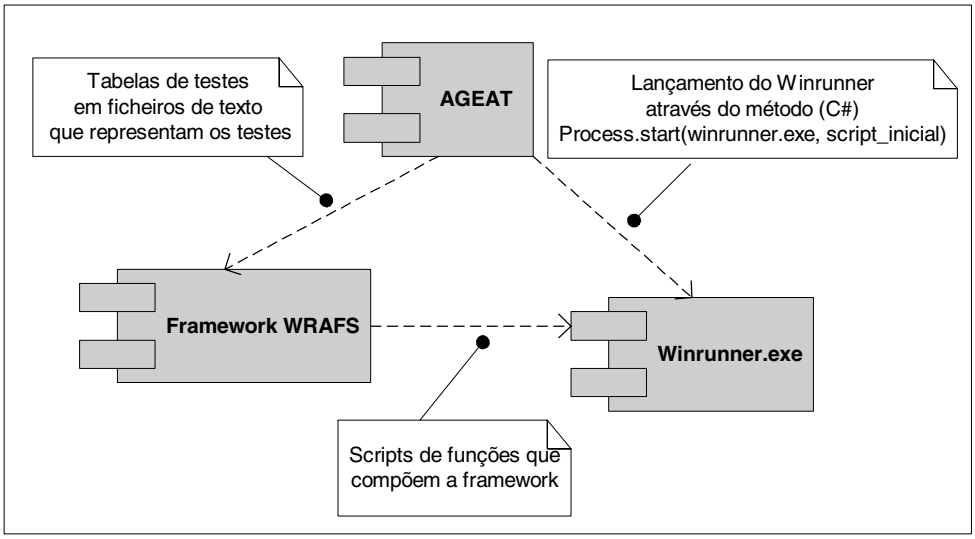

Figura 5.1 - Dependência entre os componentes de software

O desenvolvimento do AGEAT foi assente sobre a *framework* WRAFS, possibilitando a redução de tempo e riscos inerentes ao desenvolvimento do sistema. Os ficheiros de texto criados pelo sistema AGEAT, que corresponderão a tabelas de nível baixo, médio e alto para a *framework*, constituem o ponto de ligação entre as duas entidades.

Durante a execução dos testes, o sistema AGEAT não interage com a ferramenta Winrunner, ficando o controle da execução da responsabilidade da *framework*, através de todos os seus *scripts* TSL de funções existentes. No entanto, o início da execução dos testes é feito a partir do sistema AGEAT que invoca e carrega o Winrunner passando como parâmetro o *script* TSL ("script_inicial" na Figura 5.1) que inicia a execução de um ciclo de teste. Este *script* pertence à *framework* mas é alterado pelo sistema AGEAT para incluir a referência ao ficheiro texto que corresponde ao ciclo escolhido de ser executado. Desta forma, os detalhes da *framework* e do Winrunner são transparentes ao utilizador final do sistema AGEAT. Ao nível da instalação todos os componentes devem coabitar num mesmo nó físico.

5.2. Requisitos não funcionais

Os **requisitos não-funcionais** definem propriedades e restrições do sistema [Sommerville2000]. Foram identificados para este projecto os seguintes requisitos não-funcionais.

Usabilidade: o sistema desenvolvido deve proporcionar uma fácil gestão de testes automáticos a testadores que não tenham necessariamente conhecimento em ferramentas *capture-replay* e suas respectivas linguagens de programação. O AGEAT deve apresentar uma interface simples, consistente e fácil de utilizar que abstraia a complexidade da ferramenta Winrunner e da *framework* WRAFS.

Extensibilidade: deverá ser possível utilizar o sistema AGEAT para providenciar as mesmas funcionalidades a utilizadores que utilizem uma ferramenta do tipo *capture-replay* diferente da Mercury Winrunner, mas que seja suportada por uma *framework* do projecto SAFS, sem que para isto seja necessário alterar o modelo de domínio do sistema. Por exemplo, com ajustes nos dados de configuração, o sistema AGEAT poderá ser utilizado para construção de testes a serem executados sobre a ferramenta Rational Robot [RobotInfo], suportada pela *framework* RRAFS [WRAFS2005].

5.3. Requisitos Funcionais

Os **requisitos funcionais** descrevem os serviços que o sistema deve prover, como o sistema deve reagir a valores de entrada particulares e como o sistema deve comportar-se em situações particulares [Sommerville2000]. Os requisitos funcionais do sistema AGEAT, apresentados de seguida, solucionam a maioria dos problemas da *framework* WRAFS, apresentados no Capítulo 4.

5.3.1. Geração Automática de Testes

Tal como vimos, as **ferramentas *Capture-Replay*** apenas automatizam a execução dos testes, mas a sua construção é feita manualmente, através da linguagem de *script* fornecida pela ferramenta ou de uma linguagem de nível mais alto, como a linguagem fornecida pela *framework* WRAFS para construção das tabelas de teste.

O sistema AGEAT deve possibilitar a geração automática de testes para qualquer sistema alvo com interface gráfica reconhecida pela ferramenta WinRunner. Os testes gerados através do sistema AGEAT serão executados sobre a *framework* WRAFS, que por sua vez utiliza a ferramenta WinRunner para a execução dos testes. No entanto não será exigido que o testador conheça as assinaturas das funções de componentes, nem os comandos *Driver*, nem o correcto formato das tabelas de teste, apresentados no Capítulo 3. O sistema saberá escolher os comandos *Driver* e funções apropriadas e construir as tabelas de teste no formato exigido. Os problemas decorrentes de erradas formatações dos ficheiros texto correspondentes às tabelas de teste, não mais existirão.

Tal como acontece na área dos testes manuais, é apenas necessário que o testador conheça os requisitos da aplicação alvo e não que domine particularidades de uma *framework* ou linguagem de *script*. O conhecimento dos requisitos será sempre fundamental para que todas as condições de teste possam ser identificadas e exercitadas.

Inicialmente, o testador precisará informar ao sistema AGEAT como os elementos da interface gráfica do sistema alvo devem ser manipulados para a realização das operações sobre as entidades conceptuais do sistema. Isto constituirá pouco mais que informar a ordem de navegação através dos elementos gráficos da interface gráfica, mas será suficiente para o sistema AGEAT criar, no formato exigido da *framework* WRAFS, tabelas de testes de nível baixo para a realização de cada operação CRUD.

Depois de fornecidas estas informações, através da interface gráfica do sistema AGEAT, o testador irá utilizar as operações CRUD criadas para a construção de **baterias de teste** mais completas. Durante esta actividade o testador informará os dados de entrada para cada operação de teste e o sistema AGEAT terá a informação necessária para construir as tabelas de teste de nível intermediário.

Finalmente as baterias de teste criadas poderão ser agrupadas em ciclos, conforme a intenção do testador. Caso este queira agrupar as baterias de testes de acordo com o objecto de negócio que elas manipulam, poderá criar um ciclo de teste para cada objecto de negócio para o qual existam baterias de teste. Se o testador não quiser utilizar este nível de organização, pode ter como opção *default* todas as baterias agrupadas em um único ciclo.

Os testadores que já dominem a *framework* WRAFS poderão se beneficiar da interface gráfica fornecida pelo AGEAT para uma construção mais facilitada das tabelas de testes utilizadas pela *framework* WRAFS.

Em suma os problemas decorrentes da construção manual dos testes no formato de ficheiros texto, descritos no Capítulo 4, não são mais verificados com o uso do sistema AGEAT visto que não é necessário a interacção directa do testador com a *framework* WRAFS.

5.3.2. Execução Automática de Testes

Os testes criados através do sistema AGEAT deverão ser transformados em tabelas de testes da *framework* WRAFS, garantindo assim a sua execução automática no contexto da ferramenta WinRunner.

Mais uma vez, não deverão ser necessárias intervenções manuais na ferramenta WinRunner ou ficheiros da *framework* WRAFS. O WinRunner será invocado a partir do sistema AGEAT com o *script* de inicialização da *framework* devidamente alterado de modo a conter a referência para o ciclo a ser executado. Depois de solicitada a execução dos testes pela interface do sistema AGEAT, os testes do ciclo passam a ser automaticamente executados sob a interface gráfica do sistema alvo.

No final da execução, o resultado da execução poderá ser visualizado a partir do sistema AGEAT. A análise dos ficheiros Logs exigida pela *framework* WRAFS será substituída pela verificação sob a interface gráfica do sistema AGEAT, desenhada de modo a facilitar a interpretação dos resultados.

5.3.3. Visualização dos Testes Através de Árvore Hierárquica

Os testes criados pelo testador, a partir da combinação das operações instanciadas, serão criados no formato requerido pela *framework* WRAFS, embora o conhecimento deste formato não seja exigido ao testador. Isto significa que tabelas de testes serão geradas no *FileSystem* respeitando a hierarquia de três níveis exigida pela *framework* WRAFS.

No entanto, a dificuldade de visualização da hierarquia de testes em ficheiros de texto na *framework* WRAFS, constitui mais um dos problemas citados no Capítulo 4, que o sistema AGEAT ultrapassa. O sistema AGEAT permitirá a visualização gráfica, em forma de árvore, da hierarquia de testes criada. A visualização gráfica facilitará o trabalho de criação de novos testes e de alteração de testes existentes.

5.4. Modelo de Domínio

A Figura 5.2 apresenta o modelo de domínio que suporta o sistema AGEAT. De seguida, descreve-se sumariamente as principais classes envolvidas.

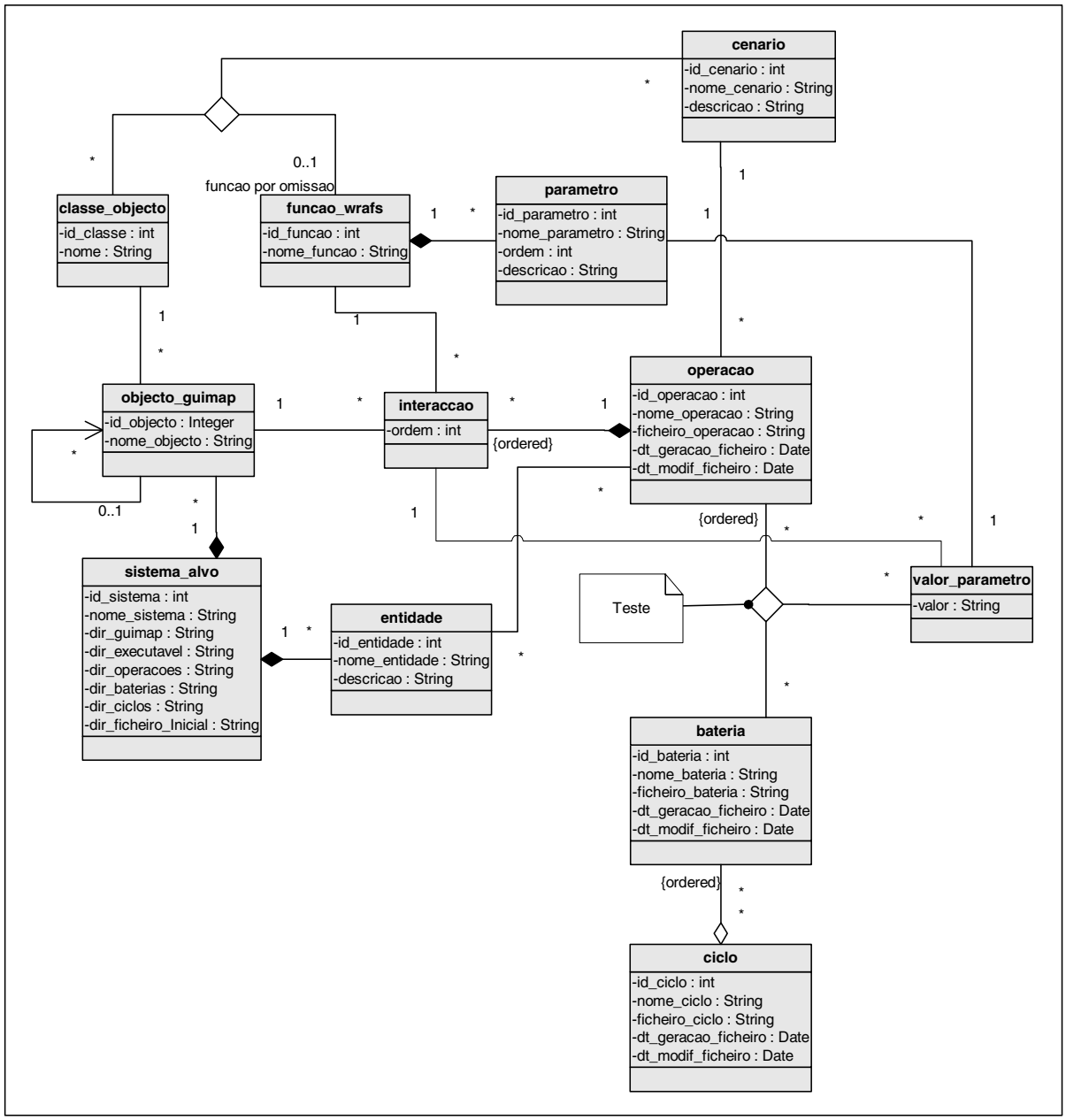

Figura 5.2 - Modelo de domínio suportado pelo AGEAT

Classes de Objectos, Funções WRAFS, Parâmetros e Cenários

O AGEAT foi desenvolvido com base na existência da *framework* WRAFS e das suas funções de componente. A definição das funções de componente WRAFS, através das quais o sistema poderá interagir com os variados tipos de objectos gráficos, foram levantadas a partir da documentação disponível e introduzidas no modelo de domínio criado para o sistema AGEAT. A definição de cada

função regista as seguintes informações: nome da função, nome dos parâmetros de entrada, ordem de posicionamento de cada parâmetro e a descrição do objectivo do parâmetro. A maioria das funções WRAFS não possui parâmetro de entrada ou possui apenas um parâmetro, porém o modelo criado para o sistema AGEAT está preparado para todos os tipos de funções de componentes disponíveis pela *framework*, independente do número de parâmetros recebidos pela função. É possível utilizar qualquer função WRAFS na criação dos testes através do sistema AGEAT.

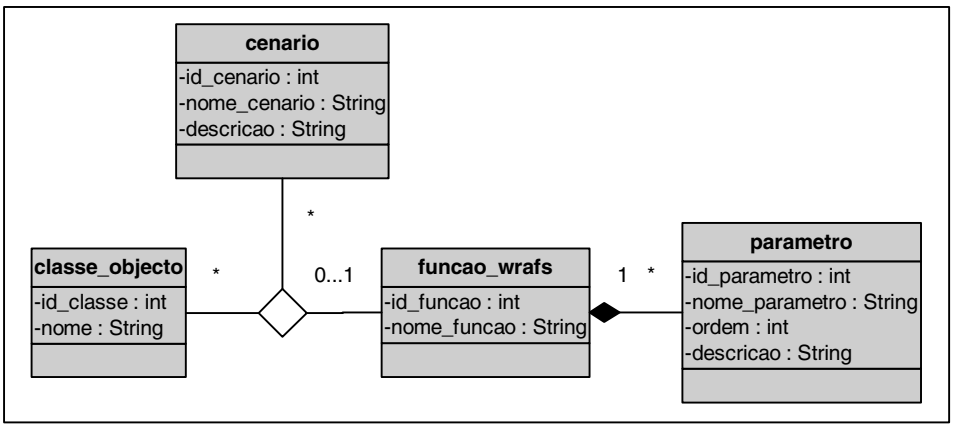

Figura 5.3 - Classes de base para o funcionamento do sistema AGEAT

O conceito de **cenário**, criado especificamente para sistema AGEAT, agrega, para um conjunto de classes de objectos gráficos, funções WRAFS. Um cenário incluindo todas as classes de objectos gráficos já é fornecido como informação básica do sistema. Este cenário, chamado de **Genérico**, associa para cada tipo de objecto gráfico existente a função WRAFS mais frequentemente utilizada. Novos cenários poderão ser introduzidos no sistema, para satisfazer necessidades específicas de testes. Os cenários são os elementos chave no sistema AGEAT, pois é com base neles que é suportada a funcionalidade de geração automática de operações. Na criação de cada operação de teste as funções WRAFS serão automaticamente sugeridas pelo sistema, para cada objecto gráfico, baseado no cenário escolhido.

Sistema alvo, Objectos GuiMap, Entidades

Os testes criados através do sistema AGEAT estão associados a um **sistema alvo** particular. A criação dos testes depende da criação e configuração e posterior selecção de um sistema alvo, através do sistema AGEAT. Parte da configuração do sistema alvo está relacionada com a estrutura de directorias estabelecida pela *framework* e que será respeitada pelo sistema. As operações feitas no sistema AGEAT estarão relacionadas ao sistema alvo que tiver sido seleccionado.

Associado ao sistema alvo estão todos os **objectos gráficos** do sistema, agrupados pela janela da qual fazem parte. Cada objecto gráfico janela é composto por uma série de outros tipos de objectos gráficos.

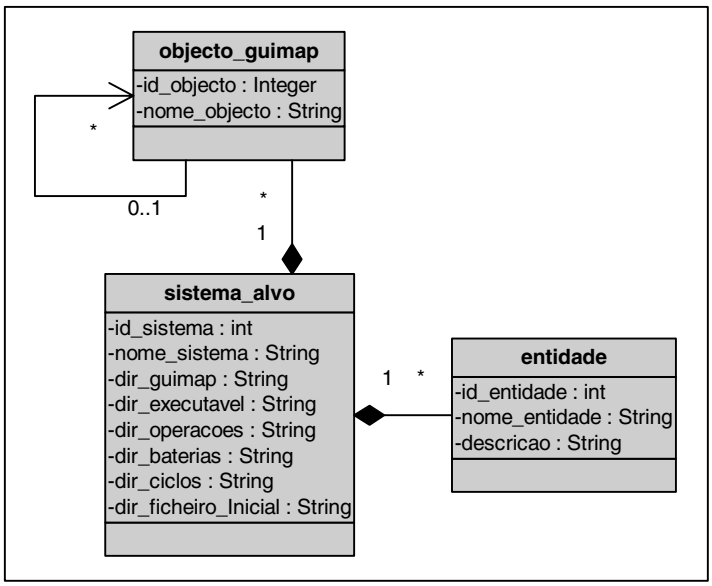

Figura 5.4 - Classes relacionadas ao sistema alvo a ser testado

Cada sistema alvo é composto pelo conjunto de **entidades** que faz sentido no respectivo contexto do negócio. As entidades possuem um nome e descrição. Cada operação de teste criada exercita um conjunto de entidades do sistema alvo.

Operação de teste

Uma operação de teste corresponde a uma sequência de interacções sobre objectos gráficos pertencentes a uma ou mais janelas do sistema alvo. A forma de interacção com cada objecto gráfico está determinada pela função WRAFS sugerida pelo sistema ou a função WRAFS posteriormente escolhida pelo testador.

Cada operação de teste criada exercita um conjunto de entidades do sistema alvo. Cada operação de teste terá um ficheiro texto associado que corresponde a uma tabela de nível baixo da *framework* WRAFS. O conteúdo deste ficheiro conterá as interacções registadas para a operação, no formato exigido pela *framework*.

Bateria de teste

Uma **bateria de teste** corresponde a uma sequência de operações de teste e seus respectivos valores de entrada. Uma mesma operação pode ser utilizada na construção de diversas baterias e uma bateria pode ser composta por várias operações. Esta estrutura permite a existência de uma bateria composta por uma sequência da mesma operação de teste. Uma bateria de teste pode ser composta por repetidas chamadas a uma mesma operação, com variados dados de entrada. Os valores de entrada passados a cada chamada à operação de teste numa bateria correspondem aos valores que serão passados aos parâmetros de entrada de cada operação WRAFS existente.

Cada bateria de teste terá um ficheiro texto associado que corresponde a uma tabela de nível médio da *framework* WRAFS. O conteúdo deste ficheiro conterá as chamadas às operações e seus respectivos valores de entrada, no formato exigido pela *framework.*

Ciclo de teste

Um ciclo de teste corresponde a uma sequência de baterias de teste com propósito comum. Uma mesma bateria pode ser utilizada na construção de diversos ciclos. O agrupamento das baterias em ciclos fornece mais uma possibilidade de organização dos testes. Um ciclo de teste indica um ponto de partida a partir do qual os testes serão executados.

Cada ciclo de teste terá um ficheiro texto associado que corresponde a uma tabela de nível alto da *framework* WRAFS. Este ficheiro conterá invocações às baterias de teste e a chamada ao comando *driver* que permite o carregamento em memória do ficheiro Guimap associado ao sistema alvo.

Teste

Um teste é uma sequência de uma ou mais operações de teste (pertencentes a uma bateria) e seus respectivos valores de entrada, associadas com um objectivo particular.

5.5. Modelo de Casos de Uso

Descreve-se nesta secção os actores do sistema, apresentando-se sumariamente as funcionalidades providenciadas pelo sistema AGEAT a cada um deles. As operações disponíveis para cada actor são descritas através de diagramas de casos de uso.

5.5.1. Principais Actores

O sistema AGEAT deverá suportar três tipos de actores (ver Figura 5.5): UAdministrador, UConfiguradorSistemaAlvo e UTestador.

UAdministrador: Este actor representa um utilizador com conhecimento da solução criada pelo sistema AGEAT para providenciar a forma automática de geração dos testes. As suas responsabilidades estão associadas fundamentalmente à gestão das funções da *framework* WRAFS e à gestão dos cenários.

UConfiguradorSistemaAlvo: Este actor representa um utilizador com conhecimento introdutório em ferramentas *capture-replay* e na *framework* WRAFS. Ele deve providenciar a informação de configuração necessária para que os testes sejam posteriormente criados com sucesso. As suas

responsabilidades estão associadas à criação das configurações associadas ao sistema alvo e à importação dos objectos gráficos do sistema alvo.

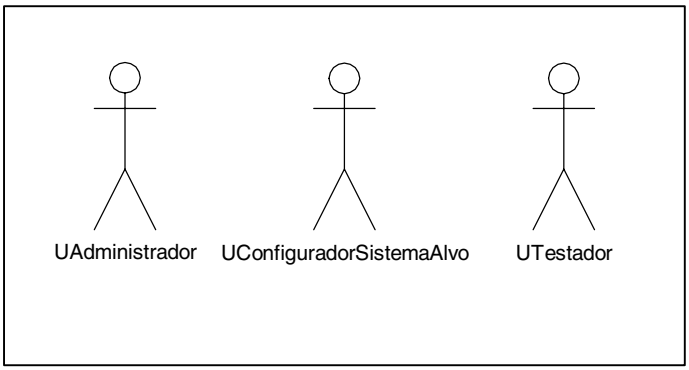

Figura 5.5 - Actores do AGEAT

UTestador: O utilizador testador representa os utilizadores com conhecimento nos requisitos do sistema a ser testado e por isto, indicado para a realização de seus testes. Este utilizador utilizará a interface gráfica de modo a exercitar o sistema e verificar que o mesmo atende aos objectivos necessários. As suas responsabilidades estão associadas à gestão das entidades conceptuais; à gestão da geração dos testes e à gestão da execução dos testes.

5.5.2. Casos de Uso

Esta secção descreve os casos de uso associados a cada actor identificado na secção anterior.

Para o utilizador **UAdministrador** foram definidos os casos de uso indicados na Figura 5.6.

Este actor é responsável por fazer a gestão da informação utilizada pelo sistema para a geração automática dos testes. Apesar do sistema AGEAT já disponibilizar, a partir instalação, todas as funções WRAFS actualmente existentes e o cenário Genérico com as funções, por omissão, a serem usadas para cada tipo de objecto gráfico, esta informação poderá ser actualizada sempre que necessário pelo actor **UAdministrador**. O actor poderá incluir no sistema novas funções WRAFS ou criar cenários com uma configuração diferente da existente no cenário Genérico.

Gerir funções WRAFS

Ao criar uma nova função WRAFS no sistema, o utilizador deverá informar o seu nome, os seus parâmetros de entrada (juntamente com uma descrição e a sua ordem de apresentação na chamada à função). De seguida deverá indicar em qual(is) tipo(s) de objecto(s) gráfico(s) a função pode ser utilizada.

Criar cenário

Ao criar um novo cenário o utilizador irá indicar para um subconjunto desejado de tipos de objectos gráficos as respectivas funções WRAFS a serem sugeridas pelo sistema aquando da geração das operações de teste associadas ao cenário em questão.

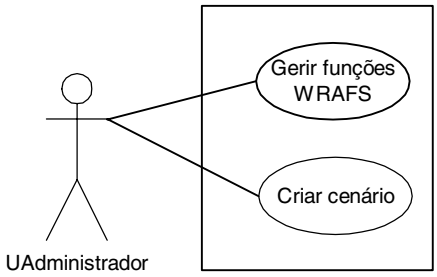

Figura 5.6 - Casos de uso do actor UAdministrador

O actor **UConfiguradorSistemaAlvo** é responsável por fornecer ao sistema AGEAT toda a informação referente ao sistema alvo, que seja necessária para a criação dos testes (Figura 5.7)

Uma de suas acções é realizada através da ferramenta Winrunner. A acção criar "ficheiro GuiMap do sistema alvo", requer um conhecimento de nível introdutório na ferramenta Winrunner, apenas para a utilização da função *Learn*, apresentada no Capítulo 3.

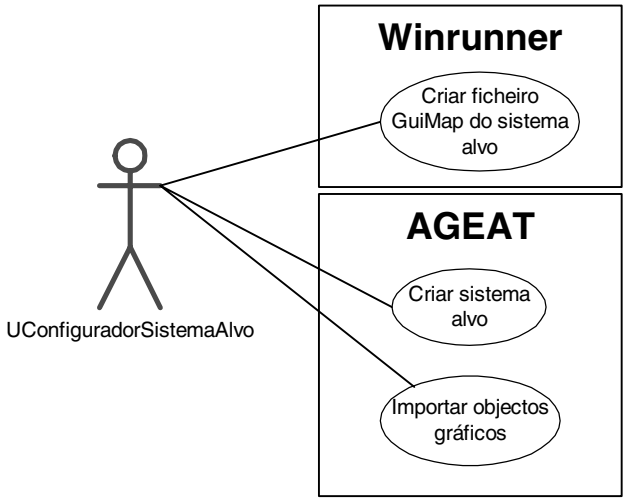

Figura 5.7 - Casos de uso do actor UConfiguradorSistemaAlvo

Criar ficheiro Guimap do sistema alvo

A criação do ficheiro GuiMap a conter todos os objectos gráficos da aplicação será bastante facilitada através da função *learn*, da ferramenta Winrunner, que permite o registo em ficheiro texto (ficheiro guimap) de todos os objectos gráficos, pertencentes a cada janela do sistema alvo. Todas

as propriedades utilizadas pelo Winrunner para o reconhecimento de cada objecto gráfico em tempo de execução são automaticamente registadas.

Antes da geração final do ficheiro *guimap*, o utilizador deverá modificar o identificador lógico dos objectos gráficos para identificadores sem espaços, isto é, constituídos de um conjunto de caracteres contínuos. Esta é uma restrição da *framework* WRAFS que não consegue captar o identificador do objecto gráfico do ficheiro texto de teste, caso o mesmo contenha espaços. Isto porque a *framework* considera que o identificador do objecto é a primeira palavra antes do primeiro espaço, o que leva a erros durante a execução dos testes. Por questões de legibilidade, antes da geração final do ficheiro *guimap,* o utilizador poderá alterar os identificadores lógicos dos objectos para nomes mais sugestivos que melhor representem o objecto gráfico.

Criar sistema alvo

O sistema AGEAT permite a criação e gestão testes para sistemas alvo distintos. O utilizador **UConfiguradorSistemaAlvo** deve responsável por introduzir as seguintes informações aquando da inclusão de um novo sistema alvo: nome, directoria onde encontra-se o ficheiro guimap, directorias onde devem ser geradas as tabelas de nível alto, médio e baixo da *framework* que correspondem aos ciclos, baterias e operações respectivamente.

Importar objectos gráficos

Durante a realização desta acção o actor deve indicar o ficheiro *guimap*, que contém os objectos gráficos do sistema, que deve ser carregado. Caso venham a existir outros ficheiros *guimap* com objectos complementares aos já carregados, o actor deve indicar que o carregamento deve ser feito de modo incremental, o que fará com que os objectos de objectos já carregados sejam mantidos.

O actor **UTestador** é responsável pela gestão das entidades conceptuais, gestão da geração e gestão da execução dos testes pretendidos (Figura 5.8).

Gestão das entidades conceptuais

Em relação a gestão das entidades conceptuais, poderá acrescentar e remover entidades conceptuais pertencentes ao sistema alvo. As entidades inseridas serão referenciadas na criação das operações o que posteriormente permitirá a extracção de relatórios que indicam as entidades conceptuais que estão a ser exercitadas pelos testes criados.

Gestão da geração dos testes

Os testes criados deverão seguir a hierarquia de construção adoptada pela *framework* WRAFS, sendo necessário a existência do conceito de ciclos, baterias e operações. A gestão da geração passa pela gestão das operações, baterias e ciclos, tradução de testes e consulta de hierarquia.

Gestão das operações de teste

No que respeita a gestão das operações de teste, o utilizador poderá criá-las, removê-las, alterá-las e personalizá-las.

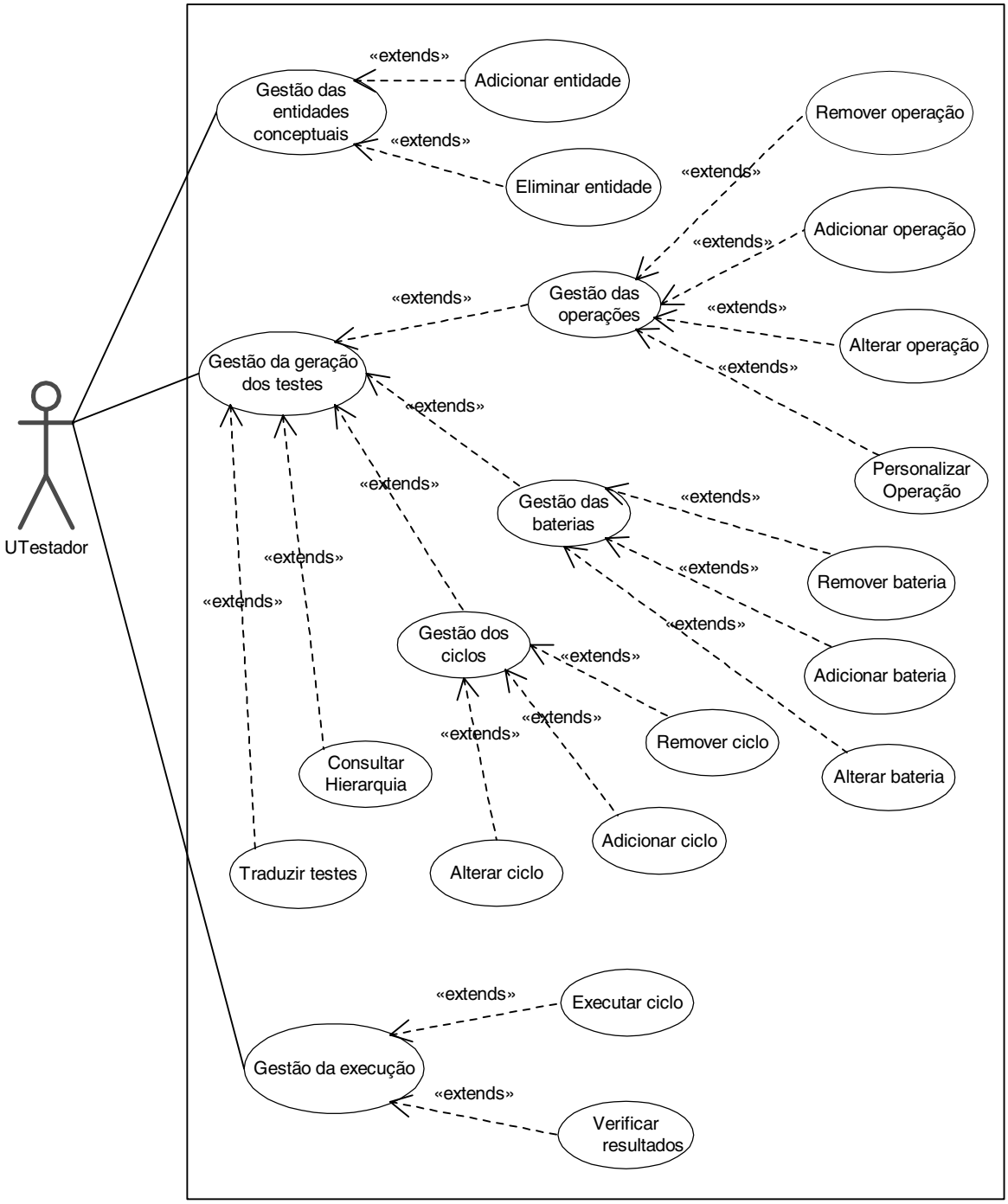

Figura 5.8 - Casos de uso utilizador UTestador

Adicionar operação

Durante a criação de uma operação, o utilizador deve informar uma ou mais entidades que serão exercitadas pela operação a ser criada e o nome do cenário a ser utilizado. Todas as janelas

pertencentes ao sistema alvo devem ser apresentadas e o utilizador deve, respeitando a ordem de navegação, seleccionar as janelas do sistema onde haverão interacções. Para cada janela seleccionada, deverão ser retornados os nomes dos objectos gráficos que a compõe. O utilizador deve seleccionar os nomes dos objectos sobre os quais haverão interacções, no âmbito da operação de teste que está a ser criada. Os objectos deverão ser seleccionados na ordem em que serão utilizados durante a realização da operação. Por fim, o utilizador deve gravar a operação.

Personalizar operação

O utilizador deve informar o nome da operação a ser personalizada. Será apresentada para cada interacção registada da operação, o nome da função WRAFS sugerida e utilizada na criação original da operação e a lista de funções WRAFS disponíveis para aquela interacção. O utilizador pode consultar a definição de cada função WRAFS disponível e fazer a alteração da interacção desejada, para que passe a utilizar outra função WRAFS que lhe pareça mais conveniente.

Remover operação

O utilizador deve informar o nome da operação e solicitar a sua remoção. A operação será removida, não sendo mais possível a sua utilização.

Alterar operação

Durante esta acção, o utilizador poderá adicionar ou remover entidades associadas à operação e alterar as interacções anteriormente gravadas. Poderá incluir ou remover janelas associadas à operação e incluir ou remover os objectos gráficos seleccionados em cada janela. O utilizador não poderá alterar o cenário associado inicialmente à operação.

Gestão das baterias de teste

No que respeita a gestão das baterias de testes, poderá criá-las, removê-las e alterá-las.

Adicionar bateria

Para a adição de uma nova bateria, todas as operações anteriormente criadas pelo sistema serão retornadas. O utilizador deverá seleccionar cada operação desejada, na ordem em que deverão ser invocadas na bateria. Para cada operação seleccionada o sistema deve retornar quais parâmetros devem ser preenchidos, sendo consultadas aquelas interacções da operação cujas funções WRAFS recebem parâmetros de entrada. Para uma melhor identificação da interacção, o sistema deve apresentar o nome da janela, o nome do objecto gráfico e o(s) nome(s) do(s) parâmetro(s) de entrada que a função para aquele objecto gráfico necessita. O utilizador poderá manualmente informar o valor destes parâmetros pela interface gráfica ou informar o nome do ficheiro de onde deverão ser carregados estes valores. Cada linha do ficheiro corresponderá a uma chamada da

função. Caso o ficheiro contenha várias linhas, várias invocações a mesma operação serão registadas para a bateria em questão. Por fim o utilizador deverá gravar a bateria de teste.

Remover bateria

O utilizador deve informar o nome da bateria e solicitar a sua remoção. A bateria será removida do sistema, não sendo mais possível a sua utilização.

Alterar bateria

Durante esta acção, o utilizador poderá adicionar ou remover operações associadas à bateria e alterar os valores dos parâmetros anteriormente gravados.

Gestão dos ciclos de teste

Em relação aos ciclos de teste, as operações disponíveis permitem a criação, remoção e alteração de ciclos.

Adicionar ciclo

Para a adição de um novo ciclo, todas as baterias anteriormente criadas pelo sistema serão retornadas. O utilizador deve seleccionar as baterias desejadas, na ordem em que serão invocadas a partir do ciclo. Por fim, deverá gravar o ciclo criado.

Remover ciclo

O utilizador deve informar o nome do cicloe solicitar a sua remoção. O ciclo será removido do sistema, não sendo mais possível a sua utilização.

Alterar ciclo

Durante esta acção, o utilizador poderá adicionar ou remover baterias associadas ao ciclo. Poderá também alterar a ordem de invocação de cada bateria pertencente ao ciclo.

Traduzir testes

O utilizador **UTestador** poderá efectuar a tradução dos testes criados para a interface de ficheiros texto entendida pela *framework* através da operação de tradução disponível. Para tal, deve seleccionar o nome do ciclo, bateria ou operação a ser traduzida e solicitar a sua tradução. Um ficheiro de alto, médio ou baixo nível definido pela *framework* será gerado nas directorias apropriadas.

Consultar hierarquia

O utilizador deve informar o nome do ciclo a ser consultado e toda a hierarquia de testes, em formato de árvore, será retornada para sua consulta.

Gestão de Execução

O utilizador poderá gerir a execução dos testes, estando disponíveis funcionalidades que permitem a escolha do ciclo de testes a ser executado e uma posterior verificação de resultados.

Executar ciclo

O utilizador deve seleccionar o nome do ciclo a ser executado. A ferramenta Winrunner é carregada, de forma minimizada, iniciando a execução da hierarquia de testes pertencente ao ciclo. O sistema alvo será carregado e posteriormente todas as acções distribuídas pelo ciclo, baterias e operações, pertencentes a hierarquia, serão automaticamente realizadas sobre o sistema alvo, sem intervenção do testador.

Verificar resultado

O utilizador deve informar o ciclo executado e o sistema deve retornar o resultado da realização de cada interacção realizada nos objectos gráficos do sistema. Caso a interacção não tenha sido realizada com sucesso deverá retornar, no princípio da descrição da acção, a palavra *Failed*. Caso tenha sido realizada com sucesso, deverá retornar a palavra *OK*.

5.6. Arquitectura de Componentes

5.6.1. Definições dos Módulos

A estrutura interna do sistema AGEAT é representada na Figura 5.9.

O motor de geração, executor, tradutor foram desenvolvidos utilizando-se a linguagem C# do ambiente Visual Studio .Net.

Motor de Geração - Este módulo implementa as funcionalidades descritas nos casos de uso **Gestão da geração do testes** (ver **Casos de Uso)** com base na informação gerida pelos casos de uso **Gestão das funções WRAFS** e **Criação de cenário** (ver **Casos de Uso).** Este módulo fornece ao módulo Tradutor um conjunto de operações, baterias e ciclos.

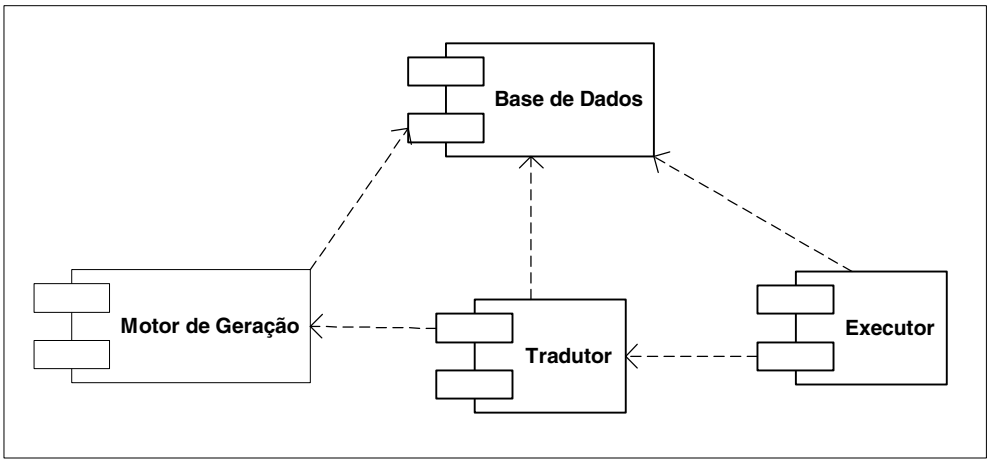

Figura 5.9 - Módulos que compõem o sistema AGEAT

Tradutor - Este módulo implementa as funcionalidades descritas no caso de uso **Traduzir testes** (ver **Casos de Uso).** É o módulo responsável por transformar a informação fornecida pelo Motor de Geração nas respectivas tabelas de teste reconhecidas pela *framework* WRAFS, tabelas de nível baixo, médio ou alto. Este módulo conhece o formato exigido pela *framework* WRAFS para a criação das tabelas de teste que serão fornecidas ao módulo Executor.

Executor - Este módulo implementa as funcionalidades descritas no caso de uso **Gestão da execução (**ver **Casos de Uso).** É o módulo que invoca a ferramenta Winrunner passando-lhe como parâmetro o *script* que iniciará a execução dos testes.

Base de Dados - A base de dados mantém os dados manipulados pelo sistema. A base de dados é gerida através do SGBD *Microsoft Access*.

Capítulo 6

Casos de Estudo: *Windows* e *Web*

Este capítulo apresenta dois Casos de Estudo que foram realizados com o objectivo de avaliar sob cenários reais, a aplicabilidade do sistema AGEAT e respectivos ganhos de produtividade no âmbito da engenharia de testes.

A Secção 6.1 apresenta o sistema *web* Fénix e as actividades realizadas durante a geração e execução automática dos testes. A Secção 6.2 apresenta o sistema *windows L'Avocat* e as acções realizadas durante o processo de testes que foi definido para o caso de estudo. Por fim, a Secção 6.3 compara a abordagem de testes utilizada para o sistema *Windows* e para o sistema *Web*.

6.1. Sistema Fénix

Nesta secção apresenta-se a aplicação do sistema AGEAT para criação e execução automática de testes ao sistema Fénix [FenixIST]. O objectivo desta Secção é apresentar a utilização do sistema AGEAT na construção de testes para um sistema *Web*.

O sistema Fénix consiste num sistema de informação *Web* que suporta a gestão de vários processos de negócio do Instituto Superior Técnico e dos seus diferentes actores (e.g. alunos, professores, coordenadores, entre outros). O sistema Fénix permite a gestão de diferentes perfis de utilizadores com acesso a operações específicas. Alguns tipos de utilizadores existentes são: alunos, docentes, operadores e gestores pedagógicos. Neste caso de estudo, escolhemos automatizar os testes de algumas das operações disponíveis ao utilizador com perfil "Docente". A Figura 6.1 apresenta um diagrama de actividades possíveis para um utilizador de perfil Docente.

6.1.1. Actividades de Preparação e Análise

Apresenta-se nesta secção as actividades de preparação e análise necessárias à construção dos testes.

Identificação das Condições de Teste

A escolha dos testes a serem automatizados neste caso foi baseada nos seguintes requisitos: (1) deverão ser apresentados testes que realizem todas as operações CRUD; (2) deverão ser apresentados testes realizados sobre janelas da interface que possuam os mais variados tipos de componentes gráficos.

Utilizando os requisitos acima referidos, seleccionamos os seguintes módulos do sistema: (1) Ficha de Docente – Qualificações Académicas; (2) Ficha Docente – Carreira na Docência e (3) Administração de Publicações. Para cada módulo iremos automatizar a realização das operações de: (1) consulta; (2) criação; (3) alteração; (4) remoção.

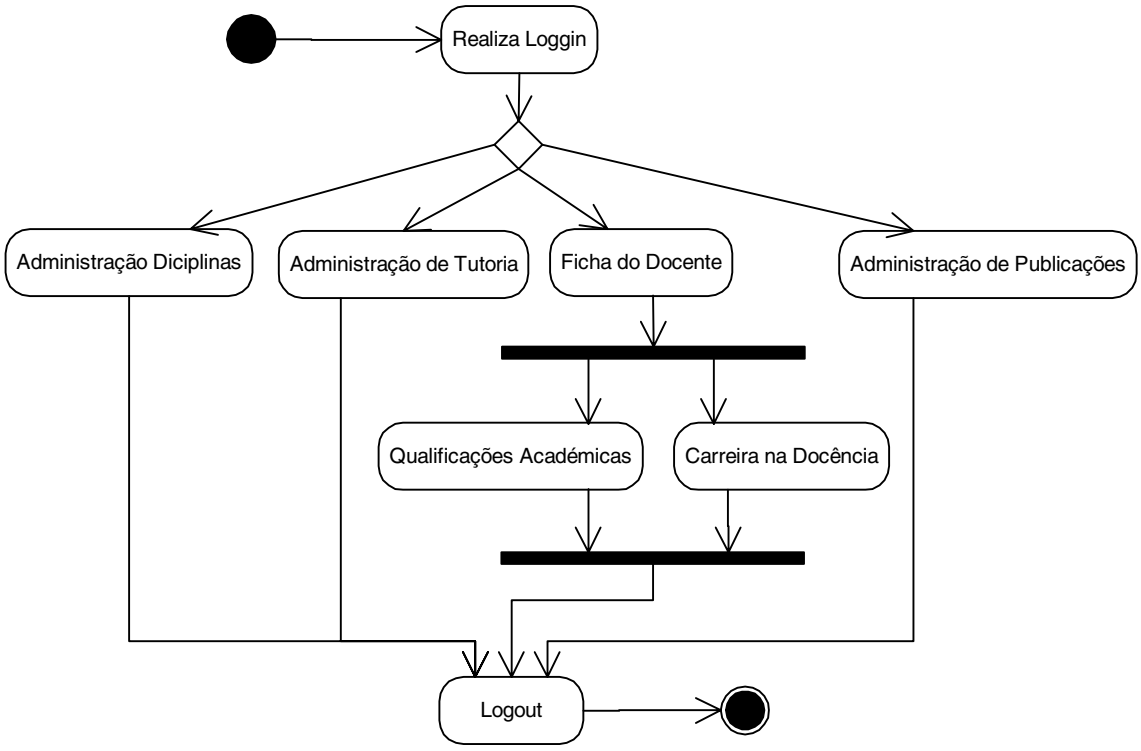

Figura 6.1 - Diagrama de actividades possíveis para o módulo "Docência" no Fénix

Construção e Importação de GuiMap

A primeira operação a ser feita no AGEAT é a criação do sistema alvo e inclusão da informação de configuração necessária à construção dos testes.

A construção do ficheiro GuiMap é a única operação que deve ser realizada directamente na interface da ferramenta WinRunner. Ao utilizarmos a função *Learn*, para reconhecimento de cada objecto gráfico, constatamos que todas as operações destinadas ao módulo "Docência" no Fénix são realizadas num mesmo objecto do tipo *Windows* que apresenta os objectos gráficos internos apropriados, consoante a operação seleccionada pelo utilizador. Esta constatação levou-nos a decisão de renomear os identificadores lógicos dos objectos gráficos internos, segundo a seguinte regra: **<Nome do sub-módulo a partir do módulo Docência>_<Nome do campo na interface Fénix>.**

Desta forma, o utilizador poderá mais facilmente seleccionar os objectos gráficos durante a construção das operações de teste no AGEAT, porque saberá o contexto ao qual pertence cada objecto gráfico.

Criado o ficheiro GuiMap, designado de Fenix.gui, passamos à utilização directa do sistema AGEAT. A Figura 6.2 apresenta parte do resultado da importação do ficheiro, apresentado na interface do AGEAT. O Anexo A.1 apresenta o conteúdo integral do ficheiro guimap, importado pelo sistema AGEAT.

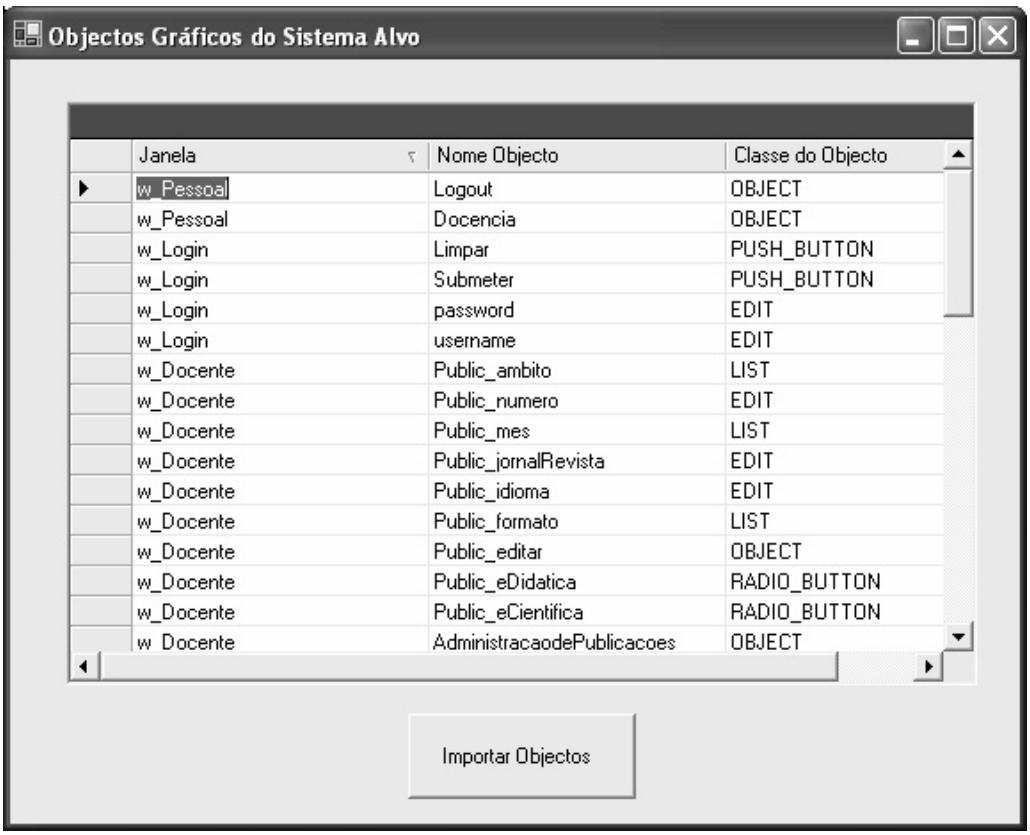

Figura 6.2 - Objectos gráficos do sistema Fénix importados pelo AGEAT

6.1.2. Construção e Execução dos Testes

Operações de Teste

Numa única janela é possível criar toda a informação necessária associada à operação de testes, composta por um número potencialmente infinito de interacções.

Destacamos na Figura 6.3 a informação seleccionada para a construção da operação de teste **inserir_publicacao**. O cenário utilizado foi o "**Genérico"** e a entidade de negócio exercitada é "**Publicacoes**". Na caixa denominada "Janelas Utilizadas Durante a Operação" foi introduzida a janela "**w_Docente"**, onde a operação **inserir_publicacao** é realizada, mas outras janelas poderiam ter sido introduzidas e o sistema respeitaria a ordem indicada nesta caixa.

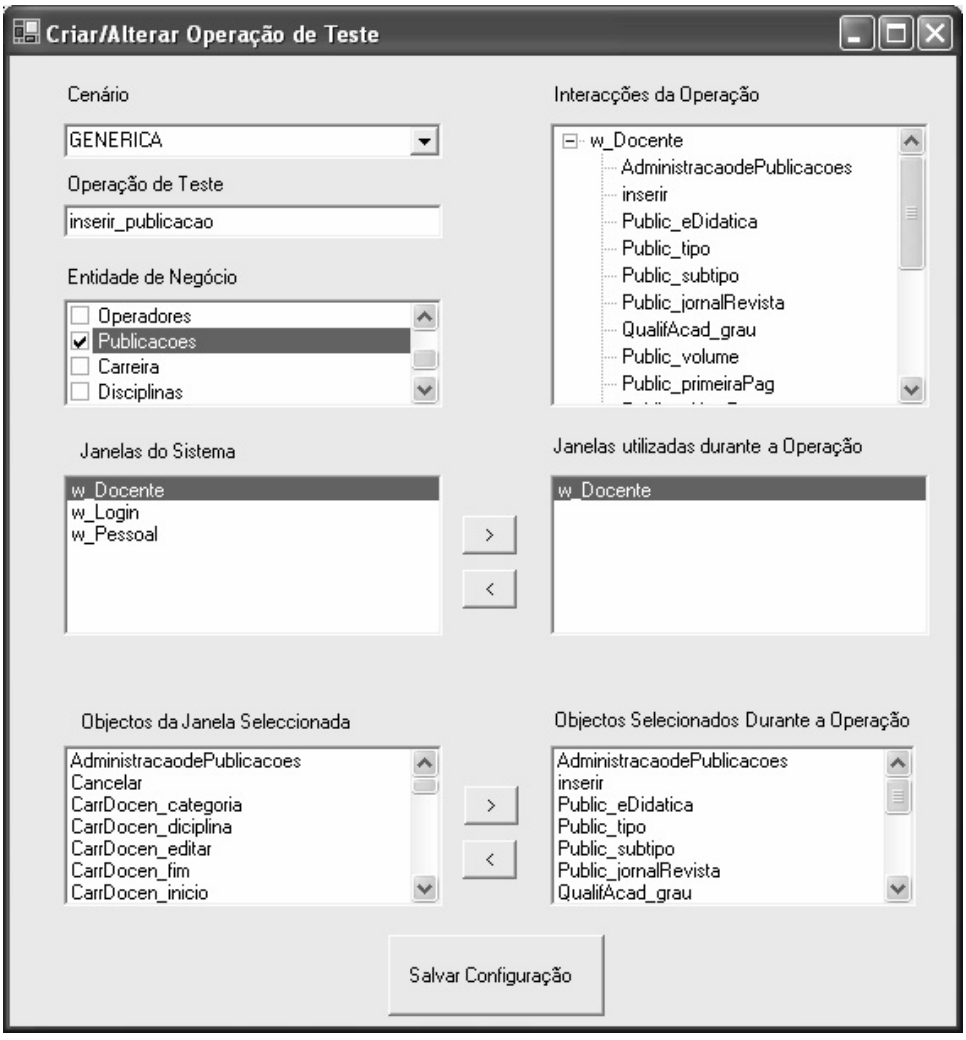

Figura 6.3 - Operação inserir_publicacao criada através do AGEAT

Os objectos apresentados na caixa denominada "Objectos Seleccionados Durante a Operação" representam os objectos gráficos que sofrerão interacções respeitando-se a ordem em que são visualizados na caixa. O tipo de cada objecto gráfico seleccionado não é apresentado ao utilizador

nesta janela. O testador ao construir as operações deve apenas indicar (baseado nos nomes dos objectos) quais deles serão utilizados durante a operação e em que ordem. Os nomes dos objectos apresentados equivalem aos nomes dos identificadores lógicos criados na construção do ficheiro *GuiMap*, daí a importância já ressaltada da utilização de nomes sugestivos nesta altura.

No decorrer deste caso de estudo utilizamos o sistema AGEAT para a construção das seguintes operações de teste identificadas: 1. inserir_publicacao; 2. alterar_publicacao; 3. remover_publicacao; 4. consultar_publicacoes; 5. inserir_qualificacao_academica; 6. alterar_qualificacao_academica; 7. remover_qualificacao_academica; 8. consultar_qualificacoes; 9. inserir_carreira_docente; 10. alterar_carreira_docente; 11. remover_carreira_docente; 12.consultar_carreira_docente; 13. efectua_loggin, 14. efectua_logout. Criamos um total de 14 operações que serão combinadas para a criação de diversas baterias de teste.

Procuramos, sempre que possível, criar operações independentes entre si, isto é, a execução de cada operação é o menos possível dependente da execução de outras operações, anteriores ou posteriores. Desta forma, teremos mais flexibilidade durante a criação das baterias.

Baterias de Teste

O sistema AGEAT, baseado na estrutura hierárquica sugerida pela *framework* WRAFS, faz a separação entre as operações de teste criadas e os seus dados de entrada, através do conceito de baterias de teste. Desta forma, torna-se possível a criação de testes *Data-driven* [FewsterGrahm1999]. Numa única janela do sistema AGEAT iremos criar algumas baterias de teste, informando para isto, as operações que a compõem e seus respectivos dados de entrada.

Destacamos na Figura 6.4 a informação seleccionada para a construção da bateria de teste **exercita_publicacoes**.

A caixa denominada “Operações Instanciadas” contém todas as operações criadas através do sistema. A caixa denominada “Operações Seleccionadas” contém as operações seleccionadas para a bateria **exercita_publicacoes** que estamos a criar.

Para cada operação de teste associada à bateria, o sistema informou os seus parâmetros de entrada, para que pudéssemos introduzir os respectivos valores de atribuição, na tabela denominada “Parâmetros Operação Seleccionada”. Caso se pretendesse uma introdução massiva de valores de parâmetros, estes poderiam ser importados a partir de ficheiro. A composição da bateria **exercita_publicacoes** é visualizada através da caixa denominada “Configuração Actual da bateria”, visualizada no canto inferior direito desta janela. Na Figura 6.4 não é possível apresentar toda a composição da bateria, mas o Anexo A.2 contém a informação completa de todas as baterias de testes criadas durante este caso de estudo, nomeadamente: 1. exercita_carreira; 2. exercita_qualificacoes; 3. exercita_publicacoes; 4. loggin, 5. logout

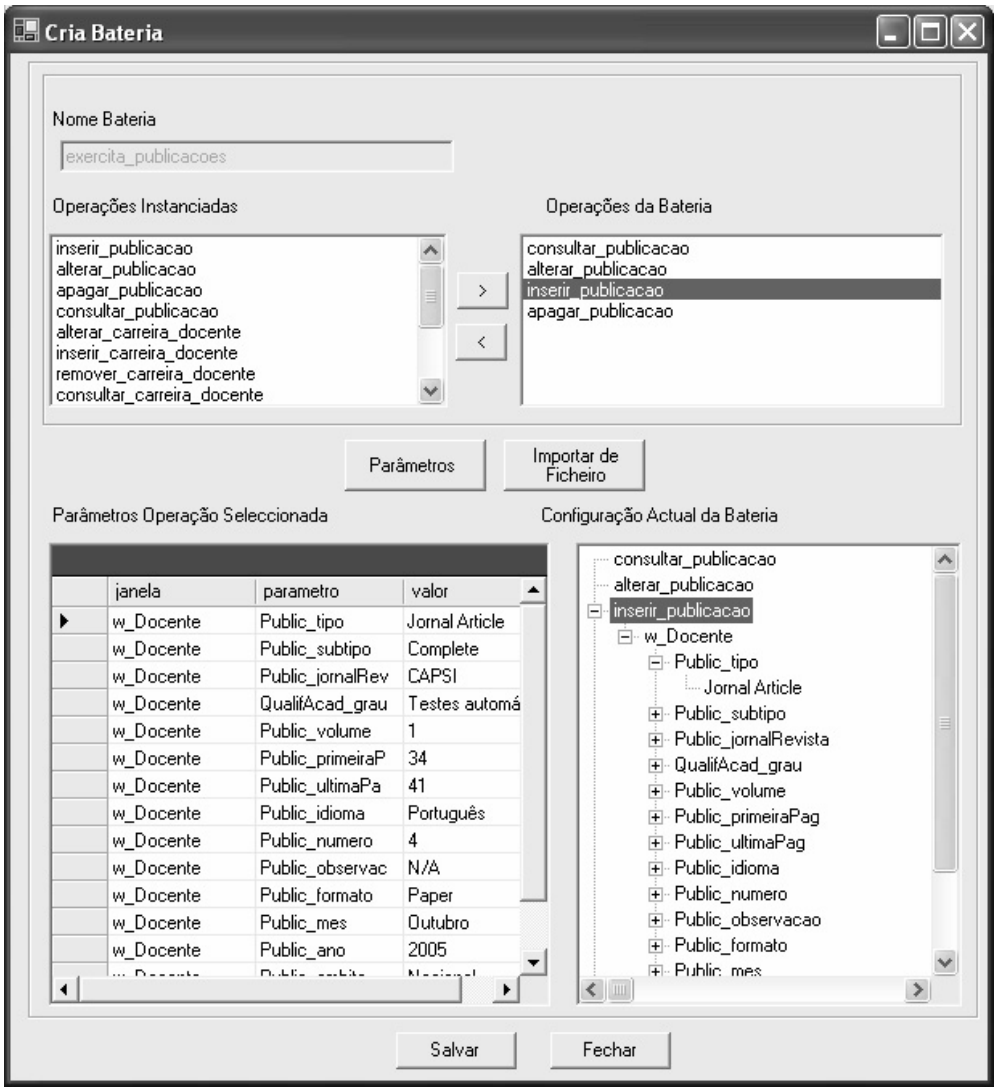

Figura 6.4 - Criação da bateria "exercita_publicacoes"

Finalizamos esta secção a apresentar um diagrama de objectos que ilustra os objectos e relações relacionados à operação de teste **efectua_login** e a bateria de testes **loggin** (Figura 6.5). Escolhemos a operação **efectua_login** por envolver um número pequeno de objectos gráficos, de modo a facilitar a legibilidade do diagrama.

Ciclos de Teste

Neste caso de estudo criamos um único ciclo de teste, chamado "**docencia"** que referencia as baterias: **loggin, exercita_publicacoes, exercita_qualificacoes, exercita_carreira, logout,** nesta ordem. A Figura 6.6 apresenta a informação necessária para a criação do ciclo "**docencia".**

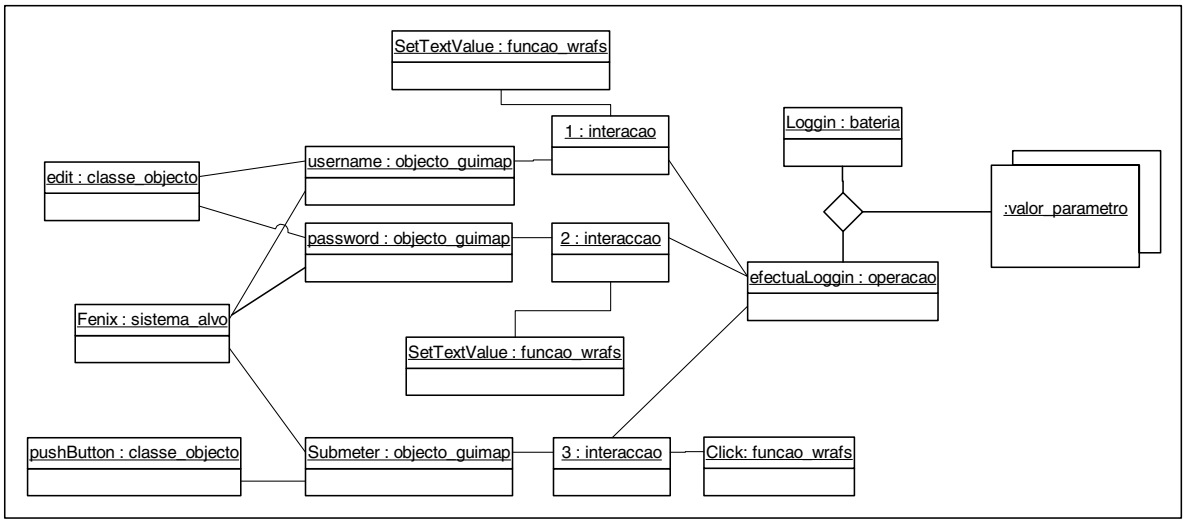

Figura 6.5 - Diagrama de objectos que ilustra a operação efectua_login

Tradução de Testes

Antes da execução dos testes criados realizamos a tradução de todas as operações, baterias e o ciclo **"docencia"**. A Figura 6.7 mostra o resultado da tradução para a operação de teste **alterar_carreira_docente**: o conteúdo do ficheiro alterar_carreira_docente.SDD.

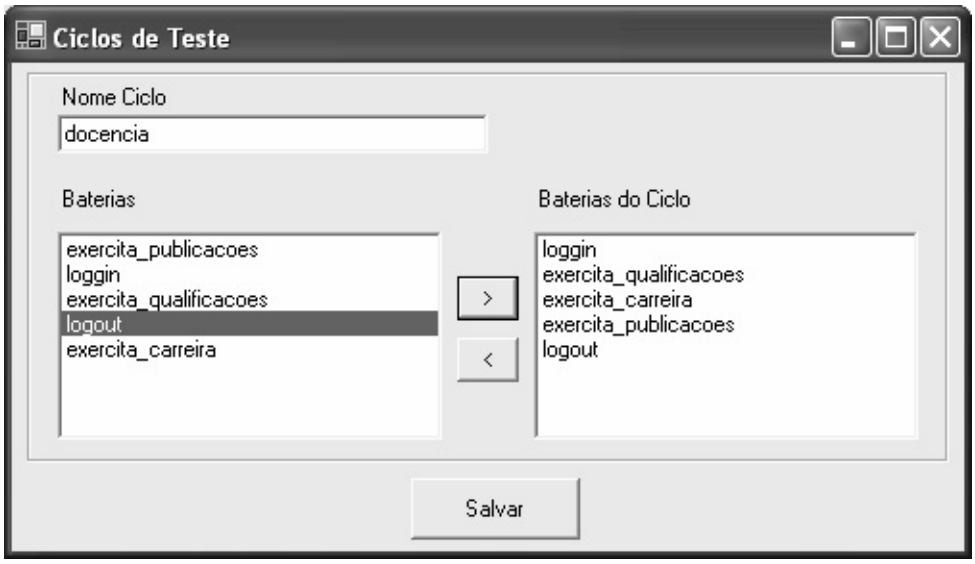

Figura 6.6 - Criação do ciclo "docencia" através do AGEAT

Execução e Verificação dos Resultados

Uma vez criada toda a informação necessária, seleccionamos o ciclo que queremos executar e passamos a visualizar a realização automática de todos os testes criados (Figura 6.8). Todos os testes foram realizados e nenhuma interacção manual foi necessária.

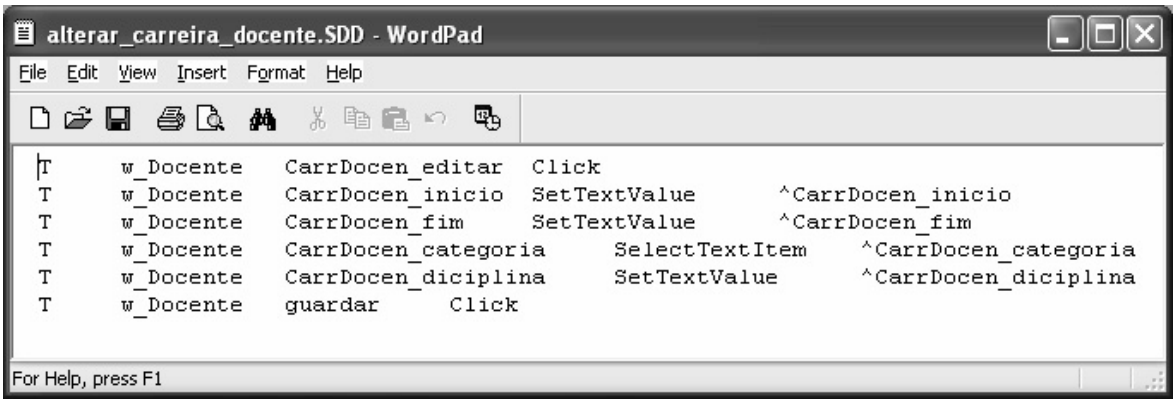

Figura 6.7 - Conteúdo do ficheiro correspondente à operação de teste alterar_carreira_docente

Uma listagem completa do resultado obtido encontra-se no Anexo A.3. O resultado dos testes é apresentado como uma lista de acções realizadas e uma indicação se foi executada com sucesso ou não. Como constata-se, todas as acções foram realizadas com sucesso, indicado pela sigla "OK", no princípio de cada linha do relatório. Este relatório é criado pela *framework* WRAFS durante a execução dos testes e é apresentado pelo AGEAT ao testador. Caso determinada acção não tivesse sido realizada com sucesso, a linha correspondente seria iniciada pela palavra "Failed".

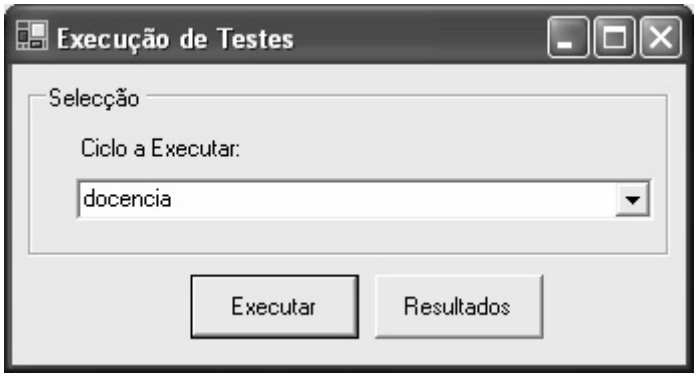

Figura 6.8 - Janela para execução e subsequente verificação de resultados dos testes

6.1.3. Evolução do Sistema Alvo

Na secção anterior mostramos a utilização do sistema AGEAT na construção e execução de testes ao sistema Fénix. O ciclo de teste criado, chamado "**docencia**", exercita algumas funcionalidades do sistema que poderão ser exercitadas quantas vezes forem necessárias, sem qualquer esforço adicional.

Imaginemos agora que uma nova *release* do sistema Fénix tivesse sido desenvolvida para inclusão de novas funcionalidades ao módulo Docência, sem alteração relativamente às operações pertencentes ao Ciclo "**docencia**" já criado. O esforço de testes a esta nova *release* estaria resumido a construção das novas operações e baterias de teste, mas os testes criados neste caso de estudo, permaneceriam válidos. A execução repetida aos testes estariam a garantir que os novos desenvolvimentos não influenciaram o bom funcionamento das operações anteriormente já validadas.

Uma vez que as baterias criadas neste trabalho poderão ser executadas repetidas vezes, em testes de regressão, procuramos configurá-las de forma que, ao final da execução, os dados do sistema Fénix tenham-se mantidos iguais. Para tal, por exemplo, após uma operação de inserção, a operação de remoção que se segue, remove a informação anteriormente inserida.

Caso as alterações introduzidas numa nova *release* do sistema Fénix incluíssem alterações às operações validadas pelos testes do ciclo "**docencia**", as operações e baterias existentes precisariam ser actualizadas, bem como a configuração dos objectos gráficos existentes. O sistema AGEAT permite a importação incremental de novos objectos e a alteração das operações e baterias de teste, para adaptação dos testes às mudanças no sistema alvo.

6.2. Sistema L'avocat

Nesta Secção iremos apresentar a aplicabilidade do sistema AGEAT para criação e execução automática dos testes ao sistema *L'Avocat.*

O sistema *L'Avocat* é um sistema de informação *windows* que realiza a gestão de alguns processos de negócio dum escritório de advocacia. O sistema possui uma interface comum a maioria das aplicações *Windows*, composta por um número variado de objectos gráficos que permitiram uma ampla validação das características e potencialidades do sistema AGEAT.

No *L'Avocat*, as janelas acedidas a partir do item de menu "**Ficheiro"** ou correspondentes botões da barra de ferramentas, manipulam as principais entidades de informação do sistema, nomeadamente: Clientes, Processos e Órgãos Jurídicos. Operações auxiliares como: agendamento de actividades, controlo de créditos e débitos, administração do sistema e relatórios são acedidos a partir de outros itens de menus existentes. Durante este caso de estudo iremos focalizar os testes no exercício das operações relacionadas às entidades **Clientes** e **Processos**.

6.2.1. Actividades de Preparação e Análise

Apresenta-se nesta secção as actividades de preparação e análise necessárias à construção dos testes.

Identificação das Condições de Teste

Identifica-se nesta Secção, as condições de teste que serão verificadas através dos testes sobre o sistema *L'Avocat*. Pretende-se verificar o comportamento do sistema em variadas condições, impostas por específicos dados de entrada e restrições de navegação (Tabela 6.1).

#	**Condição de Teste**	**Objectivo**
1	InserePrimeiroCliente	Inserir um cliente quando não há nenhum em base de dados
2	InsereClienteSemNome	Inserir um cliente sem o campo obrigatório: nome do cliente
3	InserirClientePessoaFísica	Inserir cliente do tipo pessoa física
4	InserirClientePessoaJuridica	Inserir cliente do tipo pessoa jurídica
5	AlterarDadosAdicionaisCliente	Alterar dados adicionais do cliente inserido no teste #2
6	IncluiTelefoneResidencial	Incluir telefone residencial ao cliente inserido no teste #3
7	IncluiTelefoneComercialSemNumero	Incluir telefone comercial sem o informar o campo obrigatório: número do telefone
8	ConsultaProcessosCliente	Consultar processos em que o cliente inserido no teste #3, participa como Autor ou Réu
9	ArquivaCliente	Arquivar (desactivar) cliente inserido no teste #3
10	RestauraCliente	Restaurar (reactivar) cliente arquivado no teste #8
11	InsereProcessoComValorInvalido	Inserir Processo com valor da causa inválido, isto é, diferente de numeral
12	InsereProcessoAccaoTrabalhista	Inserir processo Trabalhista da 4A Vara
13	InsereProcessoRepetidamente	Inserir processo com número já inserido no teste #11
14	CriaAndamentoProcesso	Criar andamento do processo inserido no teste #11
15	AlteraAndamentoProcesso	Alterar andamento inserido no teste #11
16	CriaAndamentoProcesso	Criar novo andamento do processo inserido no teste #11
17	ExcluiAndamentoProcesso	Excluir andamento inserido no teste #14
18	RemoveCliente	Remover cliente inserido no teste #3
19	RemoveCliente	Remover cliente inserido no teste #4
20	RemoveUltimoCliente	Remover último cliente da base de dados

Tabela 6.1 - Condições de teste a serem verificadas pelos testes automáticos criados para o caso de estudo do sistema L'Avocat

Construção e Importação de GuiMap

Através da função *Learn*, disponível pela ferramenta Winrunner, realizou-se o reconhecimento de todos os objectos gráficos do sistema *L'Avocat*, envolvidos nas operações que serão realizadas nas entidades: Cliente e Processo.

Durante esta actividade, verificou-se que a janela principal para manipulação de dados de Clientes e a janela principal para manipulação de dados de Processos, utilizam o mesmo tipo de objecto gráfico para agrupar informações relacionadas, nomeadamente o objecto *Tab*. Através deste tipo de objectos é possível ter-se diferentes vistas de informação, acessíveis a partir de uma mesma janela. A Figura 6.9 ilustra a janela de Processos do sistema *L'Avocat* e o objecto *Tab* que permite a visualização de duas vistas disponíveis, nomeadamente **Dados Gerais e Andamento.**

Com o objectivo de termos nomes de objectos mais expressivos, que facilitem a tarefa de criação dos testes e, baseado nesta característica da interface gráfica do sistema alvo, decidimos utilizar a seguinte nomenclatura para os identificadores de objectos internos à janela de Clientes e de Processos: o nome de cada objecto gráfico será prefixado de um nome abreviado que represente cada vista permitida pelo objecto ***Tab***. Por exemplo, o objecto gráfico que apresenta a **morada** do cliente é apresentado na vista **Dados Principais**. O identificador deste objecto será portanto alterado para **Prin_Morada**, onde o prefixo **Prin** remete ao nome da vista a que ele pertence e **Morada** remete ao tipo de informação que ele armazena.

O objecto *Tab* disponível na janela de Clientes, permite uma navegação que acede as seguintes vistas: Dados Principais, Telefones/Fax, Processos e Dados Adicionais. Os nomes dos objectos gráficos pertencentes à vista **Dados Principais** foram precedidos da abreviação **Princ**. Os nomes dos objectos gráficos pertencentes à vista **Telefones/Fax** foram precedidos da abreviação **Tel**. Os nomes dos objectos gráficos pertencentes à vista **Processos** foram precedidos da abreviação **Proc** e finalmente, os nomes dos objectos gráficos pertencentes à vista **Dados Adicionais** foram precedidos da abreviação **Adic**.

Na janela de Processos, os nomes dos objectos gráficos pertencentes à vista **Dados Gerais** foram precedidos do nome abreviado **Ger** e os nomes dos objectos gráficos pertencentes à vista **Andamento** foram precedidos do nome abreviado **And**.

Criado o ficheiro GuiMap, o qual chamamos de *Lavocat.gui*, passamos à utilização directa do sistema AGEAT. O Anexo B.1 apresenta o conteúdo integral do ficheiro guimap, que foi importado pelo sistema AGEAT.

Figura 6.9- Janela de Processos do sistema L'Avocat

6.2.2. Construção e Execução dos Casos de Teste

Uma vez que temos identificadas as condições de teste, passamos à fase de construção dos testes, através do sistema AGEAT. Para isto, iremos transformar a informação da Tabela 6.1 em operações, baterias e ciclos de testes. Para a realização destas actividades o testador precisa conhecer como é feita a navegação sobre o sistema alvo e qual seu comportamento perante as variadas possibilidades de interacção.

Por exemplo, para a criação da operação de teste que exercite a condição #2 da Tabela 6.1, o testador deve saber que, ao tentarmos criar no *L'Avocat* um cliente sem informar o seu nome, o sistema retorna uma mensagem de alerta a dizer que o nome do cliente é um campo obrigatório. Portanto, ao criar esta operação de teste o testador deve informar as interacções na janela principal de cliente e na janela que contém a mensagem de alerta, que será visualizada após pressionarmos no botão que regista as alterações feitas ao cliente. Se após a realização automática desta operação, todas as acções tenham sido realizadas com sucesso, teremos o indicativo que o teste foi realizado com sucesso. Imaginando-se que, após pressionarmos no botão Salvar, a janela com a mensagem de alerta não era visualizada, a acção planeada para esta janela não poderia ser executada e no resultado da execução do teste teríamos a indicação de erro nesta acção. O erro apresentado nesta acção seria suficiente para determinarmos que o teste falhou.

Por outro lado, é claro que existirão diferentes níveis de exigência para verificação dos resultados. Vejamos o exemplo de um teste à condição #3 da Tabela 6.1. Através da interface do *L'Avocat* ao informarmos os dados do cliente e pressionarmos sobre o botão **Salvar**, caso os dados sejam coerentes e suficientes, o cliente é inserido em base de dados e nenhuma mensagem de confirmação é visualizada. Se pensarmos que a correcta realização destas acções é suficiente para a verificação do resultado do teste, a operação de teste terá como última acção a acção correspondente ao pressionar sobre o botão **Salvar**. Se, por outro lado, pensarmos que precisamos de uma confirmação mais segura que de facto o cliente foi inserido em base de dados, a esta operação de teste devem ser acrescidas de seguida acções que façam a consulta do cliente supostamente inserido. Desta forma a operação de inserção do cliente era composta de acções para a inserção e consulta do cliente inserido. Caso todas as acções que compõem a operação fossem realizadas com sucesso, o resultado do teste era positivo.

Em suma, as operações devem conter todas as acções necessárias para a execução e verificação do resultado do teste. Dado que os resultados dos testes indicam se cada acção foi executada com sucesso ou não, as acções de teste devem ser criadas na sequência duma execução com sucesso da operação de teste.

Identificação de um novo *Cenário* para o AGEAT

Antes mesmo de iniciar a construção das operações de teste, verificamos que o objecto *Tab*, existente nas janelas de Clientes e de Processos, é reconhecido pelo Winrunner com o tipo genérico "***Object***". Sendo que, para os objectos gráficos reconhecidos como ***object***, sabe-se que a função sugerida pelo sistema AGEAT, pertencente ao único cenário existente em base de dados, nomeadamente o cenário **Genérico**, será a função ***click***, sem parâmetros de entrada. No entanto, a função apropriada para a interface do sistema *L'Avocat* sobre o objecto *Tab* seria a função *click*, mas com a passagem de um parâmetro de entrada que corresponde ao nome da vista pretendida. Embora a função tenha o mesmo nome, correspondem a funções diferentes para o sistema AGEAT.

Se durante a criação dos testes utilizarmos o cenário Genérico, a acção que será sugerida pelo AGEAT para o objecto Tab será acção ***click*** sem passagem de parâmetro, que não permite a navegação entre as diferentes visões, portanto não adequada. O sistema AGEAT fornece duas opções para a solução da questão.

A primeira passa pela criação de um novo **Cenário** de teste, composto pela mesma informação do cenário **Genérico**, a excepção da acção associada ao objecto gráfico do tipo "***object*"**, que passaria a ser a função ***click*****,** com o recebimento de um parâmetro de entrada. Este novo cenário de teste, passaria a ser utilizado na criação de todas as operações de teste para o sistema *L'Avocat.*

A segunda opção de solução passa pela utilização do cenário **Genérico** existente para a criação de todas as operações de teste. No entanto, para aquelas operações que envolvessem interacções sobre

o objecto Tab, seria necessário a utilização da funcionalidade de **Personalização** disponível. Através desta funcionalidade seria possível alterar-se a função sugerida pelo cenário **Genérico**, pela função **click** com passagem de parâmetro, mais adequada.

Uma vez que a maioria das operações de teste do *L'Avocat* iriam precisar ser "personalizadas", utilizamos a primeira solução e criamos um novo cenário ao qual chamamos **CasodeEstudo.** Utilizando este cenário, todas as operações de teste serão geradas automaticamente sem necessidade por adaptações posteriores. A criação de um cenário é normalmente feita por um utilizador com perfil de administrador do sistema.

Operações de Teste

Para cada **condição de teste** identificada na Tabela 6.1 foi criada uma **operação de teste** correspondente sobre o sistema AGEAT. Acrescida a operação que realiza a entrada (*Loggin*) no sistema, criamos um total de 21 **operações de teste**.

As operações para o sistema *L'Avocat* foram criadas através da mesma janela utilizada para a criação das operações do sistema Fénix, apresentada na Secção 6.1.2 (Figura 6.3). As mesmas práticas foram utilizadas em ambos casos de estudo e a única diferença que ressaltamos foi a utilização do novo cenário – **CasodeEstudo-** que é informado na criação de cada **operação de teste.**

Baterias de Teste

Através da mesma janela utilizada para criação das baterias do sistema Fénix (Figura 6.4), criamos as seguintes **baterias de teste**: (1) efectua_loggin, (2) exercita_clientes, (3) exercita_processos, (4) refaz_estado_inicial.

Neste caso de estudo criamos baterias específicas para o exercício de cada entidade de negócio e por isto não temos exemplos de operações que pertençam a mais de uma bateria. Além disto, como só temos uma único ciclo de teste não temos exemplos de baterias que pertençam a mais de um ciclo de teste. No entanto, ambas situações são possíveis de serem construídas através do AGEAT. Poderá ser bastante aplicável num cenário de testes mais complexo, a existência de operações de teste que sirvam apenas de suporte à criação de operações mais complexas, pertencentes a diferentes cenários.

A bateria **refaz_estado_inicial** foi criada com o intuito de trazer a base de dados do sistema *L'Avocat* para um estado igual àquele inicialmente existente, antes da execução dos testes. Com esta estratégia os testes criados poderão ser re-executados as vezes que forem necessárias sem alterações nas acções realizadas e sem alterações nos dados de entrada. Sem a execução da bateria **refaz_estado_inicial**, à próxima execução os testes seriam interrompidos por mensagens de erro enviadas pelo sistema *L'Avocat* que diriam que os registos já existiam em base de dados, para o

caso das operações de inserção de dados. A construção de **operações de teste** que fizessem o tratamento de tais mensagens seria possível, mas exigiria maior esforço durante a fase de construção, constituindo o oposto do pretendido para este trabalho, que é de manter simples a tarefa de construção dos testes automáticos.

A composição das baterias e operações de teste criadas através do sistema AGEAT para este caso de estudo é apresentada no Anexo B.2.

Ciclos de Teste

Através da mesma janela utilizada para criação das baterias do sistema Fénix (Figura 6.6), criamos um único **ciclo de teste**, a quem chamamos de "**regressao",** dado que é composto por testes de regressão. Este ciclo é composto pelas seguintes baterias de teste: (1) efectua_loggin, (2) exercita_clientes, (3) exercita_processos, (4) refaz_estado_inicial.

Tradução dos Testes

Antes da execução dos testes criados realizamos a tradução de todas as operações, baterias e o único ciclo **"regressao"**. A Figura 6.10 mostra o resultado da tradução para a operação de teste **InsereProcessoAccaoTrabalhista**, o conteúdo do ficheiro InsereProcessoAccaoTrabalhista.SDD.

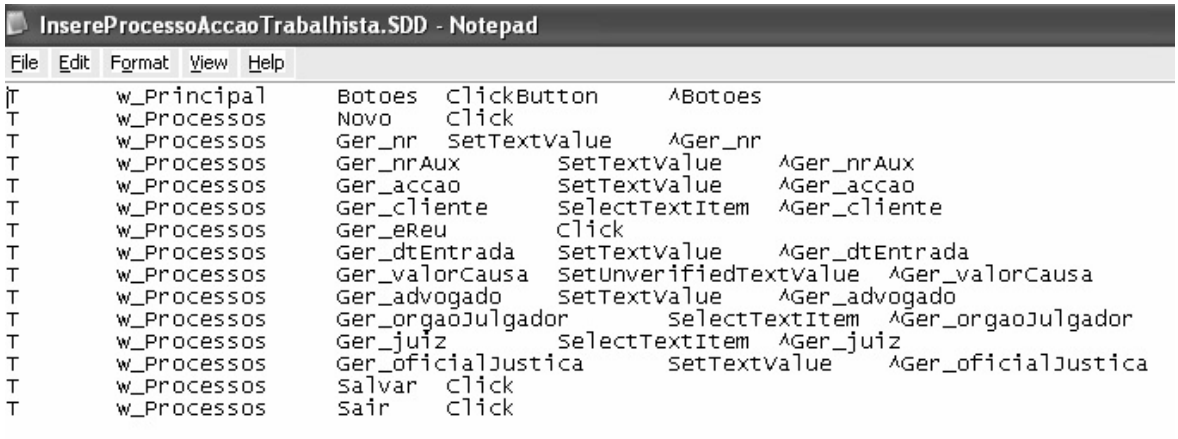

Figura 6.10 - Conteúdo do ficheiro correspondente à operação de teste InsereProcessoAccaoTrabalhista

Execução e Verificação dos Resultados

Após uma primeira execução do ciclo "**regressao**", verificamos algumas mensagens de erro nos resultados dos testes apresentados pelo AGEAT. Todos os erros encontrados foram verificados durante acções sobre os objectos gráficos do tipo ***edit***, com a particularidade de serem objectos com alguma máscara de formatação.

Na janela de **Processos** verificamos erro decorrente de acção no objecto que representa o valor da causa do processo (**Ger_valorCausa**) e o objecto que representa a data de entrada do processo no escritório de advocacia (**Ger_dtEntrada**) e na janela de **Clientes** no objecto que representa a data de nascimento do cliente (**Adic_nascimento**). Os erros registados durante as acções sob os estes objectos estão apresentados na Tabela 6.2.

FAILEDEditBox SetTextValue in w_Clientes verification failed. See table AlterarDadosAdicionaisCliente.SDD at line 6. w_Clientes:Adic_nascimento Set to:23041950, but returns: / / .
SetTextValue 15000 sent to Ger_valorCausa. **FAILED**EditBox SetTextValue in w_Processos verification failed. See table InsereProcessoValorValido.SDD at line 1. w_Processos:Ger_valorCausa Set to:15000, but returns:15000,00.
FAILEDEditBox SetTextValue in w_Processos verification failed. See table InsereProcessoAccaoTrabalhista.SDD at line 8. w_Processos:Ger_dtEntrada Set to:10052005, but returns: / / .
SetTextValue 55000 sent to Ger_valorCausa. **FAILED**EditBox SetTextValue in w_Processos verification failed. See table InsereProcessoAccaoTrabalhista.SDD at line 9. w_Processos:Ger_valorCausa Set to:55000, but returns:55000,00.

Tabela 6.2 - Erros retornados após primeira execução

Em relação aos erros ocorridos nos campos que recebem valores representativos de datas, nomeadamente **Ger_dtEntrada** e **Adic_nascimento,** foi verificado que representam de facto defeitos no sistema *L'Avocat.* Durante uma execução manual do teste, comprovou-se que o sistema de facto não está a registar qualquer valor inserido nestes campos.

Em relação aos erros ocorridos no campo **Ger_valorCausa** verificamos que estão relacionados com a máscara que introduz vírgulas e/ou pontos no valor introduzido pelo utilizador, dado que representa um valor monetário. Durante a execução do teste a função utilizada neste objecto (*SetTextValue*), altera o valor da propriedade *text* do objecto **Ger_valorCausa** para o valor enviado como parâmetro da operação de teste e posteriormente, verifica se o valor actual da propriedade *text* está igual ao valor que foi enviado. Neste caso, como o objecto possui a máscara, o valor da propriedade *text* foi modificada e por isto houve o registo do erro.

No entanto, como estas mensagens não representam erros no *L'Avocat*, utilizamos a funcionalidade de **personalização** de modo a substituirmos, para estes caso, a função sugerida pelo AGEAT – *SetTextValue* - pela função semelhante - *SetUnverifiedText* - que não realiza a verificação após a modificação do valor da propriedade *text.* As operações personalizadas foram as seguintes: **InsereProcessoAccaoTrabalhista** e **InsereProcessoValorValido**. A Figura 6.11 apresenta a janela onde é feita a personalização da operação **InsereProcessoAccaoTrabalhista**, com destaque às operações que foram trocadas. Na caixa identificada com o texto "**Objectos de Funções**

Sugeridas" são apresentadas o nome dos objectos que sofrem interacções durante a operação e o nome das funções, sugeridas pelo AGEAT, que vão ser utilizadas na execução dos testes. Para cada objecto gráfico que é seleccionado nesta primeira caixa, as funções que estão disponíveis, consoante seu tipo, são apresentadas na caixa inferior "**Funções possíveis para o objecto**". O utilizador poderá alterar a função sugerida e utilizar aquela que lhe for mais conveniente.

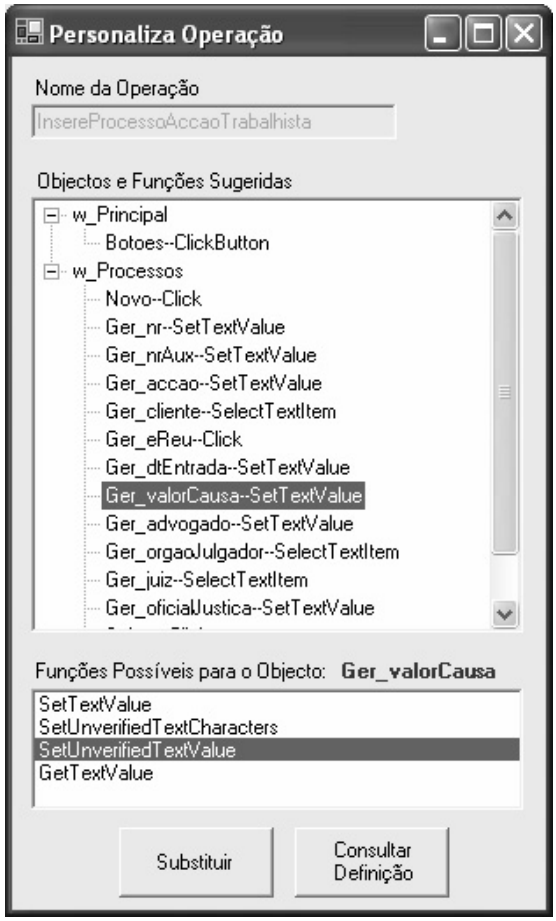

Figura 6.11- Personalização da operação InsereProcessoAccaoTrabalhista

Todos os testes que compõem o ciclo **"regressao"** foram executados com sucesso e nenhuma interacção manual foi necessária. Uma listagem completa do resultado obtido encontra-se no Anexo B.3.

6.3. Comparação entre as abordagens: windows e web

Durante a realização destes dois casos de estudo, verificamos algumas diferenças entre os sistemas *Web* e *Windows* que influenciaram a forma de criação dos testes automáticos, que destacamos nesta secção.

A primeira diferença tem a ver com o número de objectos *windows* existentes. Enquanto num sistema *windows* temos normalmente várias janelas, num sistema *web*, temos frequentemente uma janela principal que contém todos os objectos gráficos, mas que são visualizados gradualmente de acordo com o que é seleccionado no *frame* de navegação.

Constatamos que a criação das operações de teste para o sistema *web* exigiu-nos maior esforço do que a tarefa de criação de operações para o sistema Windows, dado o número elevado de objectos gráficos pertencentes a uma única janela. O esforço foi minimizado ao utilizarmos a regra descrita na Secção 6.1.1 para a identificação dos objectos gráficos.

A segunda diferença está relacionada com a forma de apresentação dos dados e de manipulação existente para cada tipo de sistema. Embora as características que destacamos de seguida se verifiquem para muitos sistemas *web*, sabemos que não constituem requisitos de construção de tais sistemas e portanto existirão vários sistemas *web* onde tais características não são verificadas.

Enquanto no sistema *L'Avocat* temos uma dissociação entre os dados manipulados e os objectos gráficos que permitem a sua manipulação, no sistema Fénix *web*, esta dissociação não existe. Por exemplo, as qualificações académicas dum docente são apresentadas numa tabela, onde cada linha contém os dados de uma certa qualificação. Ao final de cada linha na tabela apresentada temos dois *links* com texto igual a "**Editar**" e "**Apagar**", respectivamente. Ao seleccionarmos o *link* **Editar** da primeira linha de qualificações apresentada, estamos a indicar que queremos alterar os dados desta primeira linha. Se seleccionarmos o *link* **Apagar** da terceira linha, estamos a indicar que queremos remover a qualificação apresentada na terceira linha.

Por estes motivos as operações **alterar_qualificacao_academica** e **remover_qualificacao_academica,** ao serem criadas, foram específicas para determinada qualificação académica escolhida durante a criação da operação. Se quiséssemos realizar a alteração de uma qualificação académica diferente, uma nova operação deveria ser criada, com referência ao *link* **Editar** associado à qualificação académica pretendida.

Para o sistema *L'Avocat*, na janela de processos, por exemplo, só é possível visualizar um processo de cada vez. Caso se quisesse visualizar os dados de um novo processo, teria-se que o seleccionar no *combobox* existente para este efeito. Os objectos gráficos, nomeadamente os botões "**Editar**", "**Salvar**", "**Arquivar**" e "**Excluir**", permitem a manipulação dos dados de qualquer processo que esteja correntemente seleccionado. As operações **CriaAndamentoProcesso** e **AlteraAndamentoProcesso** podem ser utilizadas para a manipulação de qualquer processo, pois

têm como parâmetro o número que identifica o processo a ser manipulado e que será consultado no *combobox* existente no início da operação.

Capítulo 7
Conclusão

Durante o desenvolvimento desta dissertação procurámos conhecer em detalhe as características das ferramentas *capture-replay,* para o conhecimento de seus pontos mais vulneráveis e percepção realista sobre suas capacidades.

Após esta fase concluímos que as técnicas de criação de *scripts*, nomeadamente *keyword-driven* e *data-driven,* apresentadas no Capítulo 2, fornecem uma estrutura eficaz e sólida para a criação de testes automáticos estruturados. Em particular, a técnica *keyword-driven* oferece uma abordagem de separação entre a concepção e a implementação do teste. Esta separação permite a divisão do esforço de realização das tarefas de teste distintas por diferentes perfis de testadores. No entanto, a tarefa de construção ainda representa desenvolvimento de software e como tal, exige um tempo de realização relativamente extenso.

De seguida investigámos em detalhe a *framework* WRAFS [WRAFS2005] e em menor profundidade as *frameworks TestFrame* [BuwaldaJanssenPinkster2002] e CSDDT (Control Synchronized Data-Driven Testing) [ArcherCSDDT]. Todas seguem a técnica *keyword-driven* e oferecem ao testador algumas *keywords* e o código de sua implementação já desenvolvido de modo que possam utilizá-las na construção de seus testes. Após esta fase concluímos que tais *frameworks* constituem recurso valioso para os testes automáticos. Em [MosleyPosey2002] a expressão *framework-driven* é usada inclusive para descrever a realização dos testes através de *frameworks* que separam o sistema alvo dos *scripts* de teste, providenciando um conjunto de funções em uma biblioteca compartilhada.

No entanto, a utilização de testes automáticos e seus benefícios podem ser superiores caso o testador não precise dispender tempo na aprendizagem de *frameworks* e na criação manual de ficheiros extensos com referência às funções disponibilizadas pela *framework* ou desenvolvidas à medida. Pensamos que podíamos avançar para a facilitação da construção dos testes automáticos, tal como enunciado num dos nossos principais objectivos da tese:

> **"Identificação de uma abordagem para construção de testes automáticos que não requeira conhecimento de *frameworks* específicas ou em linguagens de programação de ferramentas *capture-replay*"**

Para a satisfação deste objectivo precisámos reflectir sobre os testes funcionais realizados sobre aplicações com interface gráfica. Concluímos que os tipos de componentes gráficos que constituem a interface gráfica do sistema alvo são recorrentes e conhecidos (e.g. botões, caixas de texto), bem como as acções que podem ser realizadas por um utilizador (e.g. click sobre botão, introdução de dados). Além disto, sabemos que os testes sobre interfaces gráficas são basicamente constituídos por sequências de acções sobre componentes gráficos definidas com objectivo de exercitar o sistema de diferentes maneiras.

Baseado nesta constatação, pensamos que uma abordagem apropriada deveria relacionar os tipos de componentes gráficos às acções permitidas em cada tipo de componente. Esta relação permitiria a criação de diferentes cenários para diferentes contextos de utilização onde, cada cenário relaciona, uma e apenas uma acção, para cada tipo de componente gráfico. Esta relação directa entre tipos de componentes e respectivas acções foi a ideia subjacente à criação de um sistema que permitisse a **geração** automática de testes. Esta geração automática implica uma redução do esforço de construção e manutenção dos testes, permitindo que tenhamos conseguido atingir o objectivo enunciado:

> **"Criação de sistema que permita a redução do esforço de construção e manutenção de testes automáticos. Tal sistema deverá utilizar uma ferramenta *capture-replay* e uma *framework opensource* existente"**

Para validação deste trabalho e consequente satisfação do último objectivo enunciado no Capítulo 1, seguimos um processo de criação de testes orientado pela utilização exclusiva do sistema AGEAT aplicado a dois sistemas reais, nomeadamente *Fénix [FenixIST]* e *L'Avocat [LavocatUS4]*. Os testes foram executados e seus resultados analisados. Variados testes puderam ser criados e executados sem restrições, quer para o sistema com interface *web,* quer para o sistema com interface *windows*. Durante a realização destes casos de estudo nos deparámos com situações particulares onde foi possível verificar, com satisfação, a flexibilidade do sistema AGEAT.

7.1. Trabalho Futuro

Analisam-se de seguida possíveis caminhos que podem ser desenvolvidos/percorridos em trabalho futuro. Identificamos tal percurso segundo duas pespectivas: (1) evolução sobre o sistema AGEAT actual e (2) adaptação da solução AGEAT para a utilização de outras *frameworks,* com o objectivo de permitir a sua integração e utilização com outras ferramentas *capture-replay.*

Segundo a primeira pespectiva, propomos três melhorias à solução AGEAT actual, nomeadamente: criação de mecanismo para interpretação dos resultados dos testes, registo histórico dos resultados

das execuções e disponibilização dos comandos *drivers* implementados pela *framework* WRAFS a testadores mais experientes e com necessidades mais específicas. Segundo a segunda pespectiva, propomos a utilização do AGEAT para outras *frameworks* do projecto SAFS.

7.1.1. Interpretação de Resultados

No Capítulo 4, entre os aspectos desfavoráveis à framework WRAFS foi discutido aquele relacionado com a forma de verificação dos resultados. No entanto, o sistema AGEAT não implementa melhorias significativas neste sentido, uma vez que apenas apresenta, a partir de sua interface, o resultado da execução já registada nos ficheiros de *logs* da *framework*.

Em tais ficheiros de *log,* para cada acção, poderão ser atribuídos resultados bem conhecidos, nomeadamente: (1) OK – Nenhum erro registado na realização da Acção; (2) – Warning – Situação de incoerência que precisa de ser verificada (não é necessariamente atribuído a um erro de execução do teste); (3) Failed – Quando a acção pretendida não foi realizada (os motivos para tal acontecer podem ser vários, por exemplo, o objecto não é encontrado ou o valor pelo qual se fará a pesquisa nos itens de um combobox, não existe).

Uma vez que os ficheiros de *log* criados pela *framework* WRAFS seguem um formato bem definido, onde para cada acção realizada é registado o resultado de sua execução, pensamos que o sistema AGEAT poderia realizar uma interpretação do conteúdo dos ficheiros e apresentar a informação de forma mais resumida ao testador. Desta forma, ao consultar a interpretação feita pelo sistema o testador saberia de imediato os *warnings* e erros graves ocorridos na execução do teste, com a indicação da operação, janela e objecto gráfico sob os quais ocorreram. A leitura sequencial sob todo o resultado da execução seria, na maioria das vezes, desnecessária.

7.1.2. Registo das Execuções

Outra limitação identificada no Capítulo 4 foi o facto da *framework* WRAFS não manter o histórico das execuções e por isto não ser possível a verificação de resultados de execuções passadas. O AGEAT também não implementou qualquer melhoria neste sentido.

Através da extensão do modelo de domínio para representação de cada execução do teste (execução do ciclo, baterias ou operações) e resultado associado, seria facilmente possível a consulta do histórico das execuções de teste.

7.1.3. Disponibilização de comandos *drivers*

A geração automática de testes feita pelo sistema AGEAT utiliza, basicamente, as funções existentes para cada tipo de componente gráfico. No entanto, a biblioteca de funções da *framework*

WRAFS fornece um conjunto de comandos *drivers* que disponibilizam funcionalidades acrescidas. Alguns exemplos de comandos *drivers* são visualizados na Tabela 7.1.

Comando *driver*	**Descrição**
SetApplicationMap	Carrega um ficheiro guimap
LaunchApplication	Lança uma aplicação
CallScript	Executa um script automático
Pause	Interrompe a execução por certo intervalo
ExecSQLQuery	Executa uma query SQL numa base de dados

Tabela 7.1 - Exemplos de comandos drivers

O comando *driver SetApplicationMap é* utilizado pelo sistema AGEAT durante a geração automática, mas pensamos que uma funcionalidade desejada para o sistema seria disponibilizar ao testador todos os outros comandos *drivers* existentes, nos casos em que fosse necessário a sua utilização.

É óbvio que a utilização de tais comandos seria restrita a utilizadores com conhecimento em automatização de testes e na estrutura interna da *framework* WRAFS, uma vez que a inclusão de tais comandos durante a construção dos testes, seria feita a pedido do testador. Cada comando possui sua assinatura específica e os valores de atribuição a seus respectivos parâmetros seria feito pelo testador através da interface gráfica.

A utilização de comandos que realizam *querys* em bases de dados, possibilitariam, inclusivamente, a verificação de resultados dos testes com validações directas sobre a informação existente na base de dados do sistema alvo.

7.1.4. Adaptação do AGEAT para outras frameworks

Uma vez que o AGEAT foi criado sobre a framework WRAFS, é requerido que seja utilizado a ferramenta *Winrunner* para execução de testes. No entanto, uma vez que o projecto SAFS possui *frameworks* desenvolvidas para outras ferramentas, entre as quais está a mais conhecida RRAFS para a ferramenta *Rational Robot*, muito semelhante à WRAFS, o sistema AGEAT poderá vir a ser utilizado também para testes que correrão em *Rational Robot*. A Figura 7.1 apresenta as camadas de plataformas da configuração actual em comparação com as futuras que levam a esta evolução.

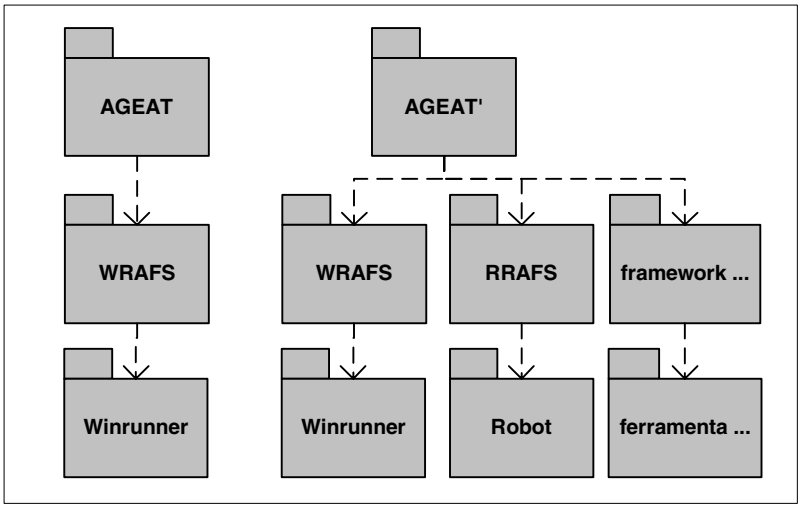

Figura 7.1 - Comparação entre as camadas de plataformas actuais e as futuras

Considerando que assinaturas das funções de componentes são as mesmas para as *frameworks* WRAFS e RRAFS, nenhuma alteração significativa no modelo de domínio seria necessária para a utilização do AGEAT sobre a *framework* RRAFS. Apenas seria necessário ter-se a *framework* RRAFS e a ferramenta *Rational Robot* instaladas. Todos os ajustes poderiam ser realizados através da actualização de novos valores de configuração nomeadamente, nome e directoria do executável da ferramenta *capture-replay* a ser utilizado e valores de algumas variáveis a serem passadas no arranque deste executável.

Uma vez que o projecto SAFS está actualmente a desenvolver uma única *framework* que poderá ser utilizada por várias ferramentas *capture-replay* um bom tema de desenvolvimento futuro seria a adapatação do sistema AGEAT para utilização sobre esta nova *framework*. Desta forma o sistema AGEAT poderia vir a ser utilizável sobre qualquer ferramenta *capture-replay* suportada pela *framework* SAFS. A Figura 7.2 apresenta as camadas de plataformas desta proposta de evolução.

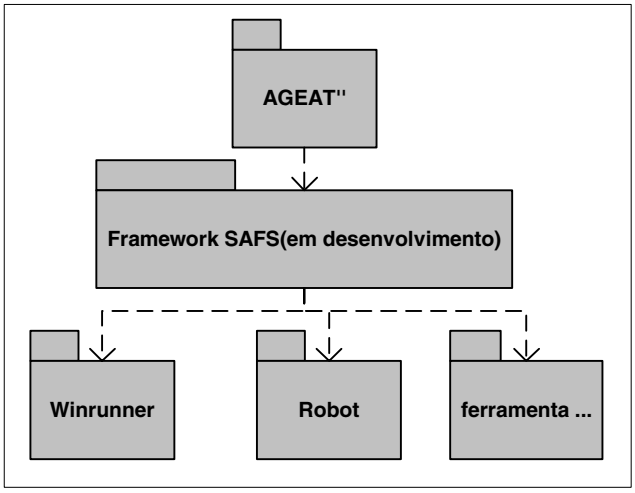

Figura 7.2 - Camadas de plataformas utilizando a framework SAFS

Referências

[ADL] ADL Translation System. http://adl.opengroup.org/

[Akervold2004] Akervold K.: *Framework for Automated Software Testing*. Tese de Mestrado. Fakult for Informasjonsteknologi, Matematikk Og Elektroteknikk, 2004.

[ArcherCSDDT] Archer Group. *Control Synchronized Data-Driven Testing Framework*. www.phptr/mosley.

[Beck2000] Beck K.: *Extreme Programming Explained*. Addison-Wesley, 2000.

[BerezaJarocinski2001] Bereza B., Jarocinski: Tools and Automation in a Shoestring. *European conference on Software Testing (WorkShop Eurostar)*. Estocolmo, 2001.

[BuwaldaJanssenPinkster2002] Buwalda H., Janssen D. e Pinkster I.: *Integrated Test Design and Automation: Using the TestFrame Method*. Addison-Wesley, 2002.

[CorreiaSilva2004] Correia S., Silva A.: *Técnicas para Construção de Testes Funcionais Automáticos*. V Conferência para Qualidade nas Tecnologias da Informação e Comunicações - Quatic, 2004.

[DoughertyHaber2002] Dougherty J., Haber K.: Test Automation: Reducing Time to Market. *International Conference on Software Testing, Analysis & Review,* 2002.

[DustinRashka2001] Dustin E., Rashka J.: *Automated Software Testing: Introduction, Management and Performance*. Addison – Wesley, 2001.

[FenixIST] Sistema Fenix. http://www.ist.utl.pt/

[FewsterGrahm1999] Fewster M., Grahm D.: *Software Test Automation*. Addison-Wesley, 1999.

[Hendricson1998] Hendricson E.: White Paper: *The Difference between Test Automation Failure and Success, 1998.* www.qualitytree.com/feature/dbtasaf.pdf'

[Hendrickson1999] Hendrickson E.: *Making the Right Choice*. Revista StqeMagazine, Maio/Junho 1999. http://www.qualitytree.com/feature/mtrc.pdf

[HorwathGreenLawler2000] Horwath, T., Green, J., Lawler, T.: White Paper: *SilkTest and WinRunner Feature Descriptions*, 2000.

[IPL] IPL, www.iplbath.com (AdaTEST, Cantata, Cantata++)

[ISO8402] ISO: 8402 – *Quality Managment and Quality Assurance* – Vocabulary, 2000.

[Kaner1997] Kaner C.: Improving the Maintainability of Automated Test Suites. *Quality Week Conference,* 1997.

[KanerFalkNguyen1999] Kaner G., Falk J., Nguyen H.Q.: *Testing Computer Software*. John Wiley & Sons, 1999.

[Kit1995] Kit, E.: *Software Testing in the Real World: Improving the Process.* Addison-Wesley, 1995.

[Kit1999] Kit. E.: *Integrated Effective Test Design and Automation.* Software Development 7, no 2, 1999

[KoomenPol1999] Koomen T., Pol M.: *Process Improvement*: *A practical step-by-step guide to sctructed testing*. Addison-Wesley, 1999.

[LavocatUS4] Sistema *L'Avocat* desenvolvido pela extinta empresa Us4, 1999.

[LDRA] LDRA Ltd, www.ldra.com (LDRA Testbed, geração automática de ambiente de testes - harness, stubs e drivers)

[Leite1998] Leite R. A.: *Um Ambiente de Testes Automáticos para o SIS*. Tese de Mestrado em Ciência da Computação. Instituto de Ciências Exatas, Universidade Federal de Minas Gerais, Brasil, 1998.

[LinzDaigl1998] Linz T., Matthias D.: White Paper*: How to Automate Testing of Graphical User Interfaces*. Alemanha, 1998. http://www.imbus.de/forschung/pie24306/gui/aquis-full_paper-1.3.html.

[Linz1998] Linz T.: *Case Study: IMBUS GmbH. Automated Testing of Graphical User Interfaces*, 1998. www.esi.es/VASIE/Reports/All/24151/gui-test1.pdf

[Mcconnell1996] Mcconnell S.: *Rapid Development*. Microsoft Press, 1996.

[Momeni2001] Momeni N.: *Functional Testing of GUI Based Software without Using the GUI*. Tese de mestrado. Lund University. Sweden. 2001.

[MosleyPosey2002] Mosley D. J., Posey B. A.: *Just Enough Software Test Automation.*. Prentice Hall PTR, 2002.

[Myers1979] Myers G.J.: *The Art of Software Testing*. New York: John Wiley and Sons, 1979.

[Nagle2000] Nagle C.: *Test Automation Frameworks.* SAS Institute. http://safsdev.sourceforge.net/FRAMESDataDrivenTestAutomationFrameworks.htm

[Pettichord2001] Pettichord B.: Seven Steps to Test Automation Sucess. *STAR West Conference*, 1999. Versão revisada em 2001

[Pfleeger2001] Pfleeger S. L.: *Software Engineering: Theory and Practice*. Prentice Hall, 2001.

[PolVeenendaal1998] Pol M., Veenendaal E. V.: *Structured Testing of Information Systems: an Introduction to Tmap*. Kluwer, 1998.

[ProjectIT2005] Projecto de I&D *ProjectIT*. http://berlin.inesc-id.pt/alb/ProjectIT@81.aspx

[RobotInfo] Ferramenta Rational Robot. http://www-306.ibm.com/software/awdtools/tester/robot/

[RunnerInfo] Informação WinRunner. http://www.mercury.com/us/products/quality-center/functional-testing/winrunner/

[RunnerManual76] WinRunner User's Guide Versão 7.6, Mercury Interactive Corporation.

[Sommerville2000] Sommerville I.: *Software Engineering*. Addison-Wesley, 2000.

[Testwell] Testwell, www.teswell.sci.fi/homepage.html (ferramentas de teste para C, C++ e Java)

[WRAFS2005] Documentaçao sobre a Framework WRAFS. http://safsdev.sourceforge.net/Default.htm

[Zambelich1999] Zambelich K.: *Totally Data-driven Automated Testing. With Paper*, 1999. http://www.sqa-test.com/w_paper1.html

Apêndice A Artefactos de Testes do Sistema Fénix

A.1 Ficheiro *Fenix.gui*

A seguir apresenta-se o conteúdo do ficheiro texto ***Fenix.gui*** necessário à construção dos testes automáticos relativos ao caso de estudo do sistema *web* **Fénix,** descrito no Capítulo 6. Para um melhor aproveitamento de espaço e legibilidade, utilizou-se um formato tabular, onde cada célula corresponde um objecto gráfico do sistema Fénix.

w_Docente:{ class: window, MSW_class: html_frame, html_name: ".IST - Docente"} { rtree_state: open, ltree_state: open}
w_Docente.CarrDocen_fim:{ class: edit, MSW_class: html_edit, html_name: endYear}
w_Docente.AdministracaodePublicacoes:{ class: object, MSW_class: tml_text_link, html_name: "Administração de Publicações"}
w_Docente.CarrDocen_inicio:{ class: edit, MSW_class: html_edit, html_name: beginYear}
w_Docente.Cancelar:{ class: push_button, MSW_class: html_push_button, html_name: Cancelar}
w_Docente.Confirmar:{ class: push_button, MSW_class: html_push_button, html_name: Confirmar}
w_Docente.CarrDocen_categoria:{ class: list, MSW_class: html_combobox, html_name:"infoCategory#idInternal"}
w_Docente.Criar:{ class: push_button, MSW_class: html_push_button, html_name: Criar}
w_Docente.CarrDocen_diciplina:{ class: edit, MSW_class: html_edit, html_name: courseOrPosition}
w_Docente.FichadeDocente:{ class: object, MSW_class: html_text_link, html_name: "Ficha de Docente"}
w_Docente.CarrDocen_editar:{ class: object, MSW_class: html_text_link, html_name: Editar, location: 1}
w_Docente.Public_ambito:{ class: list, MSW_class: html_combobox, html_name: scope}
w_Docente.Public_ano:{ class: edit, MSW_class: html_edit, html_name: year}
w_Docente.Public_jornalRevista:{ class: edit, MSW_class: html_edit, html_name: journalName}

w_Docente.Public_apagar:{ class: object, MSW_class: html_text_link, html_name: Apagar, location: 5}
w_Docente.Public_mes:{ class: list, MSW_class: html_combobox, html_name: month}
w_Docente.Public_eCientifica:{ class: radio_button, MSW_class: html_radio_button, html_name: isDidatic, part_value: 0}
w_Docente.Public_numero:{ class: edit, MSW_class: html_edit, html_name: number}
w_Docente.Public_eDidatica:{ class: radio_button, MSW_class: html_radio_button, html_name: isDidatic, part_value: 1}
w_Docente.Public_observacao:{ class: edit, MSW_class: html_edit, html_name: observation}
w_Docente.Public_editar:{ class: object, MSW_class: html_text_link, html_name: Editar, location: 2}
w_Docente.Public_primeiraPag:{ class: edit, MSW_class: html_edit, html_name: firstPage}
W_Docente.Public_formato:{ class: list, MSW_class: html_combobox, html_name: format}
w_Docente.Public_subtipo: class: list, MSW_class: html_combobox, html_name: subtype}
w_Docente.Public_idioma:{ class: edit, MSW_class: html_edit, html_name: language}
w_Docente.Public_tipo:{ class: list, MSW_class: html_combobox, html_name: infoPublicationTypeId}
w_Docente.Public_ultimaPag:{ class: edit, MSW_class: html_edit, html_name: lastPage}
w_Docente.QualifAcad_nota:{ class: edit, MSW_class: html_edit, html_name: mark}
w_Docente.Public_volume:{ class: edit, MSW_class: html_edit, html_name: volume}
w_Docente.alterar:{ class: push_button, MSW_class: html_push_button, html_name: Alterar}
w_Docente.QualifAcad_ano:{ class: edit, MSW_class: html_edit, html_name: tempDate}
w_Docente.apagar:{ class: object, MSW_class: html_text_link, html_name: Apagar, location: 3}
w_Docente.QualifAcad_curso:{ class: edit, MSW_class: html_edit, html_name: degree}
w_Docente.apagarCampos:{ class: push_button, MSW_class: html_push_button, html_name: Apagar}
w_Docente.QualifAcad_editar:{ class: object, MSW_class: html_text_link, html_name: Editar, location: 0}
w_Docente.continuarFichaDocente:{ class: push_button, MSW_class: tml_push_button, html_name: Continuar}
w_Docente.QualifAcad_escola:{ class: edit, MSW_class: html_edit, html_name: school}
w_Docente.guardar:{ class: push_button, MSW_class: html_push_button, html_name: Guardar}

w_Docente.QualifAca_grau:{ class: edit, MSW_class: html_edit, html_name: title}
w_Docente.inserir:{ class: object, MSW_class: html_text_link, html_name: Inserir}
W_Login:{ class: window, MSW_class: html_frame, html_name: ".IST - Login" } { rtree_state: open, ltree_state: close}
w_Login.username:{ class: edit, MSW_class: html_edit, html_name: username}
w_Login.Limpar:{ class: push_button, MSW_class: html_push_button, html_name: Limpar}
w_Pessoal:{ class: window, MSW_class: html_frame, html_name: ".IST - Área Pessoal"} { rtree_state: open, ltree_state: open}
w_Login.Submeter:{ class: push_button, MSW_class: html_push_button, html_name: Submeter}
w_Pessoal.Docencia:{ class: object, MSW_class: html_text_link, html_name: "Docência"}
w_Login.password:{ class: edit, MSW_class: html_edit, html_name: password }
w_Pessoal.Logout:{ class: object, MSW_class: html_rect, html_name: Logout}

A.2 Composição das Baterias de Teste

A seguir, apresenta-se a informação relativa a composição das baterias de testes criadas para o caso de estudo do sistema web Fénix, descrito no Capítulo 6. A leitura da tabela de modo sequencial fornece o fluxo de execução de todas as baterias de teste construídas.

Bateria ou *Operação* ou Janela	**Objecto Gráfico**	**Valor de Parâmetro Associado**
loggin		
efectua_loggin		
w_Login	username	D2628
w_Login	Password	pass
W_Login	Submeter	
exercita_qualificacoes		
consultar_qualificacao_academica		
w_pessoal	docencia	
w_Docente	FichaDocente	
w_Docente	QualifAcad_editar	
alterar_qualificacao_academica		

w_Docente	QualifAcad_editar	
w_Docente	QualifAcad_ano	1999
w_Docente	QualifAcad_escola	Instituto Superior Tecnico
W_Docente	QualifAcad_grau	Doutor
W_Docente	QualifAcad_curso	Engenharia Informatica
W_Docente	QualifAcad_nota	Unanimidade
W_Docente	Guardar	
inserir_carreira_docente		
W_Docente	Inserir	
W_Docente	QualifAcad_ano	2005
W_Docente	QualifAcad_escola	UFPE
W_Docente	QualifAcad_grau	Licenciatura
W_Docente	QualifAcad_curso	Ciencia da Computacao
W_Docente	QualifAcad_nota	Unanimidade
W_Docente	guardar	
remover_qualificacao_academica		
W_Docente	Apagar	
w_Docente	continuarFichaDocente	
exercita_carreira		
Consultar_carreira_docente		
w_pessoal	Docencia	
W_Docente	FichadeDocente	
W_Docente	CarrDocen_editar	
alterar_carreira_docente		
W_Docente	CarrDocen_editar	
W_Docente	CarrDocen_inicio	1999
W_Docente	CarrDocen_fim	2003
W_Docente	CarrDocen_categoria	PROF.AUXILIAR
W_Docente	CarrDocen_diciplina	Engenharia da Programação

W_Docente	guardar	
Inserir_carreira_docente		
W_Docente	inserir	
W_Docente	CarrDocen_inicio	2005
W_Docente	CarrDocen_fim	2006
W_Docente	CarrDocen_categoria	PROF.ADJUNTO
W_Docente	CarrDocen_diciplina	Inteligência Artificial
W_Docente	Guardar	
Remover_carreira_docente		
W_Docente	apagar	
W_Docente	continuarFichaDocente	
exercita_publicacoes		
consultar_publicacao		
w_Pessoal	Docencia	
W_Docente	AdministracaodePublicacoes	
alterar_publicacao		
W_Docente	AdministracaodePublicacoes	
W_Docente	Public_editar	
W_Docente	Public_eCientifica	
W_Docente	alterar	
Inserir_publicacao		
W_Docente	AdministracaodePublicacoes	
W_Docente	inserir	
W_Docente	Public_eDidatica	
W_Docente	Public_tipo	Jornal Article
W_Docente	Public_subtipo	Complete
W_Docente	Public_jornalRevista	CAPSI
W_Docente	QualifAcad_grau	Testes automáticos
W_Docente	Public_volume	1

W_Docente	Public_primeiraPag	34
W_Docente	Public_ultimaPag	41
W_Docente	Public_idioma	Português
W_Docente	Public_numero	4
W_Docente	Public_observacao	N/A
W_Docente	Public_formato	Paper
W_Docente	Public_mes	Outubro
W_Docente	Public_ano	2005
W_Docente	Public_ambito	Nacional
W_Docente	Criar	
Apagar_publicacao		
w_Docente	Public_apagar	
W_Docente	Confirmar	
Logout		
Efectua_logout		
w_Pessoal	Logout	

A.3 Resultado da Execução dos Testes

Abaixo apresenta-se o resultado da execução automática do ciclo "**docencia"**.

..... START LOGGING RESUME Text Log in StepDriver
SetTextValue d2628 sent to username.
OK username SetTextValue to d2628 successful.
SetTextValue pass sent to password.
OK password SetTextValue to pass successful.
OK w_Login:Submeter CLICKED.
OK Docencia CLICKED
OK FichadeDocente CLICKED
OK QualifAcad_editar CLICKED

OK QualifAcad_editar CLICKED
SetTextValue 1999 sent to QualifAcad_ano.
OK QualifAcad_ano SetTextValue to 1999 successful.
SetTextValue Instituto Superior Tecnico sent to QualifAcad_escola.
OK QualifAcad_escola SetTextValue to Instituto Superior Tecnico successful.
SetTextValue Doutor sent to QualifAcad_grau.
OK QualifAcad_grau SetTextValue to Doutor successful.
SetTextValue Engenharia Informatica sent to QualifAcad_curso.
OK QualifAcad_curso SetTextValue to Engenharia Informatica successful.
SetTextValue Unanimidade sent to QualifAcad_nota.
OK QualifAcad_nota SetTextValue to Unanimidade successful.
OK w_Docente:guardar CLICKED.
OK inserir CLICKED
SetTextValue 2005 sent to QualifAcad_ano.
OK QualifAcad_ano SetTextValue to 2005 successful.
SetTextValue UFPE sent to QualifAcad_escola.
OK QualifAcad_escola SetTextValue to UFPE successful.
SetTextValue Licenciatura sent to QualifAcad_grau.
OK QualifAcad_grau SetTextValue to Licenciatura successful.
SetTextValue Ciencia da Computacao sent to QualifAcad_curso.
OK QualifAcad_curso SetTextValue to Ciencia da Computacao successful.
SetTextValue Unanimidade sent to QualifAcad_nota.
OK QualifAcad_nota SetTextValue to Unanimidade successful.
OK w_Docente:guardar CLICKED.
OK apagar CLICKED
OK w_Docente:continuarFichaDocente CLICKED.
OK Docencia CLICKED
OK FichadeDocente CLICKED
OK CarrDocen_editar CLICKED

OK CarrDocen_editar CLICKED
SetTextValue 1999 sent to CarrDocen_inicio.
OK CarrDocen_inicio SetTextValue to 1999 successful.
SetTextValue 2003 sent to CarrDocen_fim.
OK CarrDocen_inicio SetTextValue to 2003 successful.
SetTextValue PROF.AUXILIAR sent to CarrDocen_categoria.
OK CarrDocen_categoria SetTextValue to PROF.AUXILIAR successful.
SetTextValue Engenharia da Programacao sent to CarrDocen_disciplina.
OK CarrDocen_disciplina SetTextValue to Engenharia da Programacao successful.
OK w_Docente:guardar CLICKED.
OK inserir CLICKED
SetTextValue 2005 sent to CarrDocen_inicio.
OK CarrDocen_inicio SetTextValue to 2005 successful.
SetTextValue 2006 sent to CarrDocen_fim.
OK CarrDocen_inicio SetTextValue to 2006 successful.
SetTextValue PROF.ADJUNTO sent to CarrDocen_categoria.
OK CarrDocen_categoria SetTextValue to PROF.ADJUNTO successful.
SetTextValue Inteligência Artificial sent to CarrDocen_disciplina.
OK CarrDocen_disciplina SetTextValue to Inteligência Artificial successful.
OK w_Docente:guardar CLICKED.
OK apagar CLICKED
OK w_Docente:continuarFichaDocente CLICKED.
OK Docencia CLICKED
OK AdministracaodePublicacoes CLICKED
OK AdministracaodePublicacoes CLICKED
OK Public_editar CLICKED
w_Docente:Public_eCientifica CLICKED.
OK w_Docente:alterar CLICKED.
OK AdministracaodePublicacoes CLICKED

OK inserir CLICKED
w_Docente:Public_eDidatica CLICKED.
SetTextValue CAPSI sent to Public_jornalRevista.
OK Public_jornalRevista SetTextValue to CAPSI successful.
SetTextValue Testes automáticos sent to QualifAcad_grau.
OK QualifAcad_grau SetTextValue to Testes automáticos successful.
SetTextValue 1 sent to Public_volume.
OK Public_volume SetTextValue to 1 successful.
SetTextValue 34 sent to Public_primeiraPag.
OK Public_primeiraPag SetTextValue to 34 successful.
SetTextValue 41 sent to Public_ultimaPag.
OK Public_ultimaPag SetTextValue to 41 successful.
SetTextValue Português sent to Public_idioma.
OK Public_idioma SetTextValue to Português successful.
SetTextValue 4 sent to Public_numero.
OK Public_numero SetTextValue to 4 successful.
SetTextValue N/A sent to Public_observacao.
OK Public_observacao SetTextValue to N/A successful.
SetTextValue 2005 sent to Public_ano.
OK Public_ano SetTextValue to 2005 successful.
OK w_Docente:Criar CLICKED.
OK Public_apagar CLICKED
OK w_Docente:Confirmar CLICKED.
OK w_Pessoal:Logout CLICKED

Apêndice B Artefactos de Testes do Sistema L'Avocat

B.1 Ficheiro *Lavocat.gui*

A seguir apresenta-se o conteúdo do ficheiro texto **lavocat.gui** necessário à construção dos testes automáticos relativos ao caso de estudo do sistema *windows* ***L'Avocat,*** descrito no Capítulo 6.

w_Alerta:{ class: window, label: "L'Avocat", MSW_class: "#32770"} {rtree_state: open, ltree_state: close}
w_Alerta."(static)":{ class: static_text, MSW_id: 20}
w_Alerta.No:{ class: push_button, label: No}
w_Alerta.OK:{ class: push_button, label: OK}
w_Alerta.Yes:{ class: push_button, label: Yes}
w_Alerta.mensagemdealerta:{ class: static_text, MSW_id: 65535}
w_Andamento:{ class: window, label: Andamento, MSW_class: ThunderRT5Form} { rtree_state: open, ltree_state: close}
w_Andamento.Cancelar:{ class: push_button, MSW_id: 2}
w_Andamento.Incluir:{ class: push_button, MSW_id: 3}
w_Andamento.data:{ class: edit, MSW_id: 0}
w_Andamento.descricao:{ class: edit, MSW_id: 1}
w_Clientes:{ class: window, label: Clientes, MSW_class: ThunderRT5Form} { rtree_state: open, ltree_state: close}
w_Clientes.Adic_conjuge:{ class: edit, MSW_id: 10}
w_Clientes.Adic_estadocivil:{ class: list, MSW_id: 13}
w_Clientes.Adic_nacionalidade:{ class: edit, MSW_id: 11}
w_Clientes.Adic_nascConjuge:{ class: edit, MSW_id: 0, location: 2}
w_Clientes.Adic_nascimento:{ class: edit, MSW_id: 0, location: 1}

w_Clientes.Adic_observacoes:{ class: edit, MSW_id: 12}
w_Clientes.Adic_profissao:{ class: edit, MSW_id: 9}
w_Clientes.Arquivar:{ class: push_button, MSW_id: 2}
w_Clientes.Editar:{ class: push_button, MSW_id: 7}
w_Clientes.Excluir:{ class: push_button, MSW_id: 4}
w_Clientes.Novo:{ class: push_button, MSW_id: 3}
w_Clientes.Prin_UF:{ class: list, MSW_id: 14}
w_Clientes.Prin_bi:{ class: edit, MSW_id: 0, location: 3}
w_Clientes.Prin_distrito:{ class: edit, MSW_id: 33 }
w_Clientes.Prin_fisica:{ class: radio_button, MSW_id: 28}
w_Clientes.Prin_freguesia:{ class: edit, MSW_id: 34}
w_Clientes.Prin_juridica:{ class: radio_button, MSW_id: 27}
w_Clientes.Prin_listadenomes:{ class: list, MSW_id: 22}
w_Clientes.Prin_morada:{ class: edit, MSW_id: 35}
w_Clientes.Prin_nif_fisica:{ class: edit, MSW_id: 0, location: 4}
w_Clientes.Prin_nome:{ class: edit, MSW_id: 32}
w_Clientes.Prin_titular_juridica:{ class: edit, MSW_id: 30}
w_Clientes.Proc_listaProcessos:{ class: object, MSW_id: 0, MSW_class: MSFlexGridLib.MSFlexGrid.1", location: 0}
w_Clientes.Restaurar:{ class: push_button, MSW_id: 1}
w_Clientes.Sair:{ class: push_button, MSW_id: 5}
w_Clientes.Salvar:{ class: push_button, MSW_id: 6}
w_Clientes.TabClientes:{ class: object, MSW_class: "TabDlg.SSTab.1"}
w_Clientes.Tel_adicionaralista:{ class: push_button, MSW_id: 21}
w_Clientes.Tel_comerc:{ class: radio_button, MSW_id: 25}
w_Clientes.Tel_fax:{ class: radio_button, MSW_id: 17}
w_Clientes.Tel_fixo:{ class: radio_button, MSW_id: 16}
w_Clientes.Tel_listaTelefones:{ class: object, MSW_id: 0, MSW_class: MSFlexGridLib.MSFlexGrid.1", location: 1}
w_Clientes.Tel_movel:{ class: radio_button, MSW_id: 18}

w_Clientes.Tel_numero:{ class: edit, MSW_id: 0, location: 0}
w_Clientes.Tel_removerdaLista:{ class: push_button, MSW_id: 19}
w_Clientes.Tel_residenc:{ class: radio_button, MSW_id: 24}
w_ClientesArquivados:{ class: window, label: "Clientes arquivados", SW_class: ThunderRT5Form} { rtree_state: open, ltree_state: close}
w_ClientesArquivados.Cancelar:{ class: push_button, MSW_id: 1}
w_ClientesArquivados.Restaurar:{ class: push_button, MSW_id: 2}
w_ClientesArquivados.listaClientes:{ class: list}
w_Login:{ class: window, label: "<No title>", MSW_class: ThunderRT5Form, location: 1} { rtree_state: open, ltree_state: close}
w_Login.OK:{ class: push_button, MSW_id: 2}
w_Login.password:{ class: edit, MSW_id: 3}
w_Login.username:{ class: edit, MSW_id: 4}
w_Principal:{ class: window, label: "L'Avocat Pessoal '99 - US4 Tecnologia", MSW_class: ThunderRT5MDIForm} { rtree_state: open, ltree_state: open}
w_Principal.Arquivo:{ class: menu_item, label: Arquivo, parent: none}
w_Principal.Botoes:{ class: toolbar}
w_Principal.Clientes:{ class: menu_item, label: "Clientes Ctrl+B", parent: Arquivo}
w_Principal.Orgaos:{ class: menu_item, label: "Órgãos Ctrl+O", parent: Arquivo}
w_Principal.Processos:{ class: menu_item, label: "Processos Ctrl+P", parent: Arquivo}
w_Principal.Sair:{ class: menu_item, label: "Sair Ctrl+S", parent: Arquivo}
w_Processos:{ class: window, label: Processos, MSW_class: ThunderRT5Form} { rtree_state: open, ltree_state: close}
w_Processos.And_editar:{ class: push_button, MSW_id: 35}
w_Processos.And_excluir:{ class: push_button, MSW_id: 16}
w_Processos.And_inserirAtrasado:{ class: push_button, MSW_id: 15}
w_Processos.And_listaAndamentos:{ class: object, MSW_id: 0, MSW_class: MSFlexGridLib.MSFlexGrid.1", location: 1}
w_Processos.And_novo:{ class: push_button, MSW_id: 34}

w_Processos.Arquivar:{ class: push_button, MSW_id: 5}
w_Processos.Editar:{ class: push_button, MSW_id: 12}
w_Processos.Excluir:{ class: push_button, MSW_id: 9}
w_Processos.Ger_accao:{ class: edit, MSW_id: 33}
w_Processos.Ger_advogado:{ class: edit, MSW_id: 30}
w_Processos.Ger_area:{ class: list, MSW_id: 28}
w_Processos.Ger_busca:{ class: edit, MSW_id: 3}
w_Processos.Ger_cliente:{ class: list, MSW_id: 20}
w_Processos.Ger_dtConclusao:{ class: edit, MSW_id: 0, location: 1}
w_Processos.Ger_dtEntrada:{ class: edit, MSW_id: 0, location: 0}
w_Processos.Ger_eAutor:{ class: radio_button, MSW_id: 19}
w_Processos.Ger_eReu:{ class: radio_button, MSW_id: 18}
w_Processos.Ger_juiz:{ class: list, MSW_id: 17}
w_Processos.Ger_listaprocessos:{ class: list, MSW_id: 21}
w_Processos.Ger_nr:{ class: edit, MSW_id: 26}
w_Processos.Ger_nrAux:{ class: edit, MSW_id: 25}
w_Processos.Ger_objecto:{ class: edit, MSW_id: 13}
w_Processos.Ger_observacao:{ class: edit, MSW_id: 31}
w_Processos.Ger_oficialJustica:{ class: edit, MSW_id: 32}
w_Processos.Ger_orgaoJulgador:{ class: list, MSW_id: 27}
w_Processos.Ger_parteAdversa:{ class: edit, MSW_id: 29}
w_Processos.Ger_procRelaccionado:{ class: push_button, MSW_id: 22}
w_Processos.Ger_procurar:{ class: push_button, MSW_id: 2}
w_Processos.Ger_valorCausa:{ class: edit, MSW_id: 24}
w_Processos.Imprimir:{ class: push_button, MSW_id: 4}
w_Processos.Novo:{ class: push_button, MSW_id: 8}
w_Processos.Sair:{ class: push_button, MSW_id: 10}
w_Processos.Salvar:{ class: push_button, MSW_id: 11}
w_Processos.TabProcessos:{ class: object, MSW_class: "TabDlg.SSTab.1"}

`w_Processos.agenda:{ class: push_button, MSW_id: 6}`
`w_Processos.arquivoMorto:{ class: push_button, MSW_id: 7}`

B.2 Composição das Baterias de Teste

A seguir, apresenta-se a informação relativa a composição das baterias de testes criadas para o caso de estudo do sistema *windows* ***L'Avocat***, descrito no Capítulo 6. A leitura da tabela de modo sequencial fornece o fluxo de execução de todas as baterias de teste construídas.

Bateria ou *Operação* ou Janela	**Objecto Gráfico**	**Valor de Parâmetro Associado**
efectua_login		
Login		
w_Login	username	Adm
w_Login	password	admin
w_Login	OK	
exercita_clientes		
InserePrimeiroCliente		
w_Principal	Botoes	Clientes
w_Clientes	Prin_nome	FILIPE SILVA
w_Clientes	Prin_morada	Av. Afonso Henriques
w_Clientes	Salvar	
w_Clientes	Sair	
InsereClientesemNome		
w_Principal	Botoes	Clientes
w_Clientes	Novo	
w_Clientes	Prin_morada	Av. Combatentes
w_Clientes	Prin_freguesia	Nossa Senhora de Fatima
w_Clientes	Prin_fisica	
w_Clientes	Salvar	
w_Alerta	OK	

w_Clientes	Sair	
InserirClientePessoaFisica		
w_Principal	Botoes	Clientes
w_Clientes	Novo	
w_Clientes	Prin_fisica	
w_Clientes	Prin_nome	FILIPE FERNANDES
w_Clientes	Prin_morada	Av. Berna
w_Clientes	Prin_freguesia	Nossa Senhora de Fatima
w_Clientes	Prin_distrito	Lisboa
w_Clientes	Prin_bi	4658531
w_Clientes	Salvar	
w_Clientes	Sair	
InserirClientePessoaJuridica		
w_Principal	Botoes	Clientes
w_Clientes	Novo	
w_Clientes	Prin_juridica	
w_Clientes	Prin_nome	BRUNO GUERREIRO
w_Clientes	Prin_morada	Av. de Roma
w_Clientes	Prin_freguesia	Areeiro
w_Clientes	Prin_distrito	Lisboa
w_Clientes	Prin_titular_juridica	12133334343455
w_Clientes	Salvar	
w_Clientes	Sair	
AlterarDadosAdicionaisCliente		
w_Principal	Botoes	Clientes
w_Clientes	Prin_listadenomes	FILIPE FERNANDES
w_Clientes	Editar	
w_Clientes	TabClientes	Dados Adicionais
w_Clientes	Adic_nacionalidade	Português

w_Clientes	Adic_nascimento	23041950
w_Clientes	Adic_profissao	Engenheiro
w_Clientes	Salvar	
w_Clientes	Sair	
IncluiTelefoneResidencial		
w_Principal	Botoes	Clientes
w_Clientes	Prin_listadenomes	BRUNO GUERREIRO
w_Clientes	Editar	
w_Clientes	TabClientes	Telefones/FAX
w_Clientes	Tel_fixo	
w_Clientes	Tel_residenc	
w_Clientes	Tel_numero	213456666
w_Clientes	Tel_adicionaralista	
w_Clientes	Salvar	
w_Clientes	Sair	
IncluiTelefoneComercialSemNumero		
w_Principal	Botoes	Clientes
w_Clientes	Prin_listadenomes	FILIPE FERNANDES
w_Clientes	Editar	
w_Clientes	TabClientes	Telefones/FAX
w_Clientes	Tel_comerc	
w_Clientes	Tel_adicionaralista	
w_Alerta	OK	
w_Clientes	Sair	
w_Alerta	Yes	
ConsultaProcessosCliente		
w_Principal	Botoes	Clientes
w_Clientes	Prin_listadenomes	FILIPE SILVA
w_Clientes	TabClientes	Processos

w_Clientes	Sair	
ArquivaCliente		
w_Principal	Botoes	Clientes
w_Clientes	Prin_listadenomes	FILIPE SILVA
w_Clientes	Arquivar	
w_Alerta	Yes	
w_Clientes	Sair	
RestauraCliente		
w_Principal	Botoes	Clientes
w_Clientes	Restaurar	
w_Clientes	listaClientes	FILIPE SILVA
w_ClientesArquivados	Restaurar	
w_Alerta	Yes	
w_Clientes	Yes	
exercita_processos		
InsereProcessoValorInvalido		
w_Principal	Botoes	Processos
w_Processos	Ger_nr	123
w_Processos	Ger_accao	CAUTELAR
w_Processos	Ger_eReu	
w_Processos	Ger_valorCausa	valor
w_Alerta	OK	
InsereProcessoValorValido		
w_Processos	Ger_valorCausa	15000
w_Processos	Salvar	
w_Processos	Sair	
InsereProcessoAccaoTrabalhista		
w_Principal	Botoes	Processos
w_Processos	Novo	

w_Processos	Ger_nr	124
w_Processos	Ger_nrAux	122
w_Processos	Ger_accao	TRABALHISTA
w_Processos	Ger_cliente	FILIPE FERNANDES
w_Processos	Ger_eReu	
w_Processos	Ger_dtEntrada	10052005
w_Processos	Ger_valorCausa	55000
w_Processos	Ger_advogado	FERNANDO CASTRO
w_Processos	Ger_orgaoJulgador	4A VARA
w_Processos	Ger_juiz	NUNO PEIXOTO
w_Processos	Ger_oficialJustica	FRANCISCO CORREIA
w_Processos	Salvar	
w_Processos	Sair	
InsereProcessoRepetidamente		
w_Principal	Botoes	Processos
w_Processos	Novo	
w_Processos	Ger_nr	124
w_Processos	Ger_nrAux	122
w_Processos	Salvar	
w_Alerta	OK	
w_Processos	Sair	
w_Alerta	Yes	
InsereAndamentoProcesso		
w_Principal	Botoes	Processos
w_Processos	Ger_listaprocessos	124
w_Processos	Editar	
w_Processos	TabProcessos	Andamento
w_Processos	And_novo	
w_Andamento	descricao	Atendimento Inicial

w_Andamento	Incluir	
w_Processos	Salvar	
w_Processos	Sair	
AlteraAndamentoProcesso		
w_Principal	Botoes	Processos
w_Processos	Ger_listaprocessos	124
w_Processos	Editar	
w_Processos	TabProcessos	Andamento
w_Processos	And_editar	
w_Andamento	descricao	Acerto Honorarios
w_Andamento	Incluir	
w_Processos	Salvar	
w_Processos	Sair	
InsereAndamentoProcesso		
w_Principal	Botoes	Processos
w_Processos	Ger_listaprocessos	124
w_Processos	Editar	
w_Processos	TabProcessos	Andamento
w_Processos	And_novo	
w_Andamento	descricao	Esclarecimentos sobre o caso
w_Andamento	Incluir	
w_Processos	Salvar	
w_Processos	Sair	
RemoveAndamentoProcesso		
w_Principal	Botoes	Processos
w_Processos	Ger_listaprocessos	124
w_Processos	Editar	
w_Processos	TabProcessos	Andamento
w_Processos	And_excluir	

w_Alerta	Yes	
w_Processos	Salvar	
w_Processos	Sair	
refaz_estado_inicial		
RemoveCliente		
w_Principal	Botoes	Clientes
w_Clientes	Prin_listadenomes	FILIPE SILVA
w_Clientes	Excluir	
w_Alerta	Yes	
w_Clientes	Sair	
RemoveCliente		
w_Principal	Botoes	Clientes
w_Clientes	Prin_listadenomes	FILIPE FERNANDES
w_Clientes	Excluir	
w_Alerta	Yes	
w_Clientes	Sair	
RemoveUltimoCliente		
w_Principal	Botoes	Clientes
w_Clientes	Prin_listadenomes	FILIPE FERNANDES
w_Clientes	Excluir	
w_Alerta	Yes	
w_Clientes	Sair	
w_Alerta	Yes	

B.3 Resultado da Execução dos Testes

Abaixo apresenta-se o resultado da execução automática do ciclo "**regressao"**.

..... START LOGGING RESUME Text Log in StepDriver
SetTextValue adm sent to username.
OK username SetTextValue to adm successful.
SetTextValue admin sent to password.
- WARNING Password Field Detected, not able to verify text for editbox password
OK w_Login:OK CLICKED.
OK w_Principal:Botoes ClickButton of Clientes performed.
SetTextValue FILIPE SILVA sent to Prin_nome.
OK Prin_nome SetTextValue to FILIPE SILVA successful.
SetTextValue Av. Afonso Henriques sent to Prin_morada.
OK Prin_morada SetTextValue to Av. Afonso Henriques successful.
OK w_Clientes:Salvar CLICKED.
OK w_Clientes:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Clientes performed.
OK w_Clientes:Novo CLICKED.
SetTextValue Av. Combatentes sent to Prin_morada.
OK Prin_morada SetTextValue to Av. Combatentes successful.
SetTextValue Nossa Senhora de Fatima sent to Prin_freguesia.
OK Prin_freguesia SetTextValue to Nossa Senhora de Fatima successful.
w_Clientes:Prin_fisica CLICKED.
OK w_Clientes:Salvar CLICKED.
OK w_Alerta:OK CLICKED.
OK w_Clientes:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Clientes performed.
OK w_Clientes:Novo CLICKED.
w_Clientes:Prin_fisica CLICKED.
SetTextValue FILIPE FERNANDES sent to Prin_nome.

OK Prin_nome SetTextValue to FILIPE FERNANDES successful.
SetTextValue Av. Berna sent to Prin_morada.
OK Prin_morada SetTextValue to Av. Berna successful.
SetTextValue Nossa Senhora de Fatima sent to Prin_freguesia.
OK Prin_freguesia SetTextValue to Nossa Senhora de Fatima successful.
SetTextValue Lisboa sent to Prin_distrito.
OK Prin_distrito SetTextValue to Lisboa successful.
SetTextValue 4658531 sent to Prin_bi.
OK Prin_bi SetTextValue to 4658531 successful.
OK w_Clientes:Salvar CLICKED.
OK w_Clientes:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Clientes performed.
OK w_Clientes:Novo CLICKED.
w_Clientes:Prin_juridica CLICKED.
SetTextValue BRUNO GUERREIRO sent to Prin_nome.
OK Prin_nome SetTextValue to BRUNO GUERREIRO successful.
SetTextValue Av. de Roma sent to Prin_morada.
OK Prin_morada SetTextValue to Av. de Roma successful.
SetTextValue Areeiro sent to Prin_freguesia.
OK Prin_freguesia SetTextValue to Areeiro successful.
SetTextValue Lisboa sent to Prin_distrito.
OK Prin_distrito SetTextValue to Lisboa successful.
SetTextValue 12133334343455 sent to Prin_titular_juridica.
OK Prin_titular_juridica SetTextValue to 12133334343455 successful.
OK w_Clientes:Salvar CLICKED.
OK w_Clientes:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Clientes performed.
OK w_Clientes:Editar CLICKED.
OK Dados Adicionais tab successfully selected in TabClientes

SetTextValue Portugues sent to Adic_nacionalidade.
OK Adic_nacionalidade SetTextValue to Portugues successful.
SetTextValue 23041950 sent to Adic_nascimento.
FAILEDEditBox SetTextValue in w_Clientes verification failed. See table AlterarDadosAdicionaisCliente.SDD at line 6
w_Clientes:Adic_nascimento Set to:23041950, but returns: / / .
SetTextValue Engenheiro sent to Adic_profissao.
OK Adic_profissao SetTextValue to Engenheiro successful.
OK w_Clientes:Salvar CLICKED.
OK w_Clientes:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Clientes performed.
OK w_Clientes:Editar CLICKED.
OK Telefones/FAX tab successfully selected in TabClientes
w_Clientes:Tel_fixo CLICKED.
w_Clientes:Tel_residenc CLICKED.
SetTextValue 213456666 sent to Tel_numero.
OK Tel_numero SetTextValue to 213456666 successful.
OK w_Clientes:Tel_adicionaralista CLICKED.
OK w_Clientes:Salvar CLICKED.
OK w_Clientes:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Clientes performed.
OK w_Clientes:Editar CLICKED.
OK Telefones/FAX tab successfully selected in TabClientes
w_Clientes:Tel_comerc CLICKED.
OK w_Clientes:Tel_adicionaralista CLICKED.
OK w_Alerta:OK CLICKED.
OK w_Clientes:Sair CLICKED.
OK w_Alerta:Yes CLICKED.
OK w_Principal:Botoes ClickButton of Clientes performed.
OK Processos tab successfully selected in TabClientes

OK w_Clientes:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Clientes performed.
OK w_Clientes:Arquivar CLICKED.
OK w_Alerta:Yes CLICKED.
OK w_Clientes:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Clientes performed.
OK w_Clientes:Restaurar CLICKED.
OK w_ClientesArquivados:Restaurar CLICKED.
OK w_Alerta:Yes CLICKED.
OK w_Clientes:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Processos performed.
SetTextValue 123 sent to Ger_nr.
OK Ger_nr SetTextValue to 123 successful.
SetTextValue CAUTELAR sent to Ger_accao.
OK Ger_accao SetTextValue to CAUTELAR successful.
w_Processos:Ger_eReu CLICKED.
SetTextValue valor sent to Ger_valorCausa.
OK Ger_valorCausa SetTextValue to valor successful.
OK w_Alerta:OK CLICKED.
SetUnverifiedTextValue 15000 sent to Ger_valorCausa.
OK w_Processos:Salvar CLICKED.
OK w_Processos:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Processos performed.
OK w_Processos:Novo CLICKED.
SetTextValue 124 sent to Ger_nr.
OK Ger_nr SetTextValue to 124 successful.
SetTextValue 122 sent to Ger_nrAux.
OK Ger_nrAux SetTextValue to 122 successful.
SetTextValue TRABALHISTA sent to Ger_accao.

OK Ger_accao SetTextValue to TRABALHISTA successful.
w_Processos:Ger_eReu CLICKED.
SetTextValue 10052005 sent to Ger_dtEntrada.
FAILEDEditBox SetTextValue in w_Processos verification failed. See table InsereProcessoAccaoTrabalhista.SDD at line 8
w_Processos:Ger_dtEntrada Set to:10052005, but returns: / / .
SetUnverifiedTextValue 55000 sent to Ger_valorCausa.
SetTextValue FERNANDO CASTRO sent to Ger_advogado.
OK Ger_advogado SetTextValue to FERNANDO CASTRO successful.
SetTextValue FRANCISCO CORREIA sent to Ger_oficialJustica.
OK Ger_oficialJustica SetTextValue to FRANCISCO CORREIA successful.
OK w_Processos:Salvar CLICKED.
OK w_Processos:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Processos performed.
OK w_Processos:Novo CLICKED.
SetTextValue 124 sent to Ger_nr.
OK Ger_nr SetTextValue to 124 successful.
SetTextValue 122 sent to Ger_nrAux.
OK Ger_nrAux SetTextValue to 122 successful.
OK w_Processos:Salvar CLICKED.
OK w_Alerta:OK CLICKED.
OK w_Processos:Sair CLICKED.
OK w_Alerta:Yes CLICKED.
OK w_Principal:Botoes ClickButton of Processos performed.
OK w_Processos:Editar CLICKED.
OK Andamento tab successfully selected in TabProcessos
OK w_Processos:And_novo CLICKED.
SetTextValue Atendimento Inicial sent to descricao.
OK descricao SetTextValue to Atendimento Inicial successful.
OK w_Andamento:Incluir CLICKED.

OK w_Processos:Salvar CLICKED.
OK w_Processos:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Processos performed.
OK w_Processos:Editar CLICKED.
OK Andamento tab successfully selected in TabProcessos
OK w_Processos:And_editar CLICKED.
SetTextValue Acerto Honorarios sent to descricao.
OK descricao SetTextValue to Acerto Honorarios successful.
OK w_Andamento:Incluir CLICKED.
OK w_Processos:Salvar CLICKED.
OK w_Processos:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Processos performed.
OK w_Processos:Editar CLICKED.
OK Andamento tab successfully selected in TabProcessos
OK w_Processos:And_novo CLICKED.
SetTextValue Esclarecimentos sobre o caso sent to descricao.
OK descricao SetTextValue to Esclarecimentos sobre o caso successful.
OK w_Andamento:Incluir CLICKED.
OK w_Processos:Salvar CLICKED.
OK w_Processos:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Processos performed.
OK w_Processos:Editar CLICKED.
OK Andamento tab successfully selected in TabProcessos
OK w_Processos:And_excluir CLICKED.
OK w_Alerta:Yes CLICKED.
OK w_Processos:Salvar CLICKED.
OK w_Processos:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Clientes performed.
OK w_Clientes:Excluir CLICKED.

OK w_Alerta:Yes CLICKED.
OK w_Clientes:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Clientes performed.
OK w_Clientes:Excluir CLICKED.
OK w_Alerta:Yes CLICKED.
OK w_Clientes:Sair CLICKED.
OK w_Principal:Botoes ClickButton of Clientes performed.
OK w_Clientes:Excluir CLICKED.
OK w_Alerta:Yes CLICKED.
OK w_Clientes:Sair CLICKED.
OK w_Alerta:Yes CLICKED.