Universidade Federal de Santa Catarina

Geração automática de casos de teste automatizados no contexto de uma suite de testes em telefones celulares

Bruno Martins Petroski

Florianópolis – SC

2006/2

Universidade Federal de Santa Catarina

Departamento de Informática e Estatística

Curso de Ciências da Computação

Geração automática de casos de teste automatizados no contexto de uma suite de testes em telefones celulares

Bruno Martins Petroski

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Florianópolis – SC

2006/2

Bruno Martins Petroski

Geração automática de casos de teste automatizados no contexto de uma suite de testes em telefones celulares

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Douglas Nascimento Rechia, M.Sc
Orientador
Universidade Federal de Santa Catarina

Ricardo Pereira e Silva, D.Sc
Co-Orientador
Universidade Federal de Santa Catarina

Vitório Bruno Mazzola, D.Sc
Banca Examinadora
Universidade Federal de Santa Catarina

Otavio Augusto Fuck Pereira, B.Sc
Banca Examinadora
Universidade Federal de Santa Catarina

"*. . . obstacles do not exist to be surrendered to, but only to be broken.*"

– Adolph Hitler

Agradecimentos

Ao mestre Douglas Nascimento Rechia pela orientação, críticas e sugestões. Agradeço também ao co-orientador Ricardo Pereira e Silva e aos membros da banca Vitório Bruno Mazzola e Otavio Augusto Fuck Pereira por suas valiosas contribuições, essenciais ao resultado final deste trabalho.

A todos os colegas do LabSoft com quem tive enormes aprendizados durante o tempo em que trabalhamos juntos. Especialmente para Otavio Pereira e Douglas Rechia por me proporem este excelente desafio e por suas contribuições durante a elaboração deste trabalho.

Agradeço aos meus amigos de curso e a todos os membros da turma 022 de ciências da computação.

A minha namorada Roberta Takeda pela paciência, incentivo e por me fazer uma pessoa cada vez melhor.

Aos meus pais pela educação, apoio, incentivo e ajuda incondicional. Meus eternos tutores na vida pessoal e profissional, sem vocês eu nunca teria conseguido.

Resumo

Colocar o produto no mercado o mais cedo possível é decisivo entre a sobrevivência do produto e sua morte e, conseqüentemente, a sobrevivência e morte da empresa. Atualmente, as empresas estão recorrendo à automatização de testes visando diminuir o ciclo de desenvolvimento de *softwares*. Automação dos casos de teste é um processo muito eficiente, pois reduz o tempo gasto no ciclo de desenvolvimento do *software*, eliminando a necessidade de execução manual destes. Contudo, o processo de gerar casos de teste automatizados não é trivial, e é responsável por grande parte do tempo despendido no processo de automatização do processo de teste. Em contrapartida ao processo de execução automatizada, temos o teste exploratório. Esta técnica de teste de *software* consiste em testar o sistema de uma forma não-sistemática, e pode ser bastante eficaz em identificar casos especiais de teste, os quais não são facilmente capturados por técnicas formais. Os resultados de uma sessão de teste exploratório não são necessariamente totalmente diferentes dos testes de roteiro, e essas duas abordagens de teste são compatíveis e comumente utilizadas no meio corporativo. Este trabalho apresenta um método que possibilita a geração automática de casos de teste automatizados, a partir dos *logs* de uma sessão de teste exploratório e de *logs* provindos da execução automatizada de casos de testes. Tal método combina a eficácia do teste exploratório e a eficiência do teste automatizado, diminuindo assim, o ciclo de desenvolvimento de *software*. Os casos de teste gerados automaticamente podem ser incorporados aos próximos ciclos de execução automatizada, e os erros descobertos na sessão exploratória poderão ser reproduzidos automatizadamente.

Palavras-chave: Teste de *software*, Automação, Raciocínio baseado em casos.

Abstract

Shipping the product as soon as possible is the key point between the product life or death and further, the life and death of the company. Nowadays, companies are appealing to automated testing aiming to reduce the time spent during the software development cycle. Automated test cases turn out to be an efficient process, since it reduces the time spent by eliminating the need to run then manually. However, the automated test case design process is not trivial and it bears a good amount of the time spent during the automation test process. On the order hand, there is the exploratory test. This software test technique consists in testing the system non-systematically, which can improve the error finding effectiveness comparing to formal techniques. The results of exploratory testing aren't necessarily radically different than those of scripted testing, and the two approaches to testing are fully compatible and commonly used within the corporate environment. This thesis presents a method that enables the automatic generation of automated test cases, using as input only the logs of an exploratory session and the logs from an automated test case. Such method aims combining the efficiency of automated software testing and the effectiveness of exploratory testing technique, reducing the software development cycle. The automated test cases generated automatically can be added to future automated test cycle and then, the errors found within the exploratory session can be reproduced automatically.

Keywords: Software testing, Automation, Case-based reasoning.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Sumário

1 Introdução

Atualmente a indústria de desenvolvimento de *software* enfrenta o seguinte paradoxo: os *softwares* estão cada vez mais complexos, dispõem de menos tempo para seu desenvolvimento e existe uma maior exigência em relação à qualidade. Colocar o produto no mercado o mais cedo possível é decisivo entre a sobrevivência do produto e sua morte e, conseqüentemente, a sobrevivência e morte da empresa.

O teste de *software* freqüentemente corresponde por cerca de metade do custo total do desenvolvimento de *software* (CHERNONOZHKIN, 2001). Consiste em uma técnica para avaliar a qualidade do produto e, indiretamente, melhorá-lo – através da identificação de defeitos e falhas. Se conduzido ao acaso, tempo é desperdiçado, esforço desnecessário é despendido e, ainda pior, erros se infiltram sem serem descobertos. Sendo assim, o teste de *software* atualmente é uma atividade crucial no processo de desenvolvimento de *softwares* de alta qualidade e deve ser incluída durante todo o processo de desenvolvimento. Então, seu planejamento deve começar desde a etapa de análise de requisitos e deve ser revisto sistematicamente e continuamente durante todo o processo de desenvolvimento (BOURQUE et al., 2001).

Teste de roteiro[1] é o processo de teste cujos passos são definidos em um documento formal – o caso de teste, e podem ser executados de forma manual ou automatizada. Quando realizado manualmente, é comum que o processo de teste se torne suscetível a erros e altamente custoso, tanto financeiramente e quanto em relação ao consumo de tempo (EICKELMANN; RICHARDSON, 1996). O processo automatizado de teste de *software* vem atacar este problema e, se aplicado corretamente, pode aumentar significativamente a quantidade de testes realizados durante um período de tempo ou reduzir o tempo de cada teste (HICKS; SOUTH; OSHISANWO, 1997). O *Test Automation Framework* (TAF) é um *framework* projetado para suportar a automação de testes funcionais dos *softwares* embutidos em telefones celulares produzidos e desenvolvidos pela Motorola Industrial Ltda (KAWAKAMI et al., 2007 (to appear)). Para Johnson e Foote (1988), *framework* é

[1] Do inglês *scripted test*.

"*um conjunto de classes que contém o projeto abstrato de soluções para uma família de problemas relacionados, e que suporta alto nível de reutilização com alta granularidade*". No TAF os mesmos casos de teste automatizados são reutilizados em diversos modelos que implementam as mesmas funcionalidades, assim a reutilização de código a partir da portabilidade dos testes é maximizada. Automação dos casos de teste é um processo muito eficiente, pois reduz o tempo gasto no ciclo de desenvolvimento do *software* embutido nos telefones, eliminando a necessidade de execução manual destes. Contudo, o processo de gerar casos de teste automatizados não é trivial, e é responsável por grande parte do tempo despendido no processo de automatização do processo de teste de *software*. Gerar casos de teste automatizados automaticamente é uma tarefa ainda mais árdua, pois não existem mecanismos simples e rápidos que o possibilitem.

Em contrapartida ao processo de teste utilizando roteiros, temos o teste exploratório. Teste *ad-hoc*, como também é conhecido, talvez ainda seja a técnica de teste de *software* mais difundida atualmente. Uma sessão de teste exploratório não contém passos pré-definidos ou especificados em algum documento, sendo somente baseado na habilidade e conhecimento do testador sobre o processo e técnicas de teste. Consiste em testar o sistema de uma forma não-sistemática, e pode ser bastante eficaz em identificar casos especiais de teste, os quais não são facilmente capturados por técnicas formais. O TAFLogger é um *software* projetado para auxiliar a execução de testes exploratórios dos *softwares* embutidos em telefones celulares produzidos e desenvolvidos pela Motorola Industrial Ltda. Cada sessão de teste exploratório realizada com o auxílio do TAFLogger produz um *log*, onde todas as ações realizadas sobre o telefone pelo testador são registradas.

Os resultados de uma sessão de teste exploratório não são necessariamente totalmente diferentes dos testes de roteiro, e essas duas abordagens de teste são compatíveis. Empresas como Nortel, Microsoft e Motorola comumente utilizam estas duas abordagens no mesmo projeto (BACH, 2003b). Juntamente com a automatização do testes, a utilização destas duas abordagens em conjunto tem como objetivo combinar a eficácia do teste exploratório juntamente com a eficiência do teste automatizado. Este trabalho propõe um método que possibilita a geração automática de casos de teste automatizados, a partir dos *logs* de uma sessão de teste exploratório utilizando o TAFLogger e de *logs* provindos da execução automatizada de casos de testes do TAF.

Utilizando o método proposto por esse trabalho será possível, por exemplo, re-executar uma sessão de teste exploratório automaticamente. Assim, *bugs* encontrados durante a sessão de teste exploratório poderão ser reproduzidos tanto no telefone em que foi aplicado

o teste quanto em outros modelos de telefone, pois o caso de teste gerado é portável para outros modelos de telefone.

1.1 Objetivos

1.1.1 Objetivo geral

Proporcionar a geração automática de casos de teste automatizados, através da utilização do *log* das atividades de uma aplicação para testes exploratórios em celulares e uma adaptação de um *framework* projetado para suportar a automação de testes dos *softwares* embutidos em telefones celulares.

1.1.2 Objetivos específicos

a) Diminuir o ciclo de desenvolvimento de automação de testes através da diminuição do tempo de criação de casos de teste automatizados.

b) Implementar uma aplicação de raciocínio baseado em casos que analise o *log* da aplicação de um teste *software* exploratório em celulares e gere casos de teste automatizados.

1.2 Metodologia

Um caso de teste é uma seqüência de passos de alto nível que, em sua maioria, tem como objetivo principal encontrar o maior número de erros com o mínimo esforço. Os passos de um caso de teste freqüentemente são reutilizados em outros casos testes, apenas alterando sua seqüência para mudar o enfoque do teste. Então, uma aplicação de raciocínio baseado em casos será proposta, onde os casos são os passos de caso de teste. Um passo é uma seqüência de ações de um nível mais baixo de abstração. No contexto do teste em celulares, as ações de mais baixo nível de abstração seriam, por exemplo, apertar uma tecla ou verificar a tela do telefone.

A seguinte metodologia será adotada a fim de alcançar os objetivos estabelecidos:

a) adaptar o *framework* de automação de testes disponível para que seus *logs* sejam compatíveis com a aplicação de teste exploratórios em celulares;

b) adequar a aplicação de testes exploratórios, para que esta possa ser usada juntamente com o *framework* de automação de testes;

c) analisar os *logs* do *framework* de automação de testes e definir uma representação para os casos do sistema de raciocínio baseado em casos (RBC);

d) prover um método que analise o *log* da aplicação de testes exploratórios e divida-os em seqüências menores que, potencialmente, possuam soluções quando analisados por um sistema de RBC;

e) implementar uma aplicação de RBC que analise uma seqüência de ações de nível mais baixo e retorne o passo de mais alto nível de abstração correspondente e compatível com o *framework* de automação de testes;

f) prover um meio de gerar um caso de teste compatível com o *framework* de automação a partir dos passos de mais alto nível encontrados.

1.3 Justificativa

Cada vez mais gerentes e desenvolvedores de *software* têm que entregar seus produtos no menor tempo e utilizando o mínimo de recursos. Segundo Dustin, Rashka e Paul (1999), mais de 90% dos desenvolvedores já perderam a data de entrega e perder prazos é uma prática comum para 67% dos desenvolvedores. Ainda, 91% deles foram forçados a remover funcionalidades durante o ciclo de desenvolvimento para cumprir prazos. Colocar no mercado o produto o mais cedo possível é decisivo entre a sobrevivência do produto e sua morte – em conseqüência a sobrevivência e morte da empresa. Visando diminuir o ciclo de desenvolvimento de *softwares*, as empresas estão recorrendo a automatização de testes. Automação de testes em *software* pode melhorar em muito sua confiabilidade enquanto diminui significantemente o custo. Então gerar automaticamente casos de teste automatizados diminui o tempo que seria despendido na sua criação, auxiliando as empresas a cumprirem suas metas.

1.4 Resultados esperados

Espera-se que a utilização da aplicação resultante desse trabalho crie automaticamente casos de teste automatizados, ou uma saída equivalente. O processo de criação de casos de teste automatizados deve utilizar-se apenas dos *logs* de casos de testes automatizados

já existentes e de uma sessão de *logs* de uma aplicação de testes exploratórios. Assim o tempo despendido gerando casos de teste automatizados deve ser diminuído, agregando qualidade aos *softwares* embutidos em telefones celulares.

1.5 Estrutura do trabalho

Neste capítulo foi apresentada uma introdução ao trabalho. Essa monografia é composta por mais dois capítulos de fundamentação teórica, um capítulo de desenvolvimento e um capítulo de conclusão.

O Capítulo 2 apresenta os conceitos básicos sobre a metodologia de raciocínio baseado em casos, visto que ela será utilizada para implementar a aplicação.

Seguindo, o Capítulo 3 realizará uma fundamentação teórica sobre: teste de *software*; um *framework* de automação de testes e uma aplicação de teste exploratório em *softwares* embutidos em telefones celulares.

A contribuição deste trabalho é descrito no Capítulo 4, junto com seus resultados obtidos.

Por fim, no Capítulo 5 são mostradas as conclusões constatados por este trabalho.

2 Raciocínio Baseado em Casos

2.1 Introdução

O raciocínio baseado em casos [1] é uma abordagem para a resolução de problemas e aprendizado que tornou-se muito popular nos últimos anos. Mostrou-se bastante útil em uma grande variedade de aplicações, por exemplo: diagnósticos médicos (ALTHOFF et al., 1998; SCHMIDT et al., 2001), aplicações *help–desk* on-line (GÖKER; ROTH-BERGHOFE, 1999; KRIEGSMAN; BARLETTA, 1993), comércio eletrônico (VOLLRATH; WILKE; BERGMANN, 1998), tutor inteligente (MASTERTON, 1997), pesquisa de defeitos em motores de aeronaves a jato (MAGALDI, 1994) e controle militar (LIAO, 2000; GOODMAN, 1989).

O RBC está contido em um ramo da Inteligência Artificial para o aprendizado de máquinas. Suas raízes foram inspiradas por trabalhos advindos da pesquisa sobre a ciência cognitiva, principalmente o papel da memória no entendimento e a teoria da memória dinâmica (MANTARAS et al., 2005). A memória dinâmica procura interagir o entendimento, o aprendizado e a memória, e postula que pacotes de organização de memória (POM) [2] organizam seqüências de eventos individuais. Esses eventos podem ser renomeados como lembranças e participam do processo de interpretação e solução de problemas.

RBC difere da visão tradicional do raciocínio vista tanto na inteligência artificial quanto na psicologia cognitiva, pois, ao invés de tratar um problema e resolvê-lo através da composição e instanciação de operadores abstratos, o novo problema é comparado à problemas antigos recuperados da memória (KOLODNER, 1992). É uma metodologia de gerenciamento de conhecimento para solucionar problemas utilizando ou adaptando soluções de problemas passados (WATSON, 1999).

Segundo Kolodner, raciocínio baseado em casos pode significar adaptar soluções antigas para ajustar-se a novas demandas; utilizar casos antigos para criticar novas soluções; ou raciocinar sobre precedentes para interpretar uma nova solução ou criar uma solução

[1] Do inglês, *Case-Based Reasoning* (CBR).
[2] Do inglês, *Memory Organization Packet* (MOP).

semelhante para o novo problema.

A qualidade das soluções advindas de um raciocinador baseado em casos depende de quatro fatores (KOLODNER, 1992):

- da sua experiência;
- da habilidade de entender novas situações baseadas nas experiências passadas;
- da adequação da adaptação; e
- da adequação da avaliação.

2.2 Raciocínio utilizando casos

Existem dois estilos de raciocínio baseado em casos: resolução de problemas e interpretativo (KOLODNER, 1992). Geralmente o RBC está associado à resolução de problemas, *i.e.*, as soluções para novos problemas são derivadas utilizando as soluções passadas como guia. A metodologia de RBC para a resolução de problemas mostrou-se bastante útil em uma grande variedade de tarefas, incluindo o planejamento, diagnóstico e o projeto. Em cada um deles, casos são úteis na sugestão das soluções e na alerta para possíveis problemas que possam aparecer.

Entretanto, no RBC interpretativo novas situações ou soluções são avaliadas no contexto de uma solução anterior. Nele, a situação ou solução é utilizada como entrada, e a saída é a classificação dessa situação, um argumento que justifica a classificação ou solução, e/ou justificativas que apóiam o argumento ou solução.

Os dois estilos de RBC são altamente dependentes do mecanismo de recuperação de testes. O estilo de resolução de problemas é caracterizado pelo alto uso do processo de adaptação para gerar as soluções e o interpretativo no processo de julgar as soluções derivadas.

2.3 Aprendizado em RBC

A noção de RBC não denota apenas um método específico de raciocínio, independentemente de como os casos são adquiridos, mas também denota o paradigma de aprendizado de máquina que suporta o aprendizado através da atualização da base de casos depois que um determinado problema é solucionado (AAMODT; PLAZA, 1994).

O aprendizado no RBC é derivado naturalmente da resolução do problema, de duas maneiras: lembrando soluções antigas e adaptando-as ao invés de derivar as soluções do princípio toda vez – o que torna o processo mais eficiente, e incorporando novos casos na sua base de casos – tornando mais competente do que quando possui menos casos na base (KOLODNER, 1992). O aprendizado eficiente em RBC exige um conjunto adequado de métodos que extraem conhecimento relevante de experiência, integra o caso na estrutura de conhecimento existente e indexa o caso para depois comparar com casos similares.

2.4 Tipos de métodos de RBC

Existem vários métodos para o raciocínio baseado em casos, e diferem consideravelmente em como cumprir com as principais tarefas de um raciocinador. Os processos em comum entre os raciocinadores que seguem o paradigma RBC suportam uma série de diferentes métodos para organização, recuperação, utilização e indexação do conhecimento retido nos casos passados. Aamodt e Plaza (1994) propõem uma classificação com cinco tipos de raciocinadores baseado em casos:

Raciocínio baseado em exemplares: a solução é um problema de classificação, *i.e.*, encontrar a classe correta para um exemplar ainda não classificado. A definição do conceito de uma classe é definido a partir de um conjunto de seus exemplares.

Raciocínio baseado em instâncias: é a especialização do método acima. Um conjunto relativamente grande de instâncias são necessárias para a definição de um conceito. A representação destas instâncias geralmente é simples, pois o foco principal dessa metodologia é o aprendizado automático sem a interação com o usuário.

Raciocínio baseado em memória: prioriza a coleção de casos como uma grande memória e o raciocínio como o processo de acessar e pesquisar nessa memória. A utilização de processamento paralelo é uma característica diferencial comparada a outros métodos.

Raciocínio baseado em casos: um típico caso geralmente contém um certo grau de riqueza de informação e uma certa complexidade na sua organização interna.

Raciocínio baseado em analogia: é caracterizado por resolver novos problemas baseado em casos passados vindos de um domínio diferente. Tem seu enfoque no estudo da reutilização do caso passado, procurando um modo de transferir ou mapear a solução para o novo problema através de uma analogia identificada.

2.5 O Ciclo de RBC

De modo geral, o RBC é descrito através do ciclo proposto por Aamodt e Plaza (1994) e ilustrado na Figura 1. O ciclo é composto por quatro processos [3]:

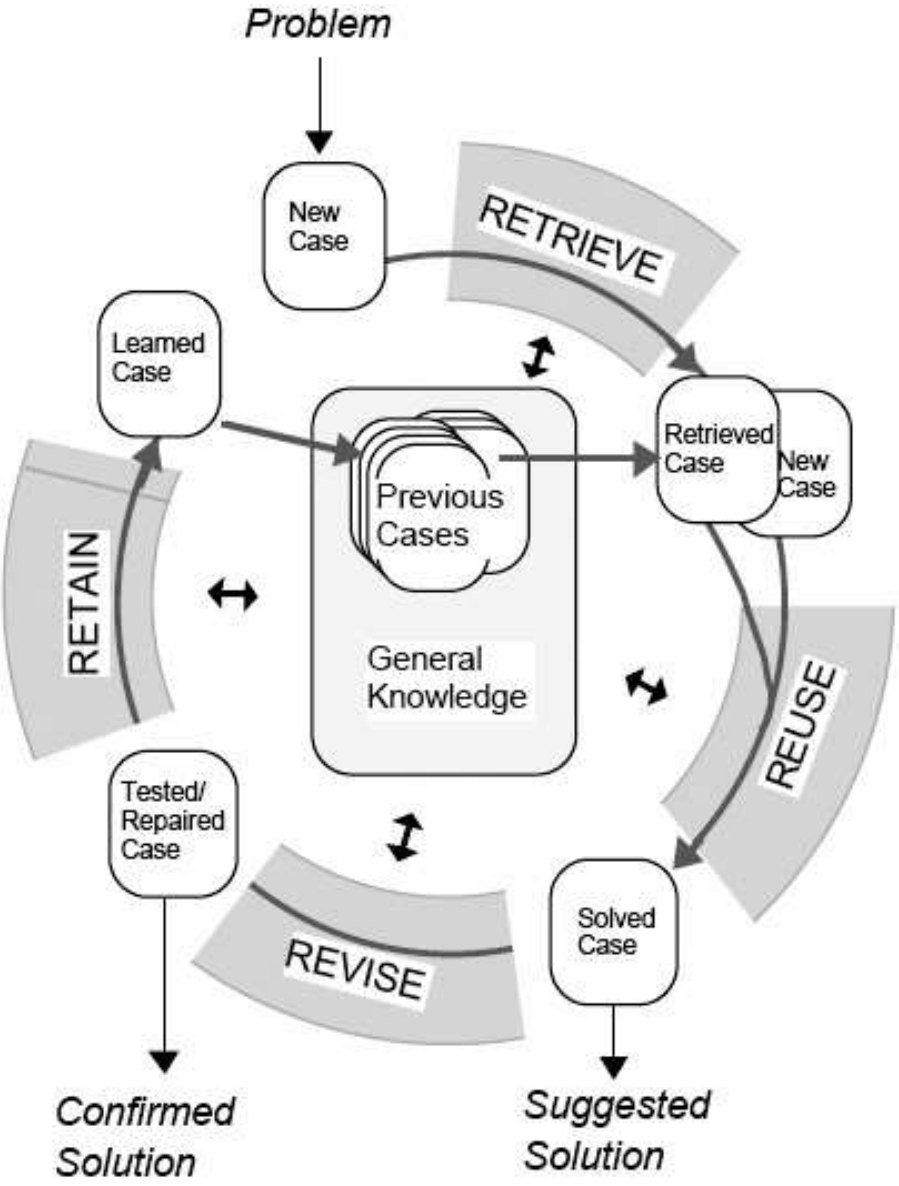

Figura 1: O ciclo do raciocínio baseado em casos segundo (AAMODT; PLAZA, 1994)

1. **recuperar** o(s) caso(s) mais similar(es);

2. **reutilizar** a informação e o conhecimento neste caso(s) para resolver o problema;

3. **revisar** a solução proposta;

[3] 4R como também é conhecido.

4. **reter** as partes desta experiência que potencialmente podem ser úteis na resolução de problemas futuros.

Quando um novo problema é proposto, sua descrição define um novo caso. Um ou mais casos passados similares a este novo são recuperados e reutilizados de alguma forma. A solução é revisada baseando-se na reutilização do caso passado, e por fim essa nova experiência é retida através de sua incorporação em uma base de conhecimento (base de casos) já existente.

2.5.1 Representação e indexação dos casos

2.5.1.1 Representação dos casos

Casos podem representar diferentes tipos de conhecimento e ser armazenado em uma base de casos em vários formatos de representação. Um sistema RBC é altamente dependente da estrutura e do conteúdo dos casos armazenados em sua base. O conteúdo dos casos é determinando pelo domínio de aplicação específico e dos objetivos do raciocinador. O problema de representação de um caso é basicamente o problema da decisão de o quê armazenar, achar uma estrutura apropriada para descrever o seu conteúdo, e decidir qual a organização e indexação do caso na memória para que a recuperação e a reutilização sejam efetivas.

A representação dos casos varia muito. Dentre as formas de representação estruturadas podemos citar as propostas de Kolodner (1992): uma tupla *problema*, *solução* e *resultado*; e de Liao (2000): uma dupla *problema* e *solução*. Contudo, as formas de representação estruturadas não são flexíveis o suficiente para representar conhecimento não-estruturado, então casos podem ser representados na forma de *linguagem natural*, como proposto por Fensel, Angele e Studer (1998). Existem prós e contras para as duas abordagens, uma abordagem semi-estruturada pode ser utilizada para representar casos, como o *Extensible Markup Language* (XML) (HUANG, 2004). As formas de representação de casos mais comuns são: atributo-valor, orientada a objetos, redes semânticas, árvores e grafos.

2.5.1.2 Indexação dos casos

Atribuir índices aos casos para futuras recuperações e comparações é o ponto central da indexação dos casos. Índice nada mais é do que um conjunto de atributos que melhor

identifica um caso. A relevância do atributo no que diz respeito a sua dissimilaridade entre os casos deve ser considerada para atribuir este atributo como parte do índice. A indexação ordena e armazena através do agrupamento dos casos através dos índices. Os índices devem ser abstratos o suficiente para que pouco processamento seja utilizado durante a recuperação e que poucos casos sejam efetivamente recuperados. A escolha dos índices é de suma importância, pois vão determinar em qual contexto os casos vão ser recuperados no futuro.

Métodos manuais e automáticos são utilizados no processo de atribuição dos índices relevantes. O processo manual baseia-se no conhecimento de especialistas sobre domínio de aplicação do raciocinador. Há vários métodos de indexação automática, dentre eles (KOLODNER, 1993 apud WANGENHEIM; WANGENHEIM, 2003):

- utilização de técnicas da estatística exploratória dos atributos que tendem a ser preditivos;
- técnicas indutivas para o aprendizado dos índices através da comparação de casos similares;
- generalização baseada em similaridade e em explicação, que produz um conjunto apropriado de índices para casos abstratos a partir de caso que compartilham um conjunto comum de atributos.

2.5.2 Recuperação

A recuperação dos casos é o processo onde, dado um caso e uma base de casos, retorna o caso ou o conjunto com os casos mais promissores para realizar-se um raciocínio. Esta tarefa pode ser dividida em quatro sub-tarefas realizadas em ordem: assessoramento da similaridade, casamento inicial e procura, e seleção.

2.5.2.1 Assessoramento da similaridade

Tem como objetivo identificar o problema através um conjunto relevante de descritores do problema. Uma abordagem mais elaborada para tentar entender o problema dentro um contexto também pode ser utilizada, se o método de recuperação é realizada através de similaridades semânticas no domínio da aplicação.

2.5.2.2 Casamento inicial e procura

É o processo de recuperar um conjunto de casos candidatos suficientemente similares e possivelmente úteis, e selecionar o melhor entre eles através de uma seleção mais elaborada. Este conjunto de casos é encontrado utilizando as características do problema como índices da base de casos direta ou indiretamente. Os casos podem ser recuperados somente através das características do problema ou por características inferidas do problema.

2.5.2.3 Seleção

A partir do conjunto de casos mais similares, escolhe-se o melhor. O melhor caso é determinado através da avaliação do grau de similaridade ou classificação inicialmente realizado de forma mais detalhada. Se o casamento do caso selecionado demonstra não ser suficientemente bom, uma tentativa de encontrar um caso melhor pode ser realizada através da comparação mais aprofundada das diferenças entre casos do conjunto retornado, o que geralmente é a tarefa mais elaborada da recuperação.

2.5.3 Reutilização

O processo de reutilização no RBC é responsável por propor uma solução para um novo problema a partir das soluções dos casos recuperados, com ou sem adaptação. Esta tarefa possui duas sub-tarefas: copiar e adaptar a solução.

2.5.3.1 Copiar

Simplesmente copiar a solução recuperada sem alterações pode ser uma forma fácil de reutilização. É geralmente utilizada nas tarefas de classificação, onde a solução provavelmente é representada com freqüência na base de casos. Contudo se há diferenças significantes entre o caso atual e o recuperado a reutilização torna-se mais difícil. Defronte a tais circunstâncias, a reutilização da solução deve ser adaptada levando em conta tais diferenças.

2.5.3.2 Adaptar

Devido ao fato de que novas situações raramente são exatamente iguais as antigas, a adaptação se faz necessária para que a antiga solução possa se enquadrar nos novos requisitos. Quais as diferenças entre o caso atual e o recuperado, e quais partes do caso

recuperado podem ser transferidas para o novo caso são as principais dificuldades na adaptação de um caso. Duas estratégias principais são utilizadas para tornar possível a adaptação automática dos casos: reutilizar a solução do caso passado (adaptação transformacional) ou reutilizar a metodologia utilizada para construção da solução do caso passado (adaptação derivacional).

Na adaptação transformacional a solução do caso passado não é diretamente a solução para o novo caso, mas existe algum conhecimento na forma de operadores transformacionais que podem ser aplicados na solução antiga para que ela se torne a solução do novo caso. A adaptação derivacional utiliza informações de como a solução do caso recuperado foi alcançada. O caso recuperado contém informações sobre o método utilizado para resolver o problema do caso recuperado, incluindo justificativas sobre os operadores utilizados, sub-metas consideradas, alternativas geradas, etc. Adaptação derivacional reinstancia a metodologia de solução recuperada para o novo caso e re-executa o plano antigo inserindo-o no novo contexto.

2.5.4 Revisão

Quando a solução gerada pela fase anterior não está correta, uma oportunidade de aprendizado surge a partir dessa falha. É um dos processos mais importantes do raciocinador baseado em casos, pois provê uma oportunidade de avaliação da solução no mundo real, habilitando-o a coletar respostas que possibilitam o aprendizado. O resultado da fase de avaliação pode ser uma adaptação adicional ou a reparação da falha da solução proposta. Esta fase consiste em duas tarefas: avaliar a solução gerada pela fase de reutilização e, caso a solução esteja errada, reparar a falta utilizando o domínio específico da aplicação ou entradas do usuário.

2.5.4.1 Avaliação da solução

A tarefa de avaliação geralmente é realizada fora do escopo da aplicação RBC, pois requer que a solução seja aplicada no ambiente real de aplicação. Alternativamente, o sistema pode simular a aplicação da solução e avaliar sua exatidão. Dependendo do tipo de aplicação, avaliar a solução pode demorar para retornar resultados. O caso ainda pode ser aprendido, e pode estar disponível na base de casos em um período intermediário, mas deve estar assinalado como não-avaliado. Assim que o resultado estiver disponível, proceder-se-á sua avaliação. Se o resultado ocorreu como esperado nenhuma análise futura é necessária, podendo ir direto a fase de retenção (secção 2.5.5).

2.5.4.2 Reparar falhas

Reparar falhas envolve detectar os erros da solução proposta e recuperar ou gerar explicações para elas. As explicações das falhas são utilizadas para modificar a solução para que elas não tornem a ocorrer. A solução revisada pode ser retida diretamente (caso a fase de revisão garanta sua exatidão) ou pode ser avaliada ou reparada novamente.

2.5.5 Retenção

É o processo de incorporação do que é útil reter de um novo episódio de solução do problema no conhecimento existente. Tanto o sucesso quanto a falha da solução proposta conduzem ao aprendizado. Envolve a seleção de qual informação do caso deve ser retido, qual o formato, como indexá-lo para futuras recuperações de casos similares e como integrar o novo caso na estrutura de conhecimento existente. O processo de retenção pode ser refinado em três fases: extração do conhecimento, indexação de casos e integração na base de casos.

2.5.5.1 Extração do conhecimento

Em uma aplicação RBC a base de casos sempre é atualizada, independentemente do método utilizado para a solução do problema. Assim, a decisão que deve ser tomada é sobre o que deve ser usado como fonte da solução. Descrições relevantes do problema e da solução para o problema são os mais óbvios, contudo uma explicação ou outra forma de justificativa do porquê da solução também podem ser incluídos como o novo caso. Outra estrutura onde pode-se também extrair conhecimento é o método de resolução do problema. Falhas podem também ser extraídas e retidas, ou como casos falhos separadamente ou juntamente aos outros casos.

2.5.5.2 Indexação

Indexação é o problema central no raciocínio baseado em casos. Ele implica em decidir que tipo de índices serão utilizados para futuras recuperações, e como estruturar o espaço de busca dos índices. É um problema de aquisição de conhecimento, e deve ser analisado como parte da análise do domínio e modelagem do conhecimento.

2.5.5.3 Integração de casos

É o passo final de atualização da base de conhecimento com o novo conhecimento do caso. Se nenhum caso ou conjunto de índices foi construído, este é o passo principal para a retenção. Através da modificação da indexação dos casos existentes a aplicação RBC torna-se melhor na avaliação da similaridade. Essa melhoria é uma parte importante no aprendizado. Em aplicações de RBC com o uso intensivo de conhecimento, o aprendizado pode ser realizado de forma explícita no contexto do modelo de conhecimento geral do domínio, através de métodos de aprendizado de máquina, ou através da interação com o usuário. O caso recém aprendido finalmente pode ser testado através da re-introdução ao problema inicial e visualizar se o sistema se comporta como esperado.

2.6 Integração com o aprendizado

Muitas aplicações de RBC utilizam domínio geral de conhecimento justamente com a representação do conhecimento dos casos. Representação e uso do domínio de conhecimento envolve integração do método baseado em casos com outros métodos e representações de resolução de problemas. A arquitetura completa de um sistema RBC deve determinar as interações e o regime de controle entre o método RBC e outros componentes. A tarefa de aprender pode ser vista como o processo de melhora da performance de um sistema de RBC, e pode ser utilizado de diversas formas, como: melhora dos repositórios de conhecimento, melhora da medida de similaridade e transformação da metodologia de solução.

3 Teste de Software

Segundo Bourque et al. (2001), teste de *software* é:

> *a verificação dinâmica do comportamento de um programa em um conjunto finito de casos de teste, selecionado apropriadamente de um domínio de execuções comumente infinito, através do comportamento esperado.*

É uma verificação dinâmica do comportamento pois sempre implica em executar o programa com diferentes entradas. Mesmo em programas simples, muitos casos de testes são plausíveis de ser usados, de tal maneira que o teste exaustivo pode levar até anos para ser executado. Portanto, o que essencialmente diferencia as técnicas de teste é o modo como os casos de teste são selecionados. Um critério pode ser adotado para selecionar os casos de teste ou para verificar se uma suite de casos de teste é adequada. Atenção aos critérios de seleção deve ser tomada, pois diferentes critérios podem produzir consideráveis diferenças na eficácia. Deve ser possível decidir também se o comportamento esperado da execução do programa é aceitável ou não, caso contrário o esforço do teste é inútil.

O comportamento observado pode ser verificado, por exemplo, segundo as expectativas do usuário (teste de validação) ou segundo as especificações do sistema (teste de verificação). Testar para identificar defeitos tem seu objetivo cumprido quando levam à uma falha do sistema. O que é bem diferente de testes que têm por objetivo demonstrar que o *software* está de acordo com suas especificações, onde o teste é um sucesso se falhas não são observadas durante sua execução. Uma famosa citação sobre o teste de *software* é a do cientista em computação holandês Edsger Dijkstra: "*teste de software pode ser apenas usado para mostrar a presença de bugs, mas nunca sua ausência*", a qual faz uma alusão ao fato de que testar completamente é inviável em sistemas reais.

3.1 Níveis de Teste

Os níveis de teste podem ser divididos segundo o que está sendo testado e qual o objetivo do teste.

3.1.1 Alvo do teste

Segundo Bourque et al., três estágios podem ser claramente visualizados no que tange o que está sendo testado: unitário, integração e sistema.

3.1.1.1 Teste unitário

Teste unitário verifica o funcionamento de módulos ou unidades de código fonte isolados. Unidades podem ser consideradas as menores partes do software que podem ser testadas. Cada unidade deve ser testada individualmente, pois independe de outras partes do sistema. Em *softwares* orientados a objetos, este nível de teste geralmente é realizado quando uma classe está com seu código maduro, *i.e.* sem erros de compilação, e possivelmente tem o auxílio de ferramentas de depuração.

3.1.1.2 Teste de integração

Nesta fase de teste, módulos individuais (possivelmente já testados isoladamente) são combinados e testados em grupos. Seu principal objetivo é verificar a presença de erros associados às interfaces entre os módulos quando estes são integrados para construir a estrutura do software, conforme descrito nas especificações do sistema e do projeto do programa.

3.1.1.3 Teste de sistema

Consiste em testar o comportamento de todo o sistema, levando em consideração as especificações de requisitos funcionais e de sistema. Além disso, seu enfoque deve ser em atitudes quase destrutivas, como se fosse um usuário final, que testam não somente o projeto mas também o comportamento esperado pelo cliente. O sistema também deve ser testado até e além dos limites definidos nas especificações de requisitos de *software* e *hardware*. E ainda devem ser avaliados neste nível: as interfaces externas com outras aplicações, utilitários, dispositivos de *hardware*, sistemas operacionais, etc.

3.1.2 Objetivos do teste

Diversas propriedades do software podem ser verificadas quando está se testando um sistema, ou parte dele. Casos de teste podem, por exemplo, verificar se as especificações funcionais estão corretamente implementadas. Contudo os aspectos não-funcionais também devem ser verificados, tais como: usabilidade, segurança, performance, qualidade e tolerância à falhas. Os principais objetivos de teste de software podem ser verificados na Tabela 1.

Tabela 1: Principais objetivos de teste de *software*

Nome do teste	**Objetivo**
Aceitação	Permite ao cliente verificar o comportamento do sistema de acordo com seus requisitos, os quais provavelmente foram especificados no contrato.
Instalação	Tem como objetivo garantir que o sistema funcione conforme esperado em diferentes configurações de *hardware* e/ou *software* e sob diferentes condições, *e.g.*, espaço insuficiente em disco e interrupção de energia. O procedimento de instalação também pode ser testado.
Alpha/Beta	Testa o sistema antes de ser liberado. É conduzido por um conjunto potencial de usuários, internos (alpha) ou externos (beta), os quais relatam possíveis problemas na utilização do produto.
Funcional	Verifica se o comportamento observado do sistema sob teste está de acordo com suas especificações de requisitos funcionais.
Regressão	É o re-teste seletivo do sistema ou componentes a fim de verificar que modificações ocorridas não causaram efeitos colaterais inesperados. Os testes de regressão devem ser conduzidos cada vez que uma mudança no sistema é realizada.
Performance	Testa o desempenho do sistema em situações normais e de pico, verificando se está de acordo com os requisitos de performance especificados, *e.g.*, capacidade e tempo de resposta.
Stress	Teste do sistema a fim de determinar o limite máximo de carga e *stress* que o *software* poderá suportar.
Recuperação	Tem como objetivo avaliar o comportamento do *software* após a ocorrência de um erro ou outras condições anormais.

continua na próxima página ...

Tabela 1: Principais objetivos de teste de *software* (continuação)

Nome do teste	**Objetivo**
Configuração	Analisa o sistema em diversas configurações de *hardware* e *software*.
Usabilidade	Avalia a fácil utilização e aprendizado do sistema, garantindo que os requisitos de usabilidade estão sendo cumpridos e aderentes às especificações do usuário.

3.2 Casos de teste e suites

Um caso de teste é um documento de teste que descreve a entrada, ações, eventos e resultados esperados para determinar se uma aplicação está funcionando corretamente ou não. É constituído basicamente por uma série de passos e seus resultados esperados. Casos de teste maiores também podem conter pré-requisitos à sua execução, como um estado ou passos a serem executados. Uma suite de testes é basicamente uma coleção de casos de teste. Usualmente também contém instruções ou objetivos mais detalhados para cada coleção de casos teste. Assim como um caso de teste individual, também pode conter pré-requisitos à sua execução.

3.3 Teste exploratório

Segundo Bach (2003a) teste exploratório é: aprendizado, projeto e execução de testes simultaneamente. Teste exploratório tem como meta utilizar os recursos disponíveis, habilidades e conhecimento da melhor maneira possível, a fim de encontrar o maior número de erros críticos dentro do tempo disponível. É baseado na habilidade e conhecimento do testador sobre o processo e técnicas de teste. Consiste em testar o sistema de uma forma não-sistemática, podendo ser útil para identificar casos especiais de teste, os quais não são facilmente capturados por técnicas formais. Uma sessão de teste exploratório deve:

- ter seus objetivos bem claros: o que deve ser testado, como e quais erros procurar;
- ter uma duração curta (cerca de 60 minutos), mas durante esse período de tempo o testador deve ficar totalmente concentrado na execução de testes;

- ter seus resultados devidamente armazenados, *i.e.*, defeitos, questões relevantes ao processo de teste e eventuais notas.

3.4 Teste automatizado

À medida que os *softwares* se tornam cada vez maiores e mais complexos, fica praticamente inviável de alocar recursos para realizar testes e verificações detalhadas. Automatizar uma ou mais partes do processo de teste de *software* está sendo apontado como uma solução para contornar tal problema. Dentre seus potenciais benefícios estão: substituir o trabalho de testar manualmente, aumentar a repetibilidade e a precisão dos testes, diminuindo os custos de desenvolvimento de *software*. Contudo, não existem somente benefícios acerca da automação do processo e, outras questões devem ser levadas em consideração quando da sua implementação, tais como: custos iniciais podem ser elevados, a abordagem exploratória do teste manual pode ser perdida, além de riscos inerentes ao desenvolvimento de qualquer *software*.

3.5 *Test Automation Framework*

O *Test Automation Framework* (TAF) é um *framework* projetado para suportar a automação de testes funcionais dos *softwares* embutidos em telefones celulares produzidos e desenvolvidos pela Motorola Industrial Ltda (KAWAKAMI et al., 2007 (to appear)). O *framework* foi projetado após observar-se que diversos modelos de telefone celular possuem as mesmas funcionalidades e que uma elevada quantidade de casos de teste em comum são executadas nesses aparelhos (RECHIA, 2005). Os mesmos casos de teste automatizados são reutilizados em diversos modelos que implementam as mesmas funcionalidades, assim a reutilização de código a partir da portabilidade dos testes é maximizada com a utilização do TAF. A automação dos casos de teste reduz o tempo gasto no ciclo de desenvolvimento do *software* embutido nos telefones, pois elimina a necessidade de execução manual destes.

Toda a interação entre o TAF e o telefone é realizada através da utilização de uma biblioteca de funções proprietária chamada de *Phone Test Framework* (PTF). O PTF comunica-se com o telefone via interface USB (*Universal Serial Bus*) e provê funções que, por exemplo, retornam o conteúdo que está sendo mostrado na tela e que simulam o pressionamento de teclas do telefone (ESIPCHUK; VALIDOV, 2006). Contudo a utilização direta do PTF para automatizar um caso de teste produz *scripts* de teste ilegíveis e difíceis

de serem portados para serem executados em outro modelo de telefone, pois as funções providas pelo PTF são de baixo nível de abstração.

3.5.1 Visão geral da arquitetura

A fim de aumentar o nível de abstração das funções providas pelo PTF, o TAF foi projetado. No TAF, chamadas diretas à biblioteca de funções do PTF são encapsuladas em *Utility Functions* (UFs). Uma UF é a implementação de um passo de alto nível que é executado em um caso de teste. Sendo assim os casos de teste automatizados são escritos em termos de UFs de alto nível de abstração, promovendo a reutilização de código. Alguns exemplos de UFs que representam um passo de alto nível em um caso de teste são: *ComposeMessage* (compõe uma mensagem de texto ou de multimídia), *LaunchApp* (abre uma aplicação qualquer, como o editor de mensagens ou a lista de contatos) e *DeleteAllPictures* (remove todas as figuras do telefone).

A interface *Step* é implementada por todas as UFs no TAF. Segundo Rechia (2005):

> Sob certo ponto de vista, um caso de teste é uma lista de *Steps*; tudo o que um caso de teste faz é invocar o método *execute* que é definido na interface *Step* e implementado nas UFs concretas. Cada UF, entretanto, além de implementar a interface *Step*, possui uma API (*Application Programming Interface*) bem definida que provê métodos que possibilitam a interação com esta UF.

Um exemplo de API pode ser verificado na UF *LaunchApp*, que possui o método *setApplication* que indica à *LaunchApp* qual aplicação deverá ser iniciada quando esta UF for executada. Um diagrama de classes simplificado do TAF é ilustrado na Figura 2, onde detalhes irrelevantes são omitidos.

Quando um caso de teste existente e funcionando é portado para outro modelo de telefone, ocorre a reutilização de código. Se uma implementação de UF não funciona para o modelo que está sendo portado, uma implementação específica necessita ser criada. A Figura 2 mostra que uma implementação específica da UF *ComposeMessage* para o telefone fictício XYZ foi criada, visto que o comportamento desse modelo para compor uma mensagem é diferente dos demais. Assim, toda UF é um potencial *hot spot*. Na prática, a API de uma UF é uma classe abstrata que implementa a interface *Step*. As implementações de uma UF são classes concretas que estendem a API desta UF. Desta

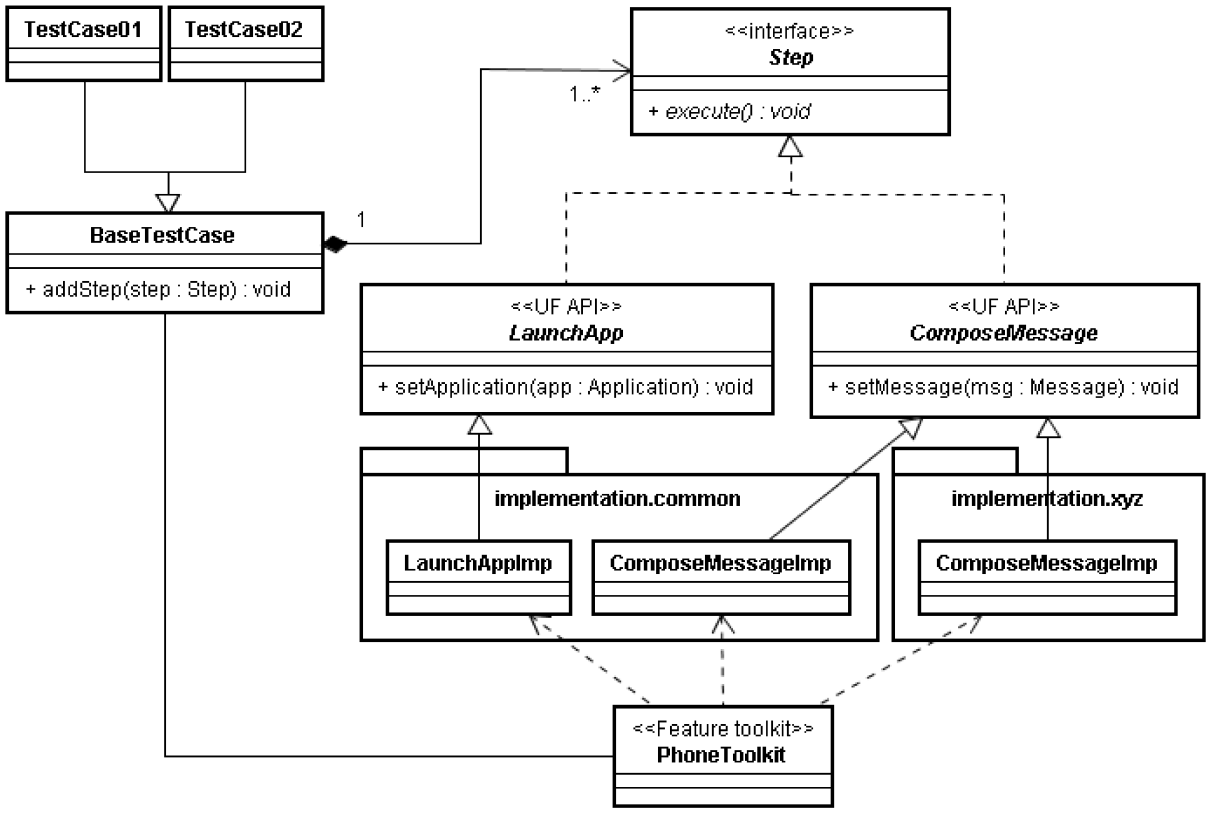

Figura 2: Diagrama simplificado de classes do TAF

forma, consegue-se reutilizar um caso de teste reimplementando somente as UFs que ainda não funcionam no telefone para o qual o caso de teste está sendo portado.

Um caso de teste é composto por várias chamadas a métodos dos *feature toolkits*, os quais adicionam um *Step* na lista de *Steps* a serem rodados pelo caso de teste. A responsabilidade de um *feature toolkit* resume-se em criar a instância de uma UF específica, passar a ela os argumentos necessários, que foram definidos na API da UF, e acrescentar a instância dessa UF na lista de *Steps* do caso de teste. Após a lista de *Steps* ser construída através dos *feature toolkits*, o teste poderá ser executado. A Figura 3 mostra a arquitetura em camadas utilizada pelo TAF, onde se visualiza a relação entre a camada onde estão os casos de teste, a camada onde os *feature toolkits* são implementados, a camada das UFs (API + implementação), e o PTF.

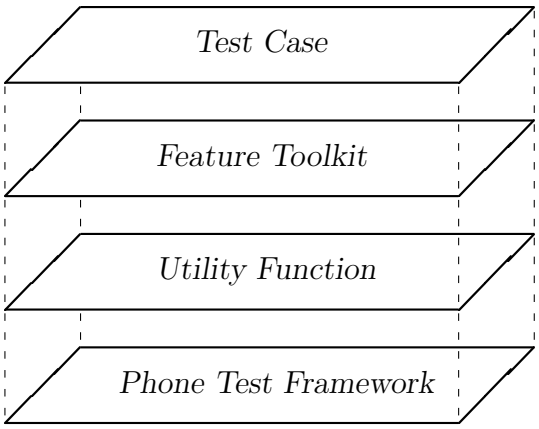

Figura 3: Arquitetura de camadas do TAF

3.5.2 Controle do estado do telefone

Cada execução de uma UF potencialmente produz uma mudança no estado do telefone. Antes da UF ser executada o telefone encontra-se no estado inicial, a UF é executada e, após sua execução, o telefone está no estado final. A Figura 4 demonstra de forma esquemática esta a transição de estados no telefone provocada por uma UF. Contudo, nem todas as UFs mudam o estado do telefone. Em geral, são UFs responsáveis por realizar verificações do estado do telefone, do tipo: garantir que o telefone encontra-se em alguma aplicação ou que existem um número de mensagens não lidas na caixa de entrada. Cada UF possui uma pré-condição para que sua execução seja possível, a qual geralmente está vinculada ao estado em que o telefone se encontra. E, após o término da execução da UF, o telefone estará no estado especificado pela pós-condição desta UF.

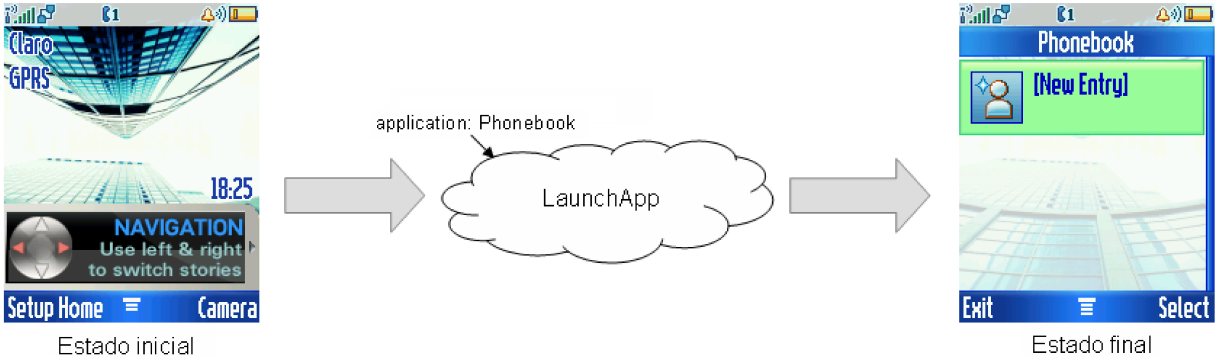

Figura 4: Transição esquemática de estados no telefone provocada por uma UF

Mudanças no estado do telefone são geralmente vinculadas a estímulos no teclado do telefone. Assim, cada vez que uma tecla do telefone é estimulada, em mais baixo nível pelo PTF, o telefone transiciona para um novo estado. Comportamento análogo à um autômato finito, como demonstrado na Figura 5.

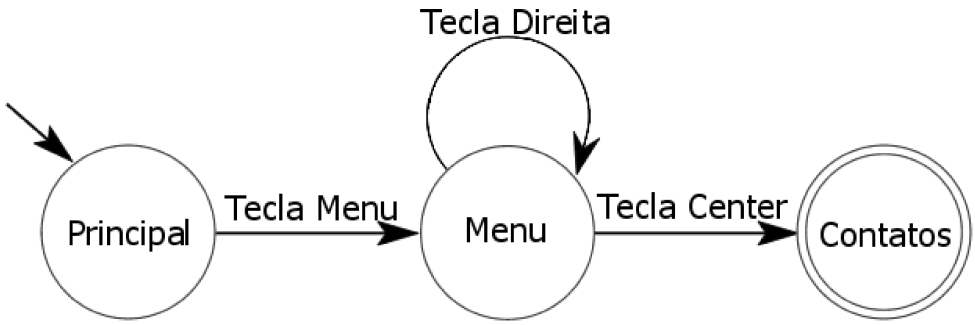

Figura 5: Analogia do telefone funcionando como um autômato finito

3.6 TAFLogger

O TAFLogger é um *software* projetado para auxiliar a execução de testes exploratórios dos *softwares* embutidos em telefones celulares produzidos e desenvolvidos pela Motorola. Nas sessões de teste exploratório realizadas com o auxílio do TAFLogger, todas as ações realizadas sobre o telefone pelo testador são registradas. Dentre os principais registros de uma sessão de testes do TAFLogger estão: horários de início e fim da sessão, notas de texto realizados pelo testador (incluindo comentários genéricos, possíveis problemas no telefone e *bugs*), teclas pressionadas e estados do telefone.

Assim como o TAF, toda a interação entre TAFLogger e o telefone é realizada através

da utilização do PTF via interface USB (ESIPCHUK; VALIDOV, 2006). Mas, ao contrário do que era feito no TAF, a função do PTF no TAFLogger é somente reportar eventos que acontecem no telefone, *e.g.* o pressionamento de teclas, e não estimular o telefone de alguma maneira. Isso deve-se ao fato de que todas as ações realizadas sobre o telefone são oriundas do testador durante a sessão.

O registro de uma sessão exploratória do TAFLogger é armazenado em um arquivo de formato XML (*Extensible Markup Language*). Uma versão simplificada da saída do TAFLogger pode ser verificada abaixo[1]. Note que toda a sessão de teste fica agrupada na tag `SESSION` (linha 2). Na linha 5 ocorre o primeiro evento da sessão: um `LogSessionStateEvent` com o valor de `LogSessionLogging`, que indica que a sessão está começando. Começada a sessão, o primeiro evento capturado pelo TAFLogger proveniente do telefone (previamente conectado), foi um pressionamento de tecla (`KeyPressEvent`) na linha 9. O `KeyPressEvent` indica que a tecla `END` foi pressionada por 172 milisegundos. Os próximos eventos capturados foram obtidos a partir de novos estados do telefone e estão localizados nas linhas 14 e 20. O segundo evento (`PhoneScreenEvent`) diz que uma nova tela do tipo `ICONICMENU` com 124 *pixels* de largura e 166 *pixels* de altura foi capturada. E assim por diante até o último evento, que indica que a sessão terminou, na linha 59.

Arquivo XML de uma sessão do TAFLogger

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <SESSION>
3  <GROUP description="0701231535&#95;Sess&#227;o&#95;Launch&#32;Phonebook">
4     <DATE>1169573751062</DATE>
5     <EVENT type="com.motorola.taflogger.events.LogSessionStateEvent" phoneID="-1" >
6        <DATE>1169574116093</DATE>
7        <VALUE>com.motorola.taflogger.logmanagement.LogSessionLogging</VALUE>
8     </EVENT>
9     <EVENT type="com.motorola.taflogger.events.KeyPressEvent" phoneID="0" >
10       <DATE>1169574116093</DATE>
11       <VALUE>END</VALUE>
12       <TIME_HELD>172</TIME_HELD>
13    </EVENT>
14    <EVENT type="com.motorola.taflogger.events.PhoneScreenEvent" phoneID="0" >
15       <DATE>1169574116218</DATE>
16       <VALUE>
17          <DISPLAY width="124" height="166" type="ScreenType.IDLE"/>
18       </VALUE>
19    </EVENT>
20    <EVENT type="com.motorola.taflogger.events.PhoneScreenEvent" phoneID="0" >
21       <DATE>1169574122421</DATE>
22       <VALUE>
```

[1] Apesar do TAFLogger registrar todos os ítens que compõem a tela do telefone, eles foram omitidos por restrições espaciais.

```
23          <DISPLAY width="124" height="166" type="ScreenType.ICONICMENU"/>
24       </VALUE>
25    </EVENT>
26    <EVENT type="com.motorola.taflogger.events.KeyPressEvent" phoneID="0" >
27       <DATE>1169574122453</DATE>
28       <VALUE>CENTER</VALUE>
29       <TIME_HELD>282</TIME_HELD>
30    </EVENT>
31    <EVENT type="com.motorola.taflogger.events.PhoneScreenEvent" phoneID="0" >
32       <DATE>1169574124953</DATE>
33       <VALUE>
34          <DISPLAY width="124" height="166" type="ScreenType.ICONICMENU"/>
35       </VALUE>
36    </EVENT>
37    <EVENT type="com.motorola.taflogger.events.KeyPressEvent" phoneID="0" >
38       <DATE>1169574125109</DATE>
39       <VALUE>LEFT</VALUE>
40       <TIME_HELD>266</TIME_HELD>
41    </EVENT>
42    <EVENT type="com.motorola.taflogger.events.PhoneScreenEvent" phoneID="0" >
43       <DATE>1169574125281</DATE>
44       <VALUE>
45          <DISPLAY width="124" height="166" type="ScreenType.ICONICMENU"/>
46       </VALUE>
47    </EVENT>
48    <EVENT type="com.motorola.taflogger.events.KeyPressEvent" phoneID="0" >
49       <DATE>1169574127875</DATE>
50       <VALUE>CENTER</VALUE>
51       <TIME_HELD>204</TIME_HELD>
52    </EVENT>
53    <EVENT type="com.motorola.taflogger.events.PhoneScreenEvent" phoneID="0" >
54       <DATE>1169574127875</DATE>
55       <VALUE>
56          <DISPLAY width="124" height="166" type="ScreenType.LIST_ENHANCED_SEARCHLIST"/>
57       </VALUE>
58    </EVENT>
59    <EVENT type="com.motorola.taflogger.events.LogSessionStateEvent" phoneID="-1" >
60       <DATE>1169574134203</DATE>
61       <VALUE>com.motorola.taflogger.logmanagement.LogSessionStopped</VALUE>
62    </EVENT>
63 </GROUP>
64 </SESSION>
```

3.7 Correlação entre teste de *software*, TAF e TAFLogger

Ambos, TAF e TAFLogger, tem como objetivo auxiliar o processo de teste dos *softwares* embutidos em telefones celulares produzidos e desenvolvidos pela Motorola. O TAF é um *framework* que automatiza os testes funcionais e de regressão nos telefones celulares.

Já o TAFLogger é responsável por auxiliar o processo manual de testes exploratórios. No próximo capítulo (4) será demostrado uma aplicação que tem como objetivo gerar casos de teste automatizados a partir de logs dos testes exploratórios e dos logs dos testes automatizados do TAF.

4 Geração de Casos de Teste Automatizados a Partir de Testes Exploratórios

4.1 Visão geral

O processo de geração de casos de teste automatizados a partir de testes exploratórios é ilustrado na Figura 6. Cada execução de um caso de teste automatizado gera um log que é armazenado. Posteriormente todos os logs gerados das execuções automatizadas são analisados à procura de chamadas de UF. Cada chamada de UF é então convertida à um caso do raciocinador baseado em casos, e serve para alimentar sua base de conhecimento. Uma sessão de teste exploratório qualquer utilizando o TAFLogger serve como entrada para o sistema implementado. Tal sessão é dividida em potenciais problemas (UFs) que são a entrada do sistema RBC. Cada problema é então submetido ao ciclo do raciocinador que tem por objetivo encontrar uma solução (UF correspondente à esta parte da sessão). Após todas as partes da sessão serem analisadas à procura por UFs correspondentes, um novo caso de teste compatível com o TAF é criado. A lista de passos (*Steps*) do novo caso de teste é montada a partir das UFs encontradas pelo raciocinador. Este novo caso de teste pode ser incluído em alguma suite do TAF e ser executado de forma automatizada.

4.2 Tecnologias utilizadas

A implementação deste trabalho deu-se utilizando a linguagem de programação orientada a objetos Java. A utilização desta linguagem deu-se principalmente por questões de compatibilidade entre o *framework* TAF e a aplicação TAFLogger, ambos implementados em Java versão 1.4. Além disso, outras características da linguagem também impactaram na sua escolha, tais como:

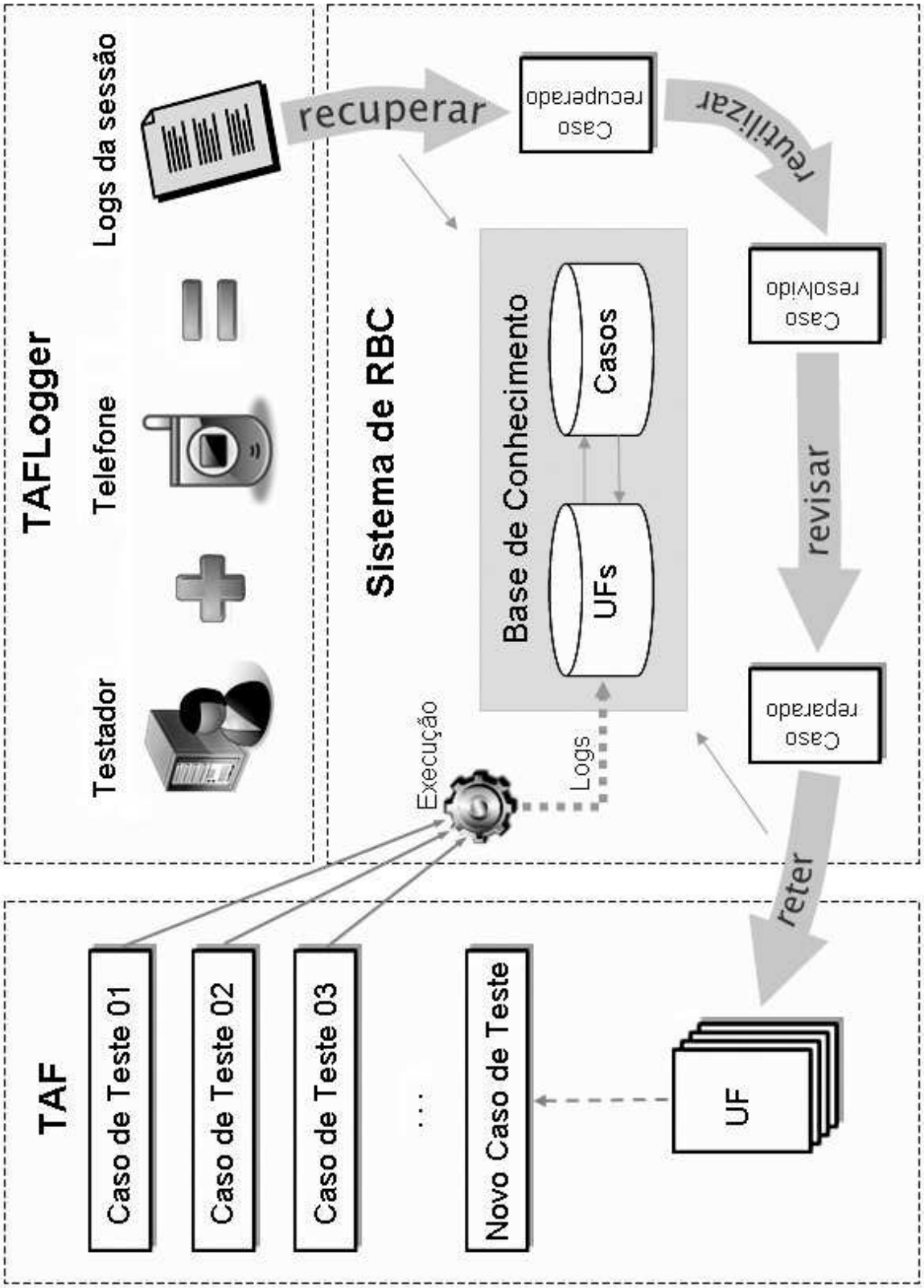

Figura 6: Visão geral do processo de geração de casos de teste automatizados a partir de testes exploratórios

- linguagem simples, eficiente, segura, robusta e portável;
- independente de plataforma;
- suporte a *multithreading*;
- eficiente mecanismo de gerenciamento de memória;
- manipulação de exceções eficiente.

Outra tecnologia utilizada foi o XML (*Extensible Markup Language*). XML é uma linguagem de marcação de propósito geral e foi originalmente concebida com o intuito de solucionar problemas concernentes à larga escala de publicações eletrônicas, incluindo a troca de dados via Web ou em qualquer outro meio. O XML provê meios de descrever e aplicar informações textuais em forma de uma estrutura baseada em árvores.

4.3 Arquitetura e descrição da implementação

4.3.1 Adaptação dos *frameworks* e aplicações disponíveis

Para que o TAF coletasse os logs de forma aproveitável, foram necessárias algumas modificações. A principal delas foi utilizar o próprio TAFLogger para coletar os eventos que o *framework* realiza sobre os telefones que estão sendo testados. A utilização do TAFLogger foi implementada utilizando o padrão de projeto *Façade*. Este padrão de projeto visa promover um baixo acoplamento entre o TAF e o TAFLogger. Isso acontece pois a comunicação e as dependências entre os dois sistemas é diminuída, tornando-os mais fácil de utilizar.

A adaptação necessária ao TAFLogger foi mínima: apenas a adesão de parte do sistema à outro padrão de projeto, o MVC (Modelo-Visão-Controlador [1]). Para tal adequação, a separação da interface gráfica com o usuário e a parte lógica foi implementada em determinadas partes do código fonte.

4.3.2 Aplicação da metodologia de RBC

4.3.2.1 Representação do caso do RBC

Conforme descrito na Seção 2.5.1.1, existem diversas formas de representar casos. O formato adotado foi o de uma dupla problema e solução. O problema é a seqüência de

[1] Em inglês, *Model-View-Controller*.

estados, teclas estimuladas e UFs chamadas durante a execução de uma UF, modelada como um autômato finito. A solução é justamente a UF que originou tal seqüência, juntamente com os valores dos parâmetros definidos em sua API. Formalmente um autômato finito é representado por uma 5-tupla (Q, Σ, δ, $s0$, F), onde: Q é o conjunto finito de estados; Σ é o conjunto finito de símbolos (alfabeto); δ é a função de transição; $s0$ é o estado inicial; e F é o conjunto de estados aceitos (ou finais). Portanto, o autômato finito modelado pelo problema é o seguinte:

Q – conjunto finito de estados (telas) capturadas e de UFs invocadas;

Σ – conjunto de teclas estimuladas durante a execução de uma UF;

δ – tabela que representa todas as transições realizadas pela UF;

s0 – é o estado em que o telefone se encontrava quando a UF foi chamada;

F – é o conjunto de estados composto apenas pelo estado em que o telefone se encontrava quando terminou a execução dessa da UF.

4.3.2.2 Base de casos e indexação

Definida a representação de um caso, um recipiente para armazená-los foi modelado – a base de casos. A base de casos nada mais é do que um repositório indexado de casos, como é possível verificar no diagrama de classes ilustrado pela Figura 7. Os índices de um caso são combinações de seus atributos mais importantes, que permitem distinguí-lo de outros. No caso modelado, os atributos mais importantes são: conjunto de estados capturados, UFs invocadas, e o conjunto de teclas estimuladas.

A estrutura de indexação adotada foi uma árvore *k-d*. Uma árvore *k-d* é uma árvore de pesquisa binária *k*-dimensional, que decompõe a base de casos iterativamente em partes menores (WANGENHEIM; WANGENHEIM, 2003). Em cada nível desse tipo de árvore utiliza-se uma chave diferente dentre suas k chaves (equivalente ao número de atributos indexáveis). Cada nodo da árvore *k-d* representa um subconjunto dos casos na base de casos, e a raiz representa toda a base de casos. As folhas da árvore contém um subconjunto específico da base de casos.

A Figura 8 mostra a estrutura de indexação com alguns índices e casos a título de exemplificação. No nível 0 está a base de casos. Logo abaixo, no nível 1, estão os grupos de UFs (`uf_grp_1`, `uf_grp_2`, `none_uf_grp`, etc.). Os casos que não invocam nenhuma UF

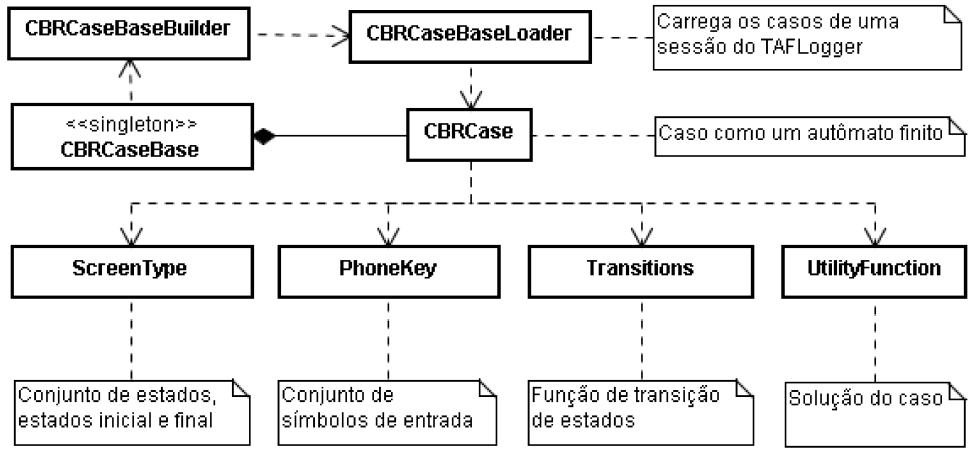

Figura 7: Diagrama de classes da base de casos

são agrupados no grupo `none_uf_grp` [2]. No nível 2 da árvore, estão os índices relativos aos grupos de estados capturados (`scrn_grp_1`, `scrn_grp_2`, `scrn_grp_n`, etc.). No nível imediatamente abaixo (3) os casos ficam agrupados segundo às teclas estimuladas. Os nodos ilustrados na Figura 8 que estão nesse nível são: (`key_grp_1`, `key_grp_2`, `key_grp_n`, etc.). Finalmente, os casos, que são as folhas da árvore, ficam armazenados no nível 4. Dentre as folhas estão os casos `case_1`, `case_2`, `case_n`, etc.

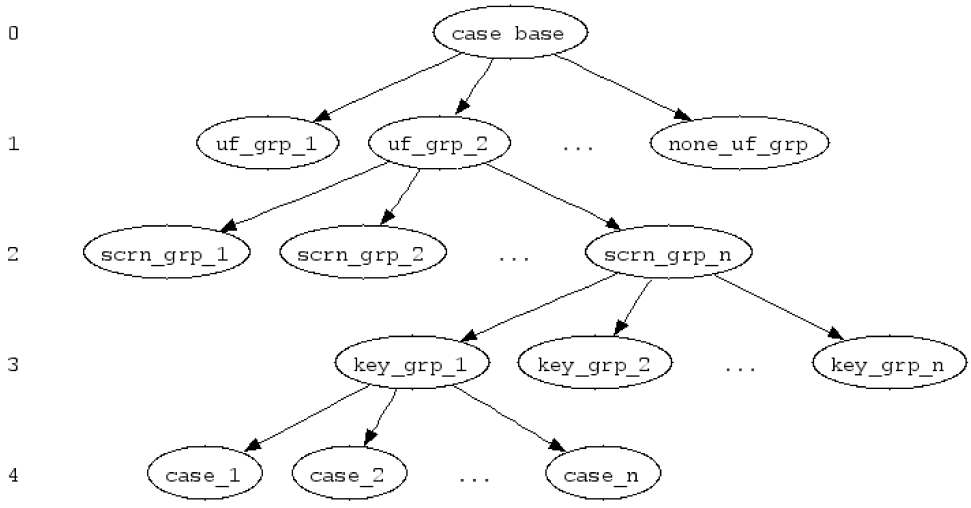

Figura 8: Estrutura de índices da base de casos

4.3.2.3 Implementação do ciclo de RBC

Para implementar o ciclo 4R da metodologia de RBC, foi definido um esqueleto do algoritmo na classe `CBRCycle` utilizando o padrão de projeto *template method*. Cada

[2]Em geral, os casos que não invocam nenhuma UF correspondem às UFs de mais baixo nível.

um dos 4 processos do ciclo é delegado à outras classes, que são responsáveis pelos passos específicos de seu processo (`CBRCycleRetrieve`, `CBRCycleReuse`, `CBRCycleRevise`, `CBRCycleRetain`). O diagrama de classes do ciclo 4R do sistema de RBC é apresentado na Figura 9. Observa-se no diagrama de classes que um objeto da classe `CBRCyleContext` é passado para todos os processos. Esse objeto é responsável por armazenar o contexto durante os quatro processos do ciclo. Já o diagrama de seqüência do ciclo 4R do sistema de RBC pode ser visualizado na Figura 10, e a seguir será mostrada a implementação de cada um dos processos separadamente.

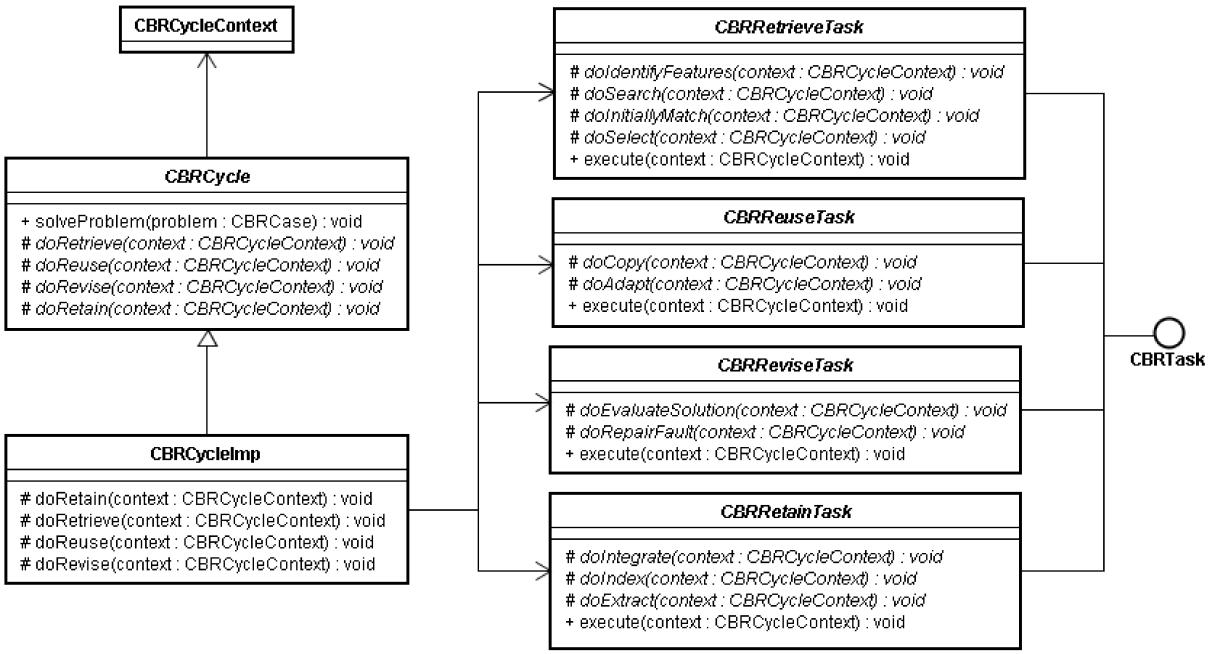

Figura 9: Diagrama de classes geral do ciclo RBC

Recuperação Nesse processo, o caso mais promissor para realizar-se o raciocínio é retornado. Sua implementação também utilizou o padrão de projeto *template method* para definir o esqueleto do algoritmo de recuperação. Este padrão de projeto foi utilizado conjuntamente com outro, o *strategy*. O *strategy* define uma família de algoritmos, encapsula-os, e os tornas cambiáveis entre si. Assim, o algoritmo varia independentemente dos clientes que o usam. Contudo, apesar do processo ser projetado de forma flexível e que suporte várias implementações de um mesmo algoritmo, apenas uma implementação de cada método foi realizada. A Figura 11 ilustra o diagrama de classes, enquanto a Figura 12 ilustra o diagrama de seqüência.

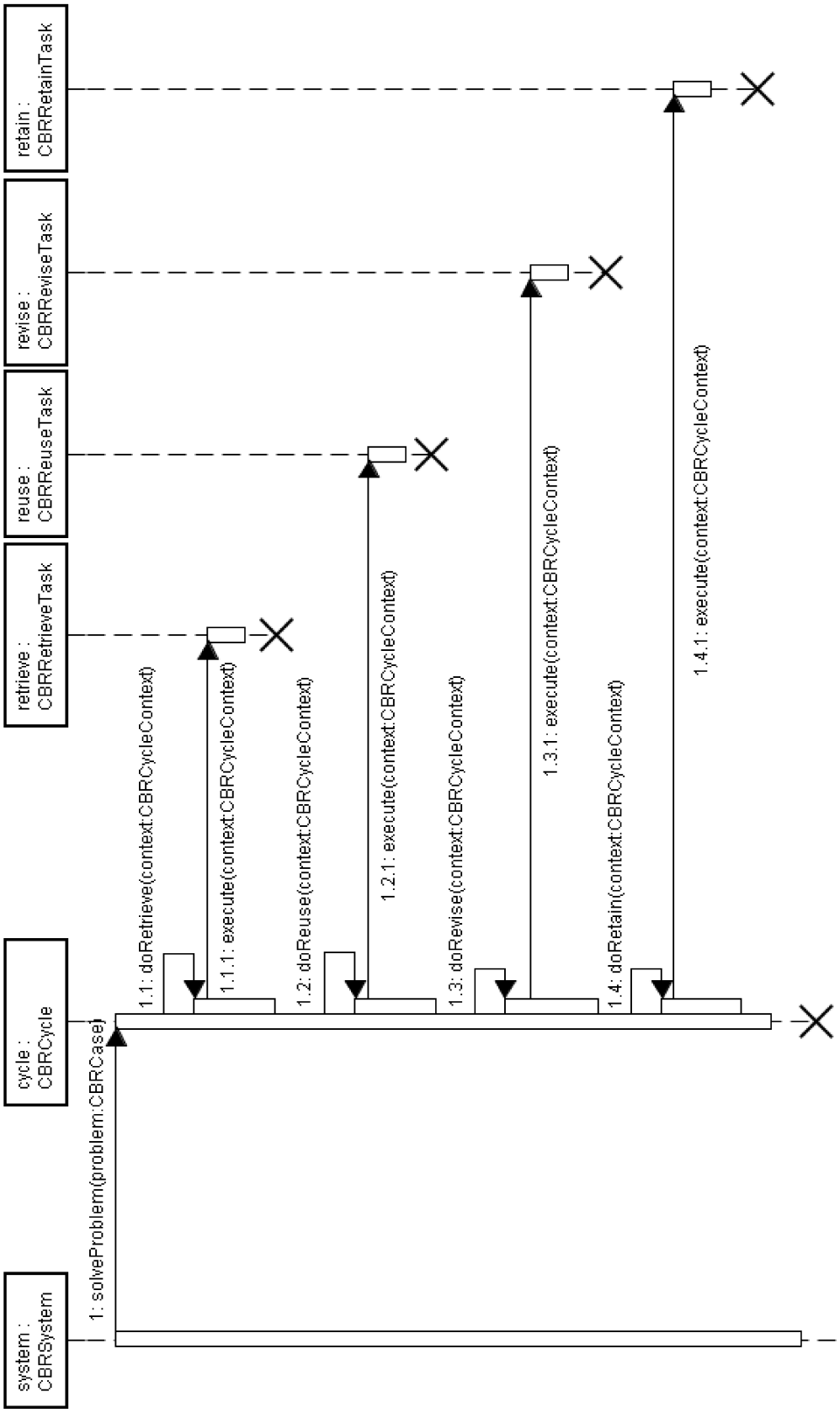

Figura 10: Diagrama de seqüência geral do ciclo RBC

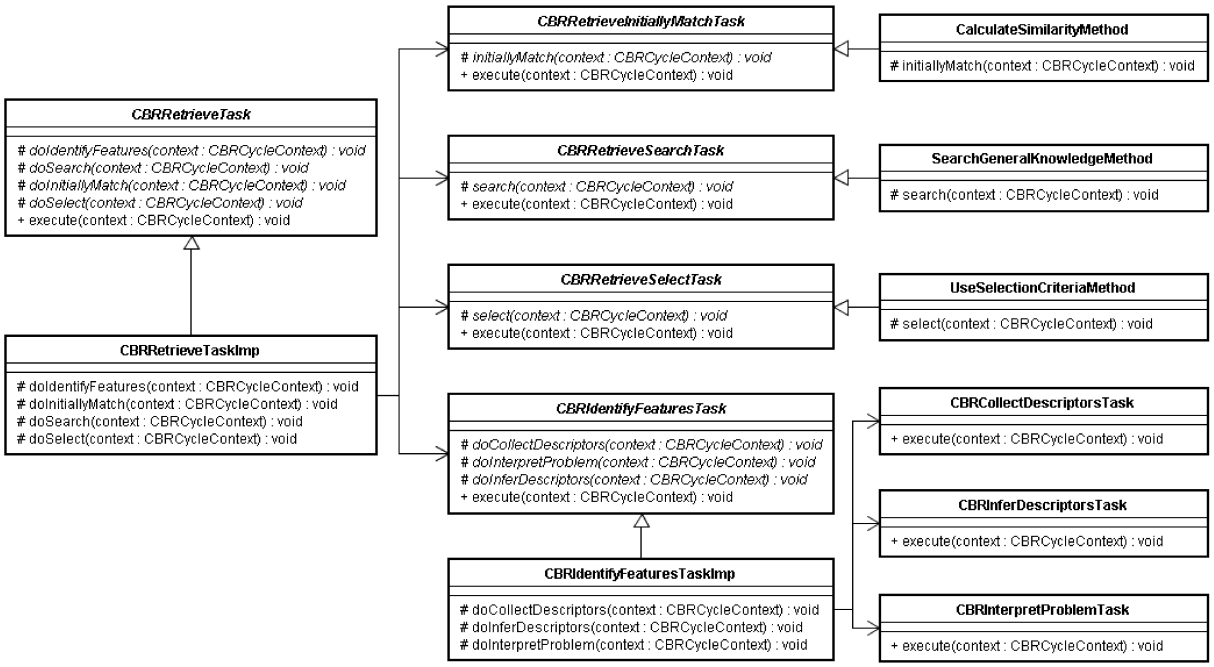

Figura 11: Diagrama de classes do processo de recuperação do ciclo RBC

Reutilização Encontrado o caso mais promissor, uma solução para o novo problema é proposta a partir das soluções dos casos recuperados, com ou sem adaptação. A reutilização do caso mais promissor foi implementada de maneira semelhante ao processo anterior. A Figura 13 ilustra o diagrama de classes, enquanto a Figura 14 ilustra o diagrama de seqüência deste processo.

Revisão Nessa etapa, a solução gerada pelo processo anterior deve ser avaliada e, caso a solução esteja errada, é necessário reparar a falta utilizando o domínio específico da aplicação ou entradas do usuário. Assim como as fases anteriores, essa parte do processo foi projetado utilizando os padrões de projetos *template method* em conjunto com o *strategy*, como pode ser verificado no diagrama de classes apresentado na Figura 15. O diagrama de seqüência é ilustrado na Figura 16.

Retenção A última fase do ciclo de RCB é a retenção. Esta fase é responsável por decidir o que é útil reter da nova solução na base de conhecimento existente. Este processo foi implementado de forma análoga às fases anteriores e seus diagramas de classe e de seqüência podem ser visualizados nas figuras 17 e 18, respectivamente.

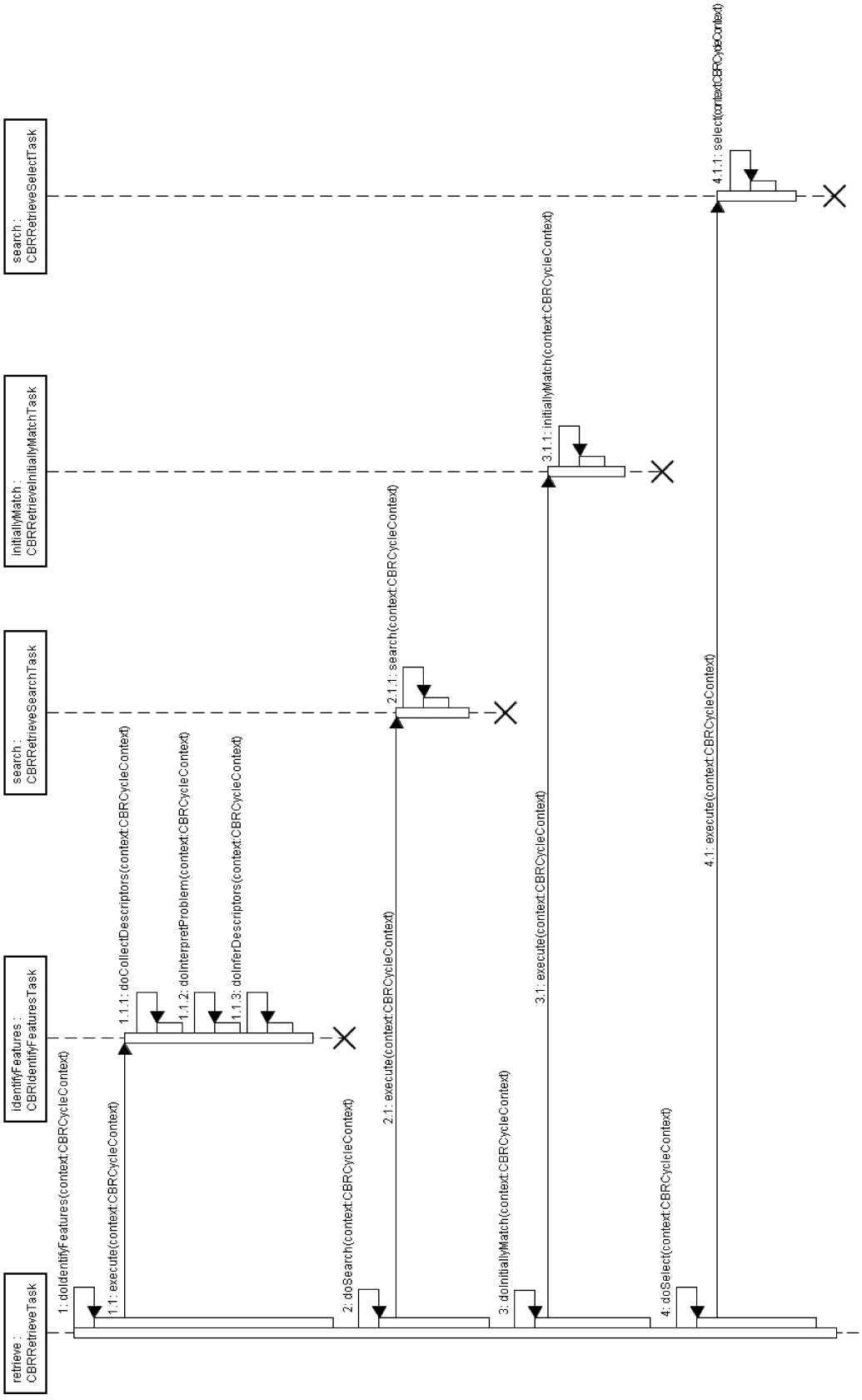

Figura 12: Diagrama de seqüência do processo de recuperação do ciclo RBC

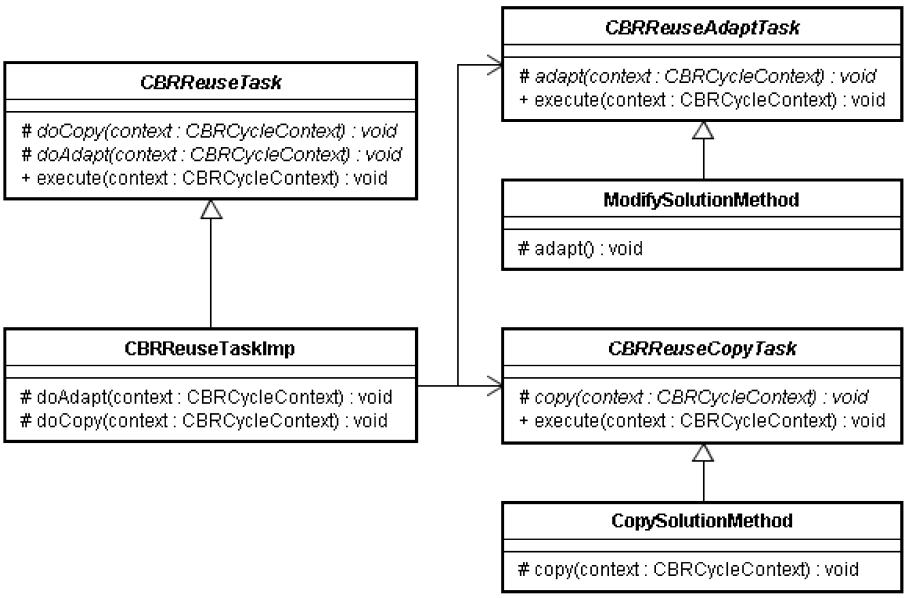

Figura 13: Diagrama de classes do processo de reutilização do ciclo RBC

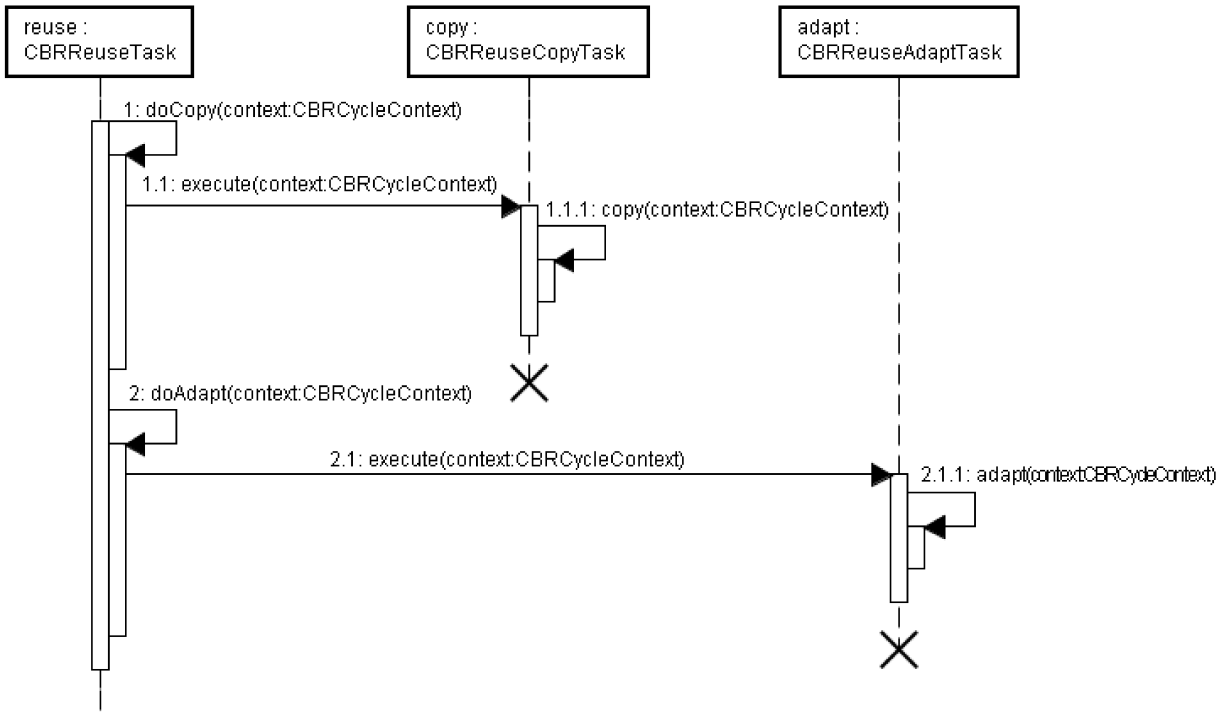

Figura 14: Diagrama de seqüência do processo de reutilização do ciclo RBC

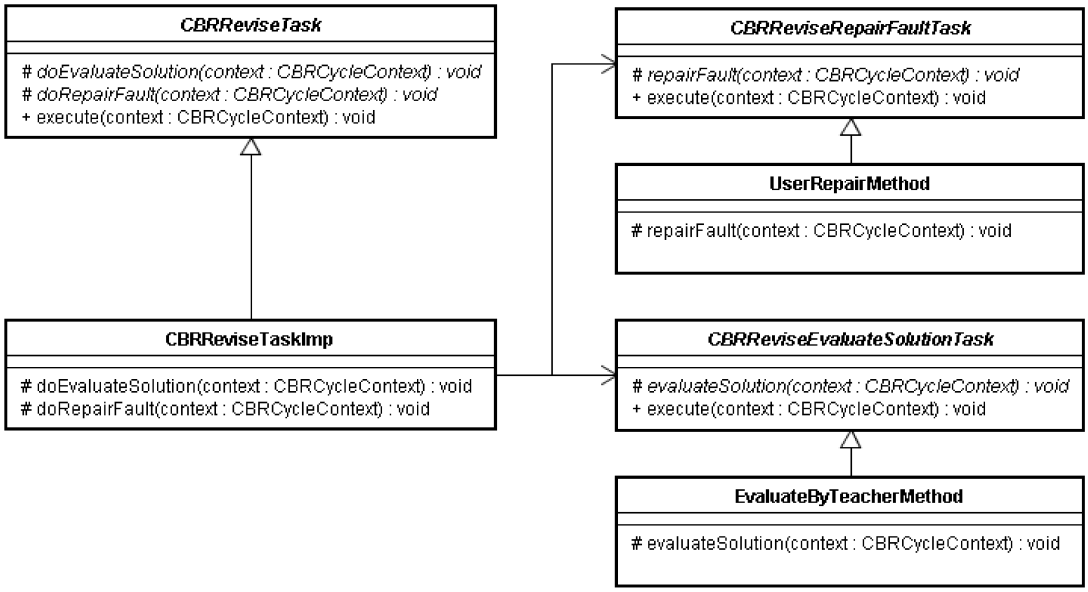

Figura 15: Diagrama de classes do processo de revisão do ciclo RBC

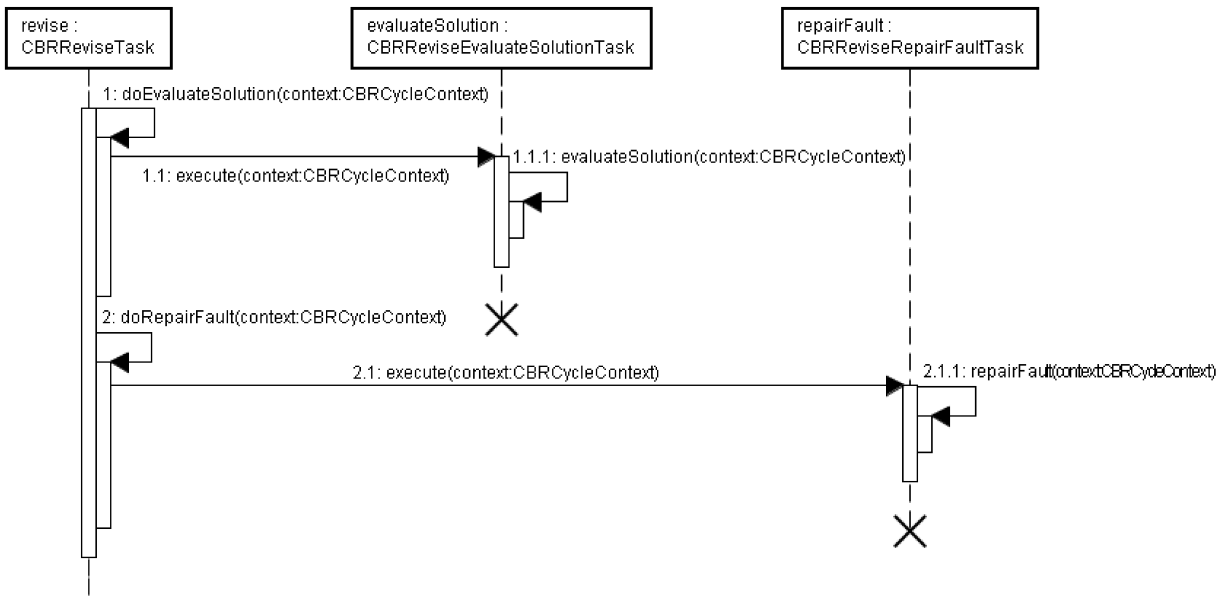

Figura 16: Diagrama de seqüência do processo de revisão do ciclo RBC

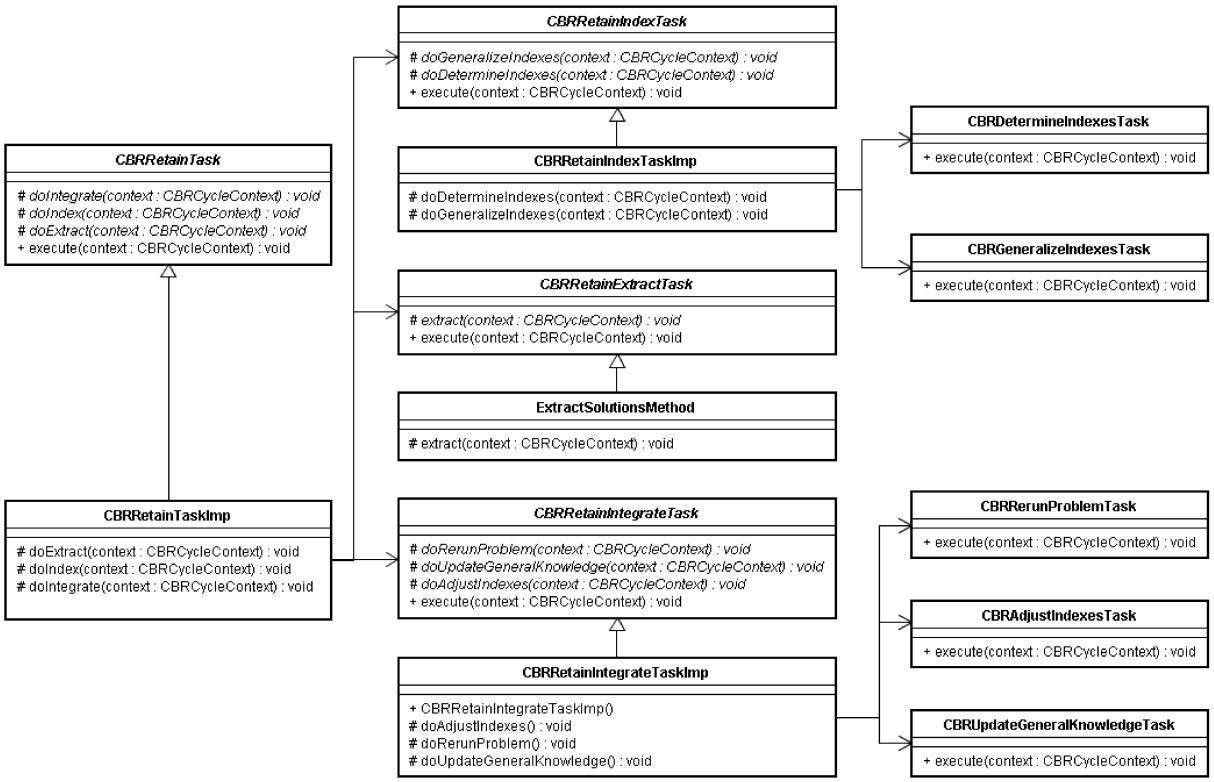

Figura 17: Diagrama de classes do processo de retenção do ciclo RBC

4.3.3 Geração de casos de teste automatizados do TAF

O algoritmo responsável por gerar os casos de teste automatizados pode ser visualizado no Algoritmo 1. Primeiramente, ele divide o log da sessão de teste exploratório em potenciais casos do sistema RBC (linha 3 que faz uma chamada para o Algoritmo 2). Após a obtenção dos potenciais casos, eles são divididos em grupos de casos mutuamente exclusivos, *i.e.*, casos que possuem seqüências de eventos do log de teste exploratório disjuntas (linha 4 que realiza uma chamada pro Algoritmo 3). Cada caso do grupo de logs serve então, de entrada para o sistema de RBC. O raciocinador baseado em casos tenta solucionar cada problema (caso) em questão (linha 8) paralelamente e depois armazena os resultados. Após todos os ciclos terminarem (linha 10), as soluções são avaliadas utilizando-se o Algoritmo 4, sendo este chamado na linha 11. Caso uma solução seja encontrada para algum caso do grupo, todo este ciclo deve ser repetido, até que todos os grupos sejam verificados e nenhuma nova solução seja encontrada.

No final do Algoritmo 1, o log de teste exploratório contém, na verdade, uma lista de UFs que substituem as determinadas seqüências de eventos. Então, um arquivo listando as UFs encontradas é criado. O arquivo é no formato XML e um exemplo pode ser

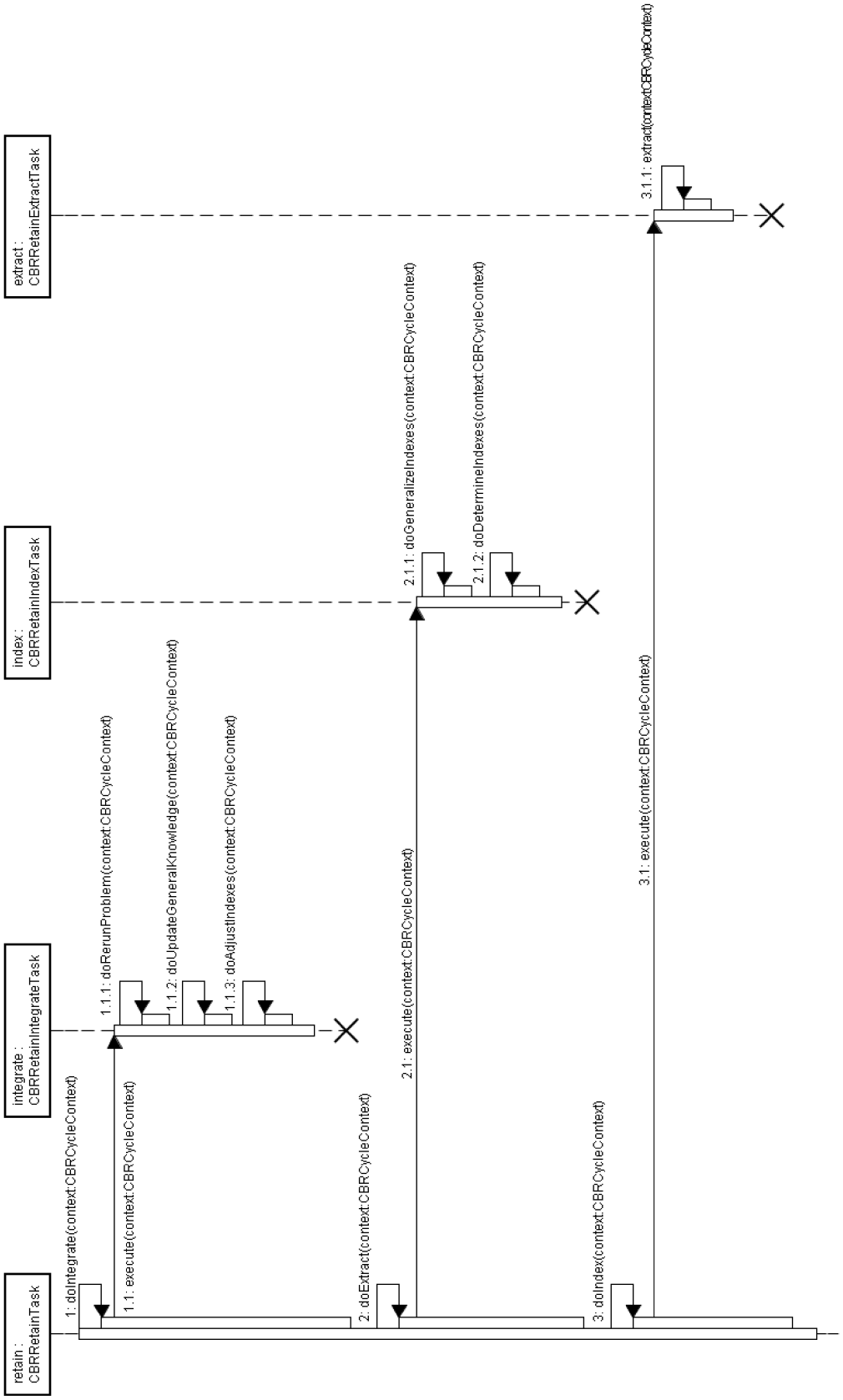

Figura 18: Diagrama de seqüência do processo de retenção do ciclo RBC

Algoritmo 1 Geração de um caso de teste automatizado do TAF a partir de uma sessão de teste exploratório do TAFLogger

Entrada: $listaLogs \leftarrow$ logs da sessão de teste exploratório gerado pelo TAFLogger
Entrada: $conjIniciais \leftarrow$ possíveis estados iniciais na base de casos do sistema RBC
Entrada: $conjFinais \leftarrow$ possíveis estados finais na base de casos do sistema RBC
Entrada: $ciclo \leftarrow$ ciclo do sistema RBC

1: **repita**
2: $solucaoEncontrada \leftarrow$ **falso**;
3: $todosOsCasos \leftarrow dividaLogsEmCasos(listaLogs, conjIniciais, conjFinais)$;
4: $listaExclusivos \leftarrow dividaCasosEmMutuamenteExclusivos(todosOsCasos)$;
5: **para todos** $listaCasos \mid listaCasos \in listaExclusivos \wedge \neg solucaoEncontrada$ **faça**
6: $listaSolucoes \leftarrow$ nova lista;
7: **para todos** $caso \mid caso \in listaCasos$ **faça**
8: $listaSolucoes \leftarrow listaSolucoes \cup ciclo.solucioneProblema(caso)$;
9: **fim para**
10: // Espere todos os ciclos terminarem
11: $solucaoEncontrada \leftarrow avalieSolucoes(listaLogs, listaCasos, listaSolucoes)$;
12: **fim para**
13: **até** $\neg solucaoEncontrada$

Algoritmo 2 Divisão de uma sessão de teste exploratório em potenciais casos do RBC

Entrada: $listaLogs \leftarrow$ logs da sessão de teste exploratório gerado pelo TAFLogger
Entrada: $conjIniciais \leftarrow$ possíveis estados iniciais na base de casos do sistema RBC
Entrada: $conjFinais \leftarrow$ possíveis estados finais na base de casos do sistema RBC
Saída: $listaCasos \leftarrow$ lista dos potenciais casos do sistema RBC

1: $i \leftarrow$ nova lista; // índices dos possíveis estados iniciais
2: $f \leftarrow$ nova lista; // índices dos possíveis estados finais
3: $index \leftarrow 0$; // índice atual
4: **para todos** $evt \mid evt \in listaLogs$ **faça**
5: **se** $\exists\ evt \mid evt \in conjIniciais$ **então**
6: $i \leftarrow i \cup index$;
7: **fim se**
8: **se** $\exists\ evt \mid evt \in conjFinais$ **então**
9: $f \leftarrow f \cup index$;
10: **fim se**
11: $index \leftarrow index + 1$;
12: **fim para**
13: **para todos** $a \mid a \in i$ **faça**
14: **para todos** $b \mid b \in f$ **faça**
15: **se** $a < b$ **então**
16: $listaCasos \leftarrow listaCasos \cup listaLogs.subLista(a, b)$;
17: **fim se**
18: **fim para**
19: **fim para**
20: **retorne** $listaCasos$;

Algoritmo 3 Divisão de potenciais casos do RBC mutuamente exclusivos

Entrada: $listaCasos \leftarrow$ lista dos potenciais casos do sistema RBC
Saída: $mExclusivos \leftarrow$ lista de listas dos potenciais casos mutuamente exclusivos

1: **enquanto** $listaCasos.tamanho() > 0$ **faça**
2: $mu \leftarrow$ nova lista; // lista dos casos mutuamente exclusivos
3: $mu \leftarrow mu \cup listaCasos.removePrimeiro()$;
4: **para todos** $casoAlvo \mid casoAlvo \in listaCasos$ **faça**
5: $exclusivo \leftarrow$ **verdadeiro**;
6: $inicioAlvo \leftarrow$ índice do início do $casoAlvo$;
7: $finalAlvo \leftarrow$ índice do final do $casoAlvo$;
8: **para todos** $casoExclusivo \mid casoExclusivo \in mu \wedge exclusivo$ **faça**
9: $inicioExclusivo \leftarrow$ índice do início do $casoExclusivo$;
10: $finalExclusivo \leftarrow$ índice do final do $casoExclusivo$;
11: $exclusivo \leftarrow finalAlvo \leq inicioExclusivo$;
12: $exclusivo \leftarrow exclusivo \vee inicioAlvo \geq finalExclusivo$;
13: **fim para**
14: **se** $exclusivo$ **então**
15: $mu \leftarrow mu \cup casoAlvo$;
16: **fim se**
17: **fim para**
18: $listaCasos \leftarrow listaCasos.removeTodos(mu)$;
19: $mExclusivos \leftarrow mExclusivos \cup mu$;
20: **fim enquanto**
21: **retorne** $mExclusivos$;

Algoritmo 4 Avalia as soluções dos casos do sistema RBC

Entrada: $listaLogs \leftarrow$ logs da sessão de teste exploratório gerado pelo TAFLogger
Entrada: $listaCasos \leftarrow$ lista de casos mutuamente exclusivos do sistema RBC
Entrada: $listaSolucoes \leftarrow$ lista das soluções dos casos
Saída: $solucaoEncontrada \leftarrow$ se achou uma solução para algum dos casos

1: $solucaoEncontrada \leftarrow$ **falso**;
2: **para todos** $caso \mid caso \in listaCasos \wedge solucao \mid solucao \in listaSolucoes$ **faça**
3: **se** $solucao \neq nulo$ **então**
4: $solucaoEncontrada \leftarrow$ **verdadeiro**;
5: $inicio \leftarrow$ índice do início do $caso$;
6: $final \leftarrow$ índice do final do $caso$;
7: $nova \leftarrow listaLogs.subLista(0, inicio - 1)$;
8: $nova \leftarrow nova \cup solucao$;
9: $nova \leftarrow nova \cup listaLogs.subLista(final + 1, listaLogs.tamanho())$;
10: $listaLogs \leftarrow nova$;
11: **fim se**
12: **fim para**
13: **retorne** $solucaoEncontrada$;

verificado a seguir. Este arquivo contém a lista de *Utility Functions* (*Steps*) do TAF, o qual é equivalente à um caso de teste automatizado do TAF.

Arquivo XML equivalente à um caso de teste automatizado

```
<?xml version="1.0" encoding="UTF-8" ?>
<TESTCASE>

<!-- Primeira Utility Function -->
<UTILITY_FUNCTION api_class="Nome da classe API" imp_class="Nome da classe de implementacao">

   <!-- Primeiro parametro -->
        <PARAM>
                <TYPE>Tipo do parametro</TYPE>
                <NAME>Nome do parametro</NAME>
                <VALUE>Valor do parametro</VALUE>
        </PARAM>

        <!-- Demais parametros -->

</UTILITY_FUNCTION>

<!-- Demais Utility Functions. -->

</TESTCASE>
```

4.4 Resultados obtidos

Com o intuito de analisar e validar a aplicação implementada, a seguinte metodologia foi seguida:

1. Foram escolhidos três casos de teste automatizados simplificados e responsáveis pelo teste de uma determinada funcionalidade, *e.g.*, a agenda de telefones ou mensagens;
2. Cada um dos casos de teste automatizados foi executado e seus logs foram armazenados;
3. Três sessões de teste exploratório serão realizadas seguindo as seguintes regras:
   - a primeira sessão exploratória terá o mesmo objetivo de um dos casos de teste automatizados;
   - a segunda sessão exploratória seguirá a risca os passos de um dos casos teste automatizado;
   - na terceira sessão exploratória será, de fato, um teste *ad-hoc*, contudo tem o mesmo escopo definido no teste.

4. Cada uma das sessões de teste exploratório terá seu log coletado;

5. Os logs coletados tanto da execução automatizada quanto das sessões de teste exploratórios servirão de entrada para aplicação implementada e três saídas serão produzidas;

6. As saídas da aplicação são confrontadas com seus resultados esperados.

O primeiro caso de teste escolhido foi o `TC_LOCK_UNLOCK_EMAIL` e seus passos de mais alto nível são:

1. `pb(0).navigationTk.launchApp(PhoneApplication.MESSAGES);` – ir para a aplicação de e-mail;

2. `pb(0).msgTk.scrollToMessage(EmailContent.SUBJECT_HELLO);` – selecionar o e-mail com o assunto da mensagem "Hello";

3. `pb(0).emailTk.changeEmailMsgState(EmailMsgState.LOCKED);` – bloquear o e-mail selecionado;

4. `pb(0).emailTk.changeEmailMsgState(EmailMsgState.UNLOCKED);` – desbloquear o e-mail selecionado;

5. `pb(0).navigationTk.goToIdle();` – voltar para a tela principal.

O caso de teste `TC_DELETE_EMAIL` foi escolhido como sendo o segundo. Os seguintes são os passos de mais alto nível:

1. `pb(0).navigationTk.launchApp(PhoneApplication.MESSAGES);` – ir para a aplicação de e-mail;

2. `pb(0).emailTk.deleteEmailMsg(EmailContent.SUBJECT_HELLO);` – remover o e-mail que contenha como assunto da mensagem: "Hello";

3. `pb(0).navigationTk.goToIdle();` – voltar para a tela principal.

Para completar a lista de 3 casos de teste foi escolhido o `TC_COMPOSE_EMAIL`, cujos passos de alto nível são:

1. `pb(0).navigationTk.launchApp(PhoneApplication.MESSAGES);` – ir para a aplicação de e-mail;

2. `EmailMsg msg = new EmailMsg(EmailContent.EMAIL_ADDRESS_ACCOUNT_1, EmailContent.SUBJECT_HELLO, EmailContent.MSG_HELLO));` `pb(0).msgTk.sendMessage(msg);` – compor e enviar uma mensagem de e-mail cujo assunto e corpo da mensagem sejam "Hello";

3. `pb(0).navigationTk.goToIdle();` – voltar para a tela principal.

Com os logs dos três casos de teste automatizados, foi a vez de realizar 3 sessões de teste exploratório e armazenar os logs. Então a aplicação implementada foi executada, tendo como entrada os logs dos casos de teste automatizados e os logs da sessão de teste exploratório. Os resultados podem ser visualizados na Tabelas 2, 3 e 4, para a primeira, segunda e terceira sessões de teste exploratório respectivamente.

Tabela 2: Resultado da primeira sessão de teste exploratório

#	**Ufs encontradas**
1	`pb(0).phoneTk.pressKey(PhoneHardKey.END)`
2	`pb(0).navigationTk.launchApp(PhoneApplication.MESSAGES)`
3	`pb(0).phoneTk.pressKey(PhoneHardKey.DOWN)`
4	`pb(0).phoneTk.pressKey(PhoneHardKey.CENTER)`
5	`pb(0).phoneTk.sleep(1000)`
6	`pb(0).navigationTk.goToAndSelectMenuItem(SmsEmsMmsItem.LOCK)`
7	`pb(0).navigationTk.goToAndSelectMenuItem(SmsEmsMmsItem.UNLOCK)`
8	`pb(0).navigationTk.launchApp(PhoneApplication.IDLE)`

Tabela 3: Resultado da segunda sessão de teste exploratório

#	**Ufs encontradas**
1	`pb(0).phoneTk.pressKey(PhoneHardKey.END)`
2	`pb(0).navigationTk.launchApp(PhoneApplication.MESSAGES)`
3	`pb(0).phoneTk.pressKey(PhoneHardKey.DOWN)`

continua na próxima página ...

Tabela 3: Resultado da segunda sessão de teste exploratório (continuação)

#	Ufs encontradas
4	`pb(0).phoneTk.pressKey(PhoneHardKey.CENTER)`
5	`pb(0).msgTk.scrollToMessage(EmailContent.SUBJECT_HELLO)`
6	`pb(0).navigationTk.goToAndSelectMenuItem(SmsEmsMmsItem.DELETE)`
7	`pb(0).navigationTk.launchApp(PhoneApplication.IDLE)`

Tabela 4: Resultado da terceira sessão de teste exploratório

#	Ufs encontradas
1	`pb(0).navigationTk.launchApp(PhoneApplication.MESSAGES)`
2	`pb(0).phoneTk.pressKey(PhoneHardKey.SOFTKEY_RIGHT)`
3	`pb(0).navigationTk.scrollTo(EmailItem.NEW_EMAIL)`
4	`pb(0).phoneTk.pressKey(PhoneHardKey.SOFTKEY_RIGHT)`
5	`pb(0).phoneTk.pressKey(PhoneHardKey.FOUR)` (2 vezes)
6	`pb(0).phoneTk.pressKey(PhoneHardKey.DOWN)`
7	`pb(0).phoneTk.pressKey(PhoneHardKey.THREE)` (2 vezes)
8	`pb(0).phoneTk.pressKey(PhoneHardKey.FIVE)` (3 vezes)
9	`pb(0).phoneTk.pressKey(PhoneHardKey.SIX)` (3 vezes)
10	`pb(0).phoneTk.pressKey(PhoneHardKey.SOFTKEY_LEFT)`
11	`pb(0).phoneTk.pressKey(PhoneHardKey.FIVE)` (3 vezes)
12	`pb(0).phoneTk.pressKey(PhoneHardKey.SIX)` (3 vezes)
13	`pb(0).phoneTk.pressKey(PhoneHardKey.SOFTKEY_RIGHT)`
14	`pb(0).phoneTk.pressKey(PhoneHardKey.SEVEN)`
15	`pb(0).phoneTk.pressKey(PhoneHardKey.THREE)` (2 vezes)
16	`pb(0).phoneTk.pressKey(PhoneHardKey.EIGHT)`
17	`pb(0).phoneTk.pressKey(PhoneHardKey.SEVEN)` (3 vezes)
18	`pb(0).phoneTk.pressKey(PhoneHardKey.SIX)` (3 vezes)
19	`pb(0).phoneTk.pressKey(PhoneHardKey.SEVEN)` (4 vezes)
20	`pb(0).phoneTk.pressKey(PhoneHardKey.FIVE)` (2 vezes)
21	`pb(0).phoneTk.pressKey(PhoneHardKey.FOUR)` (3 vezes)
22	`pb(0).phoneTk.pressKey(PhoneHardKey.ONE)` (8 vezes)

continua na próxima página ...

Tabela 4: Resultado da terceira sessão de teste exploratório (continuação)

#	Ufs encontradas
23	`pb(0).phoneTk.pressKey(PhoneHardKey.FIVE)` (3 vezes)
24	`pb(0).phoneTk.pressKey(PhoneHardKey.TWO)` (3 vezes)
25	`pb(0).phoneTk.pressKey(PhoneHardKey.SEVEN)` (4 vezes)
26	`pb(0).phoneTk.pressKey(PhoneHardKey.SIX)` (3 vezes)
27	`pb(0).phoneTk.pressKey(PhoneHardKey.THREE)` (3 vezes)
28	`pb(0).phoneTk.pressKey(PhoneHardKey.EIGHT)`
29	`pb(0).phoneTk.pressKey(PhoneHardKey.ONE)` (2 vezes)
30	`pb(0).phoneTk.pressKey(PhoneHardKey.EIGHT)` (2 vezes)
31	`pb(0).phoneTk.pressKey(PhoneHardKey.THREE)` (3 vezes)
32	`pb(0).phoneTk.pressKey(PhoneHardKey.SEVEN)` (4 vezes)
33	`pb(0).phoneTk.pressKey(PhoneHardKey.TWO)` (3 vezes)
34	`pb(0).phoneTk.pressKey(PhoneHardKey.ONE)` (2 vezes)
35	`pb(0).phoneTk.pressKey(PhoneHardKey.TWO)` (2 vezes)
36	`pb(0).phoneTk.pressKey(PhoneHardKey.SEVEN)` (3 vezes)
37	`pb(0).phoneTk.acceptDialog()`
38	`pb(0).phoneTk.pressKey(PhoneHardKey.DOWN)`
39	`pb(0).navigationTk.scrollTo(SmsEmsMmsItem.FIELD_SUBJECT)`
40	`pb(0).phoneTk.pressKey(PhoneHardKey.SOFTKEY_LEFT)`
41	`pb(0).phoneTk.pressKey(PhoneHardKey.FOUR)` (2 vezes)
42	`pb(0).phoneTk.pressKey(PhoneHardKey.THREE)` (2 vezes)
43	`pb(0).phoneTk.pressKey(PhoneHardKey.FIVE)` (10 vezes)
44	`pb(0).phoneTk.pressKey(PhoneHardKey.SIX)` (3 vezes)
45	`pb(0).phoneTk.pressKey(PhoneHardKey.SOFTKEY_RIGHT)` (2 vezes)
46	`pb(0).phoneTk.pressKey(PhoneHardKey.SOFTKEY_LEFT)`
47	`pb(0).navigationTk.goToIdle();`

Comparando os resultados obtidos pela primeira sessão de teste exploratório (Tabela 2) e o caso de teste automatizado `TC_LOCK_UNLOCK_EMAIL` pode-se ver que existem consideráveis semelhanças. A análise feita por parte da ferramenta implementada foi realizada com sucesso até a navegação que abre a aplicação de mensagens. Os dois passos seguintes realizam entrada efetiva na caixa de entrada dos e-mails, que fica dentro da aplicação de

mensagens. Uma chamada para a UF de *Sleep* foi encontrada logo em seguida, o que nos leva a pensar que ocorreu certo tempo entre a entrada na caixa de e-mails e o próximo passo, o bloqueio do e-mail com o assunto "Hello".

Para realizar o bloqueio e desbloqueio de uma mensagem de e-mail no TAF, é necessário a chamada de três outras UFs: *ScrollToMessage*, *GoToAndSelectMenuItem* e *WaitForTransientNotice*. Dessas três UFs apenas a parte que realiza efetivamente o bloqueio foi encontrada, a *GoToAndSelectMenuItem*. A UF de *ScrollToMessage* não foi encontrada pois a única mensagem existente na caixa de entrada era justamente a referida, não sendo necessário nenhuma ação. A UF *WaitForTransientNotice* também não realiza nenhuma ação direta sobre o telefone, pois apenas espera que a mensagem confirmando o bloqueio apareça e desapareça. Sem as duas UFs intermediárias, a aplicação não pôde realizar um *parse* sobre os logs de maneira mais eficiente e genérica. De maneira análoga podemos comparar a UF que realiza o desbloqueio da UF. Por fim, o retorno para tela principal foi analisado com sucesso.

Os resultados obtidos pela aplicação implementada na análise da segunda sessão de teste exploratório (Tabela 3) foram comparados ao caso de teste `TC_DELETE_EMAIL`. Novamente, a aplicação analisou certo a navegação para a aplicação de messagens, mas falhou quando a caixa de entrada de e-mails foi aberta. Seguindo o caso de teste, deveria ser encontrado uma chamada para a UF que remove um e-mail da pasta corrente. Tal UF utiliza-se de mais outras 3 UFs em ordem: *ScrollToMessage*, *GoToAndSelectMenuItem* e *VerifyDialog*. As duas primeiras foram encontradas pela ferramenta. Contudo, com o nível de precisão adotado pela aplicação, não foi possível inferir que somente 2 das 3 UFs responsáveis para remover um e-mail do telefone sejam suficientes para tal análise. No final do caso de teste, o retorno para tela principal do telefone foi encontrado.

A terceira sessão analisada foi que teve o pior resultado das três sessões, e que pode ser visualizado na Tabela 4. Comparada ao caso de teste `TC_COMPOSE_EMAIL`, somente o início, que é a navegação para a aplicação de mensagens, e o final foram analisados de forma satisfatória. A UF que manda uma mensagem de e-mail é composta por outras 2 UFs: *ComposeMessage* e *SendComposedMessage*. Essas outras duas são, por sua vez, compostas por diversas outras UFs, entre elas, UFs responsáveis pela entrada de dados em editores do telefone e outras verificações necessárias à composição da mensagem. A aplicação encontrada falhou completamente em analisar esta parte. Isso deve-se ao fato de que o teste exploratório tem o mesmo objetivo do teste, mas não segue os mesmos passos. Vários atalhos podem ser usados para uma mesma finalidade, *e.g.*, abrir o editor

de uma nova mensagem de texto. Outro fato que impacta de forma negativa é a entrada de dados no telefone, onde o algoritmo do *framework* de automação que implementa a entrada de texto exige um modo de edição específico e complexo. Já a pessoa que realiza a sessão exploratória não precisa de nenhum modo especial e entra com os dados de maneira mais eficiente, pois é mais *inteligente*. Além disso, existem outras nuâncias em relação à entrada de dados no telefone que tornam o reconhecimento mais difícil.

5 Conclusão e Trabalhos Futuros

5.1 Considerações finais

Teste de *software* é uma atividade crucial no atual processo de desenvolvimento de *softwares* de alta qualidade. Também contribui com uma grande parcela do custo total do desenvolvimento de *software* – freqüentemente cerca de 50% (CHERNONOZHKIN, 2001).

Casos de teste podem ser tanto executados manualmente ou de forma automatizada. Quando realizado de forma manual pode tornar-se suscetível a erros e altamente custoso, então o processo automatizado surge como uma das melhores alternativas. Contudo criar casos de teste passíveis de automação não é uma tarefa trivial e é responsável por grande parte do tempo despendido no processo de automatização de teste de *software*. Outra técnica de teste de software comumente difundida é o teste exploratório, e consiste em testar o sistema de uma forma não-sistemática, podendo ser útil para identificar casos especiais de teste, os quais não são facilmente capturados por técnicas formais.

Durante um ciclo de testes automatizados, apesar de eficiente, erros podem se infiltrar sem serem descobertos. Sessões de teste exploratório mostram-se bastante eficazes em identificar tais erros. A aplicação implementada durante a elaboração do presente trabalho obteve êxito combinando a eficácia do teste exploratório juntamente com a eficiência do teste automatizado.

O processo de geração de casos de teste automatizados tornou-se muito mais rápido e fácil pois os casos são gerados automaticamente a partir de uma sessão de teste exploratório. Isso porque executar uma sessão de teste exploratório utilizando um telefone celular e gerar o caso de teste automaticamente é mais fácil do que elaborá-lo programaticamente. Assim, o teste gerado automaticamente é incorporado aos próximos ciclos de execução automatizada. Então os erros descobertos durante a sessão exploratória podem ser reproduzidos automatizadamente, não somente para o modelo de telefone onde a sessão foi executada mas também para outros modelos devido a portabilidade dos casos

teste do TAF.

O ciclo de desenvolvimento de *software* pôde ser diminuído aplicando a metodologia de raciocínio baseado em casos para gerar automaticamente casos de teste automatizados. Uma aplicação foi implementada para validar a metodologia e tem como entrada apenas os *logs* de uma ferramenta de auxílio a testes exploratórios e dos *logs* de um *framework* de automação de testes dos *softwares* embutidos em telefones celulares.

5.2 Limitações

5.2.1 Limitação inerente à metodologia proposta

A metodologia de geração de casos de teste automatizados proposta possui algumas limitações devido ao domínio de entrada de dados utilizado. Somente os *logs* de uma sessão de teste exploratório não são suficientes para criar um caso de teste automatizado completo. Isso deve-se ao fato de que um caso de teste deve não somente realizar operações sobre o telefone, mas também decidir se tal operação obteve ou não o resultado esperado. Durante a sessão de testes exploratório, todas as verificações são realizadas pelo testador de forma manual, simplesmente olhando o telefone e utilizando sua experiência. E, somente caso o resultado obtido não seja o esperado, o testador armazena de forma textual em linguagem natural a verificação realizada e o possível problema encontrado. Portanto, UFs que têm como objetivo realizar algum tipo de verificação, tendem a não ter seu resultado devidamente raciocinado.

5.2.2 Limitação em relação à diferença entre a execução automatizada e *ad-hoc*

Durante a execução automatizada de um caso de teste, sempre são repetidos os mesmos passos para realizar uma determinada tarefa. Já em uma sessão exploratório isso não ocorre, pois fica ao cargo do testador manipular o telefone da forma que lhe convir, a fim de alcançar seus objetivos. Tal diferença fica evidente quando uma tarefa pode ser realizada de mais de uma maneira, *e.g.*, utilizar atalhos ou seguir caminhos alternativos para abrir uma aplicação. Como a aplicação utiliza os *logs* dos casos de teste automatizados para compor a base de casos, somente os caminhos percorridos por algum caso automatizado são passíveis de serem raciocinados. Tal limitação tem um grande impacto, *e.g.*, na entrada de dados do telefone onde o algoritmo do *framework* de automação que

implementa a entrada de texto exige um modo de edição específico e complexo.

5.2.3 Limitações com relação à performance

O processo de recuperação de casos apropriados é provavelmente o maior desafio no raciocínio baseado em casos. Tal processo está intimamente ligado à indexação dos dados e meio de armazenamento dos casos. O mecanismo de indexação adotado pode se tornar ineficiente quando a base de casos torna-se grande demais, pois não considera todos os atributos possíveis em sua indexação. Contudo, desconsiderá-los pode acarretar na diminuição da eficiência do raciocínio, porque casos potencialmente úteis podem ser simplesmente deixados de fora.

Outra questão é o próprio armazenamento dos casos na base de casos. Após os casos serem gerados, eles são incorporados à base de casos, que armazena-os em memória e não em disco. Isso se torna uma séria restrição quando o raciocínio exige que a base de casos seja grande. Contudo a utilização de um sistema de gerenciamento de base de dados mais robusto não foi utilizado por estar fora do escopo deste trabalho. Tal abordagem traria outros novos, e complexos, problemas para serem tratados durante o curto prazo disponível para a elaboração deste trabalho.

5.3 Sugestões para trabalhos futuros

Para trabalhos futuros, sugere-se:

- realizar uma análise mais embuida das restrições inerentes à metodologia, dividindo as restrições em grupos e então atacar cada grupo/sub-grupo, propondo abordagens alternativas ou mudanças necessárias na metodologia a fim de contorná-los;
- implementar um mecanismo mais eficiente de indexação da base de casos que maximizasse a recuperação de casos úteis ao raciocinador, pesquisando também por alternativas automatizadas e que utilizem outras abordagens, *e.g.*, inteligência artificial;
- utilização de um sistema de gerenciamento de banco de dados para transferir a responsabilidade da aplicação do gerenciamento do acesso, manipulação e organização dos casos.

Referências

AAMODT, A.; PLAZA, E. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, IOS Press, Amsterdam, The Netherlands, The Netherlands, v. 7, n. 1, p. 39–59, 1994. ISSN 0921-7126.

ALTHOFF, K.-D. et al. Case-based reasoning for medical decision support tasks: the inreca approach. *Artificial Intelligence in Medicine*, v. 12, n. 1, p. 25–41, jan 1998.

BACH, J. *Exploratory Testing Explained*. 2003. Disponível em: <http://www.satisfice-.com/articles/et-article.pdf>. Acesso em: 17 agosto 2006.

BACH, J. *What is Exploratory Testing?* 2003. Disponível em: <http://www.satisfice-.com/articles/what-is-et.shtml>. Acesso em: 18 janeiro 2007.

BOURQUE, P. et al. *Guide to the Software Engineering Body of Knowledge*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2001. Disponível em: <http:-//www.swebok.org/ironman/pdf/SWEBOK_Guide_2004.pdf>. Acesso em: 20 junho 2006.

CHERNONOZHKIN, S. K. Automated test generation and static analysis. *Programming and Computer Software*, v. 27, n. 2, p. 86–94, mar 2001.

DUSTIN, E.; RASHKA, J.; PAUL, J. *Automated Software Testing*. [S.l.]: Addison-Wesley Professional, 1999.

EICKELMANN, N.; RICHARDSON, D. An evaluation of software test environment architectures. In: SOFTWARE ENGINEERING, 1996. *Proceedings of the 18th International Conference*. Berlim, 1996. p. 353–364.

ESIPCHUK, I.; VALIDOV, D. Ptf-based test automation for java applications on mobile phones. In: *IEEE 10th International Symposium on Consumer Electronics*. [S.l.: s.n.], 2006. p. 1 – 3.

FENSEL, D.; ANGELE, J.; STUDER, R. The knowledge acquisition and representation language, karl. *IEEE Transactions on Knowledge and Data Engineering*, v. 10, n. 4, p. 527–550, 1998. ISSN 1041-4347.

GÖKER, M. H.; ROTH-BERGHOFE, T. The development and utilization of the case-based help-desk support system homer. *Engineering Application of Artificial Intelligence*, v. 12, n. 6, p. 665–680, dec 1999.

GOODMAN, M. Cbr in battle planning. In: *Proceedings of the Second Workshop on Case-Based Reasoning*. Pensacola Beach, US: [s.n.], 1989.

HICKS, I. D.; SOUTH, G. J.; OSHISANWO, A. O. Automated testing as an aid to systems integration. *BT Technology Journal*, v. 15, n. 3, p. 26–36, jul 1997.

HUANG, T.-L. T. C.-C. Rough set approach to case-based reasoning application. *Expert Systems with Applications*, v. 26, n. 3, p. 369–385, 2004.

JOHNSON, R.; FOOTE, B. Designing reusable classes. *Journal of Object-Oriented Programming*, v. 1, n. 2, p. 22 – 35, jun/jul 1988.

KAWAKAMI, L. et al. A test automation framework for mobile phones. In: *VIII IEEE Latin-American Test WorkShop*. [S.l.: s.n.], 2007 (to appear).

KOLODNER, J. L. An introduction to case-based reasoning. *Artificial Intelligence Review*, v. 6, n. 1, p. 3–34, mar 1992.

KOLODNER, J. L. *Case-Based Reasoning*. San Francisco, California, US.: Morgan Kaufmann, 1993.

KRIEGSMAN, M.; BARLETTA, R. Building a case-based help desk application. In: *IEEE Expert*. [S.l.: s.n.], 1993. v. 8, n. 6, p. 18–26.

LIAO, S. Case-based decision support system: Architecture for simulating military command and control. *European Journal of Operational Research*, v. 123, n. 3, p. 558–567, jun 2000.

MAGALDI, R. V. Cbr for troubleshooting on the flightline. In: *IEE Colloquium on Case-Based Reasoning: Prospects for Applications*. London, UK: [s.n.], 1994. p. 6/1–6/9.

MANTARAS, R. L. de et al. Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review*, v. 20, n. 3, p. 215–240, sep 2005.

MASTERTON, S. The virtual participant: Lessons to be learned from a case-based tutor's assistant. In: *Proceedings of Computer Supported Collaborative Learning*. Toronto, Canada: [s.n.], 1997. Disponível em: <citeseer.ist.psu.edu/masterton97virtual.html>. Acesso em: 14 agosto 2006.

RECHIA, D. N. *Especificação Formal de Restrições de Projeto para Frameworks Orientados a Objetos*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, dezembro 2005.

SCHMIDT, R. et al. Cased-based reasoning for medical knowledge-based systems. *International journal of medical informatics*, v. 64, n. 2–3, p. 355–367, 2001.

VOLLRATH, I.; WILKE, W.; BERGMANN, R. Case-based reasoning support for online catalog sales. *IEEE Internet Computing online*, v. 2, n. 4, p. 47–54, 1998.

WANGENHEIM, C. G. von; WANGENHEIM, A. von. *Raciocínio Baseado em Casos*. Barueri, São Paulo, Brasil: Manole, 2003.

WATSON, I. Case-based reasoning is a methodology not a technology. *Knowledge-Based Systems*, v. 12, p. 303–308, 1999.

APÊNDICE A – Código Fonte

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2007  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr;

import java.util.*;
import java.util.logging.Logger;

import cbr.casebase.CBRCaseBase;
import cbr.casebase.CBRCaseBaseLoader;
import cbr.casebase.caze.UtilityFunction;
import cbr.configurations.CBRConfigurationManager;
import cbr.parsers.LogSessionBuilder;
import cbr.runner.CBRBarrier;
import cbr.runner.CBRCycleRunner;
import cbr.similarity.utils.symbol.PhoneKeySim;
import cbr.similarity.utils.symbol.ScreenTypeSim;
import cbr.similarity.utils.symbol.groups.PhoneKeyGroup;

import com.motorola.taflogger.events.*;
import com.motorola.taflogger.hardware.PhoneKey;
import com.motorola.taflogger.rendering.ScreenType;

/**
 * TODO CBRSystem.java description.
 */
public class CBRSystem extends Thread
{
    /**
     * The {@link CBRSystem} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRSystem.class.getName());

    /**
     * A generic descending comparator.
     */
    private static final Comparator DESCENDING_COMPARATOR = new Comparator()
    {
        /*
         * (non-Javadoc)
         *
         * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
```

```
         */
        public int compare(Object arg0, Object arg1)
        {
            return ((Comparable) arg1).compareTo(arg0);
        }
    };

    /**
     * The log session file name. This should point to the exploratory test session's log file
     * resulted by the TAFLogger's run.
     */
    private String logSessionFileName;

    /**
     * The barrier to avoid threads to go further with all are completed.
     */
    private CBRBarrier barrier;

    /**
     * The list containing the log events.
     */
    private List logEvents;

    /**
     * The CBR configuration file name.
     */
    private String configurationFileName;

    /**
     * The output file name.
     */
    private String outputFileName;

    /**
     * Constructor.
     *
     * @param logSessionFileName The log session file name.
     * @param configFile The configuration file name.
     * @param outputFile The output file name.
     */
    public CBRSystem(String logSessionFileName, String configFile, String outputFile)
    {
        super("CBRSystem for " + logSessionFileName);
        this.logSessionFileName = logSessionFileName;
        this.configurationFileName = configFile;
        this.outputFileName = outputFile;
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Thread#run()
     */
    public void run()
    {
        // Parses the configuration from file and configures the system
        CBRConfigurationManager.getInstance().parseConfigurationFile(this.configurationFileName);

        // Initialize all utilities
        PhoneKeySim.getInstance().initialize();
        ScreenTypeSim.getInstance().initialize();

        // A barrier with the number of tasks plus one
        CBRBarrier runBarrier = new CBRBarrier(3);

        // Build the events from the log session
        LogSessionBuilder logSessionBuilder = new LogSessionBuilder(this.logSessionFileName,
                runBarrier);
        logSessionBuilder.start();

        // Builds and index the case base
        CBRCaseBaseLoader caseBaseLoader = new CBRCaseBaseLoader(runBarrier);
        caseBaseLoader.start();
```

```
            // Wait all threads to finish
            runBarrier.waitForRealese();

            // Get the log events from the log session builder
            this.logEvents = logSessionBuilder.getLogEvents();

            // Reason the log session seeking for utility function replacements of steps.
            reason();

            // Output the reasoner results
            outputReasonResults();
        }

        /**
         * Reason the log session seeking for utility function replacements of steps.
         */
        private void reason()
        {
            // Verifies if there was a change(solution reached of a sequence of log events) during one
            // cycle execution. If there was a change, the remaining threads are discarded and new ones
            // must be computed (since the log event list has changed).
            boolean hasChanged = false;

            do
            {
                hasChanged = false;
                List listOfRunners = buildAndDivideCBRCycleRunners();

                for (Iterator iter = listOfRunners.iterator(); iter.hasNext() && !hasChanged;/* empty */)
                {
                    List cbrCycleRunners = (List) iter.next();

                    // Run all CBRCycleRunners
                    runCBRCycleRunners(cbrCycleRunners);

                    // Verify the results of the CBRCycleRunners
                    hasChanged = verifyCBRCycleRunnersResults(cbrCycleRunners);
                }

            } while (hasChanged);
        }

        /**
         * Output the reasoner results.
         */
        private void outputReasonResults()
        {
            CBRSystemOutputBuilder builder = new CBRSystemOutputBuilder(this.outputFileName,
                    this.logEvents);
            builder.start();
        }

        /**
         * Starts all {@link CBRCycleRunner} in the list and wait them to finish.
         *
         * @param cbrCycleRunnersList The list of {@link CBRCycleRunner}.
         */
        private void runCBRCycleRunners(List cbrCycleRunnersList)
        {
            this.barrier = new CBRBarrier(cbrCycleRunnersList.size() + 1);

            for (Iterator iter = cbrCycleRunnersList.iterator(); iter.hasNext();/* empty */)
            {
                CBRCycleRunner runner = (CBRCycleRunner) iter.next();
                runner.setBarrier(this.barrier);
                runner.start();
            }

            this.barrier.waitForRealese();
        }

        /**
```

```
     * Verify CBRCycleRunner output and update the log events with the results.
     *
     * @param cbrCycleRunners The list of ran {@link CBRCycleRunner}.
     * @return <code>true</code> if the log events list was updated; <code>false</code>
     * otherwise.
     */
    private boolean verifyCBRCycleRunnersResults(List cbrCycleRunners)
    {
        boolean hasChanged = false;

        CBRCycleRunner runner = null;
        UtilityFunction solution = null;

        // Try to get the first solution
        Iterator cycleRunnerIterator = cbrCycleRunners.iterator();
        while (cycleRunnerIterator.hasNext() && solution == null)
        {
            runner = (CBRCycleRunner) cycleRunnerIterator.next();
            solution = runner.getSolution();
        }

        // Create the new list with log event
        List newLogEvents = new ArrayList(logEvents.size());

        int logEventIndex = 0;

        // While the CBRCycles reached an solution, create new
        while (solution != null && runner != null)
        {
            // State that was at least one change in the log events
            hasChanged = true;

            /** From last end to the begin */
            List beforeList = logEvents.subList(logEventIndex, runner.getBegin());
            newLogEvents.addAll(beforeList);

            /** From the begin to the end */
            UFEventGroup solutionGroup = new UFEventGroup(
                    GregorianCalendar.getInstance().getTime(), solution);

            List betweenList = logEvents.subList(runner.getBegin(), runner.getEnd() + 1);
            solutionGroup.addEvents(betweenList);

            // Add the first event of the group if the last event on the new log event list isn't an
            // phone screen event. This is necessary to add an possible begin/end state on the
            // automata.
            if (!beforeList.isEmpty()
                    && !(beforeList.get(beforeList.size() - 1) instanceof PhoneScreenEvent))
            {
                newLogEvents.add(betweenList.get(0));
            }

            newLogEvents.add(solutionGroup);

            // Force the solution to be null until find another
            solution = null;

            // Hold the last end index.
            logEventIndex = runner.getEnd();

            // Find the next solution
            while( cycleRunnerIterator.hasNext() && solution == null)
            {
                runner = (CBRCycleRunner) cycleRunnerIterator.next();
                solution = runner.getSolution();
            }

            /** From end until the real end */
            if (solution == null)
            {
                // There are no more solutions. Copy until the end of the old log events.
                List afterList = logEvents.subList(logEventIndex + 1, logEvents.size());
                newLogEvents.addAll(afterList);
```

```
            }
        }

279     // Replace the log events with the new one.
        if (hasChanged)
        {
            System.out.println("Changing: " + logEvents);
            logEvents = newLogEvents;
284         System.out.println("To:       : " + logEvents);
        }

        return hasChanged;
    }
289
    /**
     * Returns a list of list of {@link CBRCycleRunner}'s. Each list of runners are allowed to run
     * in parallel, because some of their log event sublist are mutually exclusive.
     *
294  * @return A list of list of {@link CBRCycleRunner}'s that are mutually exclusive.
     */
    private List buildAndDivideCBRCycleRunners()
    {
        long init = System.currentTimeMillis();
299
        // Build a list of runners
        List allCBRCycleRunners = getAllCBRCycleRunners();

        // Divide the runners so they can be mutually exclusive
304     List mutuallyExclusiveRunners = divideMutuallyExclusiveCBRunners(allCBRCycleRunners);

        logger.fine("Built and divided CBRCycleRunners in" + (System.currentTimeMillis() - init)
                + " ms.");

309     return mutuallyExclusiveRunners;
    }

    /**
     * Divides a list of CBRCycleRunners into lists of CBRCycleRunners that can run in parallel
314  * since their log event sublist are mutually exclusive.
     *
     * @param allCBRCycleRunners The list with all CBRCycleRunners.
     * @return The list of a list of CBRCycleRunners that can run in parallel since their log event
     * sublist are mutually exclusive.
319  */
    private List divideMutuallyExclusiveCBRunners(List allCBRCycleRunners)
    {
        // Divide the runners so they can be mutually exclusive
        List mutuallyExclusiveRunners = new LinkedList();
324
        // Local variables
        List runners;
        CBRCycleRunner probeRunner, otherRunner;
        boolean mutuallyExclusive;
329
        while (allCBRCycleRunners.size() > 0)
        {
            // Build a new list.
            runners = new ArrayList();
334
            // Add the first runner into the list.
            runners.add(allCBRCycleRunners.remove(0));

            for (int i = 0; i < allCBRCycleRunners.size(); ++i)
339         {
                // Get the next runner.
                probeRunner = (CBRCycleRunner) allCBRCycleRunners.get(i);

                // A priori, they're are mutually exclusive
344             mutuallyExclusive = true;

                // Verify if the target is mutually exclusive with all other runners
                for (Iterator iter = runners.iterator(); mutuallyExclusive && iter.hasNext();/* empty
                    */)
```

```
                {
                    otherRunner = (CBRCycleRunner) iter.next();
                    mutuallyExclusive &= otherRunner.isMutuallyExclusive(probeRunner);
                }

                // Verify if they end up being mutually exclusive
                if (mutuallyExclusive)
                {
                    runners.add(probeRunner);
                }
            }

            allCBRCycleRunners.removeAll(runners);
            mutuallyExclusiveRunners.add(runners);
        }

        return mutuallyExclusiveRunners;
    }

    /**
     * Returns a list with all CBRCycleRunners possible.
     *
     * @return The list with all CBRCycleRunners possible.
     */
    private List getAllCBRCycleRunners()
    {
        // Ascending ordered set.
        TreeSet initialIndices = new TreeSet();

        // Descending ordered set.
        TreeSet finalIndices = new TreeSet(DESCENDING_COMPARATOR);

        // Retrieve the list of possible initial screen types from the case base.
        Set possibleInitialScreenTypes = CBRCaseBase.getInstance().getPossibleInitialScreenTypes();

        // Retrieve the list of possible final screen types from the case base.
        Set possibleFinalScreenTypes = CBRCaseBase.getInstance().getPossibleFinalScreenTypes();

        Integer lastScreenTypeIndex = null;
        ScreenType lastScreenType = null;
        PhoneKeyGroup lastPhoneKeyType = null;

        boolean shouldBreak = false;

        for (int i = 0; i < logEvents.size(); i++)
        {
            // Get the current log event
            LogEvent evt = (LogEvent) logEvents.get(i);

            // Hold the index as an Object.
            Integer index = new Integer(i);

            ScreenType currentScreenType = getScreenType(evt);

            if (currentScreenType != null)
            {
                if (shouldBreak || !currentScreenType.equals(lastScreenType))
                {
                    if (possibleInitialScreenTypes.contains(currentScreenType))
                    {
                        initialIndices.add(index);
                    }

                    if (possibleInitialScreenTypes.contains(lastScreenType))
                    {
                        initialIndices.add(lastScreenTypeIndex);
                    }

                    if (possibleFinalScreenTypes.contains(currentScreenType))
                    {
                        finalIndices.add(index);
                    }
```

```
                    if (possibleFinalScreenTypes.contains(lastScreenType))
                    {
                        finalIndices.add(lastScreenTypeIndex);
424                 }

                    shouldBreak = false;
                }

429             lastScreenType = currentScreenType;
                lastScreenTypeIndex = index;
                continue;
            }

434         PhoneKey currentKey = getPhoneKey(evt);
            if (currentKey != null)
            {
                PhoneKeyGroup currentPhoneKeyType = PhoneKeyGroup.getGroup(currentKey);

439             shouldBreak |= !currentPhoneKeyType.equals(lastPhoneKeyType)
                        || currentPhoneKeyType.equals(PhoneKeyGroup.ACTION);

                lastPhoneKeyType = currentPhoneKeyType;
                continue;
444         }

            UtilityFunction currentUF = getUtilityFunction(evt);
            if (currentUF != null)
            {
449             initialIndices.add(index);
                finalIndices.add(index);
                continue;
            }
        }
454
        if (possibleFinalScreenTypes.contains(lastScreenType))
        {
            finalIndices.add(lastScreenTypeIndex);
        }
459
        return buildCBRCycleRunners(initialIndices, finalIndices);
    }

    /**
464  * Builds a list of CBRCycleRunners based a set with the indices of possible UF initial states
     * and its possible final indices.
     *
     * @param initialIndices The set of initial indices.
     * @param finalIndices The set of final indices.
469  * @return The list of CBRCycleRunners built.
     */
    private List buildCBRCycleRunners(Set initialIndices, Set finalIndices)
    {
        // The list containing the CBRCycleRunners.
474     List cbrCycleRunners = new ArrayList((initialIndices.size() * finalIndices.size()) / 2);

        int begin;
        int end;

479     for (Iterator initIter = initialIndices.iterator(); initIter.hasNext();/* empty */)
        {
            begin = ((Integer) initIter.next()).intValue();
            for (Iterator finalIter = finalIndices.iterator(); finalIter.hasNext();/* empty */)
            {
484             end = ((Integer) finalIter.next()).intValue();
                if (begin < end)
                {
                    cbrCycleRunners.add(new CBRCycleRunner(begin, end, logEvents));
                }
489         }
        }

        return cbrCycleRunners;
    }
```

```
494
        /**
         * Returns the phone key if the log event is a {@link KeyPressEvent}.
         *
         * @param evt The log event.
499      * @return The phone key if the log event is a {@link KeyPressEvent}; <code>null</code>
         * otherwise.
         */
        private PhoneKey getPhoneKey(LogEvent evt)
        {
504         if (evt instanceof KeyPressEvent)
            {
                return ((KeyPressEvent) evt).getPhoneKey();
            }

509         return null;
        }

        /**
         * Returns the screen type if the log event is a {@link PhoneScreenEvent}.
514      *
         * @param evt The log event.
         * @return The screen type if the log event is a {@link PhoneScreenEvent}; <code>null</code>
         * otherwise.
         */
519     private ScreenType getScreenType(LogEvent evt)
        {
            if (evt instanceof PhoneScreenEvent)
            {
                return ((PhoneScreenEvent) evt).getPhoneDisplay().getScreenType();
524         }

            return null;
        }

529     /**
         * Returns the utility function if the log event is a {@link UtilityFunction}.
         *
         * @param evt The log event.
         * @return The utility function if the log event is a {@link UtilityFunction};
534      * <code>null</code> otherwise.
         */
        private UtilityFunction getUtilityFunction(LogEvent evt)
        {
            if (evt instanceof UtilityFunction)
539         {
                return ((UtilityFunction) evt);
            }

            return null;
544     }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2007  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
549  * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
554  * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
559  * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr;
564
```

```
import java.io.File;

/**
 * This class server as the main driver for the PCBRSystem. It accepts the options in order to
 * control the system flow. The program options are:
 * <dl>
 * <dt> -config file
 * <dd> the configuration file [default "cbrsystem_config.xml"]
 * <dt> -output file
 * <dd> the output file [default "cbrsystem.output.xml"]
 * </dl>
 *
 * @author Bruno Martins Petroski
 * @author &lt;petroski@inf.ufsc.br&gt;
 */
public class CBRSystemMain
{
    /**
     * The usage string.
     */
    private static final String USAGE = "Usage: java cbr.CBRSystemMain [OPTION] LOGSESSIONFILE.xml"
            + "\n\n -config\t\t the configuration file [default \"cbrsystem_config.xml\"]\n"
            + "  -output\t\t the output file [default \"cbrsystem.output.xml\"]\n";

    /**
     * The output file. Default to 'cbrsystem.output'.
     */
    private static String outputFile = "cbrsystem.output.xml";

    /**
     * The configuration file. Default to 'cbrsystem_config.xml'
     */
    private static String configurationFile = "cbrsystem_config.xml";

    /**
     * The log session file (mandatory).
     */
    private static String logsessionFile = null;

    /**
     * Launches the CBRSystem, parsing arguments.
     *
     * @param args The line command arguments.
     */
    public static void main(String[] args)
    {
        // String f = "U:\\workspaces\\tcc\\integration\\taflogger\\0701052038_phonebook_entry.xml";
        // String f = "U:\\workspaces\\tcc\\integration\\taflogger\\0701091946_goto_phonebook.xml";
        // String f = "U:\\workspaces\\tcc\\integration\\taflogger\\0701101747_goto_addcontact.xml";
        // String f = "U:\\workspaces\\tcc\\integration\\taflogger\\0701101806_goto_phonebook.xml";
        // String f = "U:\\workspaces\\tcc\\integration\\taflogger\\0701111343_add_email.xml";
        // String f =
        // "U:\\workspaces\\tcc\\integration\\taflogger\\0701111852_gotophonebook_andaddanentry.xml";

        // Parse the arguments
        parseArguments(args);

        // Verify the argument passed
        verifyArguments();

        // Launch the CBRSystem
        CBRSystem system = new CBRSystem(logsessionFile, configurationFile, outputFile);
        system.start();
    }

    /**
     * Parse command line options and arguments to set various user-option flags and variables.
     *
     * @param args the command line arguments to be parsed.
     */
    protected static void parseArguments(String[] args)
    {
        int argumentsLength = args.length;
```

```
639         if (argumentsLength == 0)
            {
                printErrorMessageAndExit("No arguments found.");
            }

644         // Parse the options.
            for (int i = 0; i < argumentsLength; i++)
            {
                // Try to get the known options.
                if (args[i].equals("-config"))
649             {
                    // Verify argument
                    if (++i >= argumentsLength || args[i].startsWith("-"))
                    {
                        printErrorMessageAndExit("-config must have an argument.");
654                 }

                    // Store the configuration file name.
                    configurationFile = args[i];
                }
659             else if (args[i].equals("-output"))
                {
                    // Verify argument
                    if (++i >= argumentsLength || args[i].startsWith("-"))
                    {
664                     printErrorMessageAndExit("-output must have an argument");
                    }

                    // Store the output file name.
                    outputFile = args[i];
669             }
                else if (args[i].equals("--help"))
                {
                    printUsageAndExit();
                }
674             else if (args[i].toLowerCase().endsWith(".xml"))
                {
                    // Store the logsession file name.
                    logsessionFile = args[i];
                }
679             else
                {
                    printErrorMessageAndExit("Unrecognized option \"" + args[i] + "\"");
                }
            }
684     }

        /**
         * Verifies if all arguments passed were correct.
         */
689     private static void verifyArguments()
        {
            if (logsessionFile == null)
            {
                printErrorMessageAndExit("No LOGSESSIONFILE found.");
694         }

            try
            {
                if (!new File(logsessionFile).exists())
699             {
                    printErrorMessageAndExit("The file " + logsessionFile + " wasn't found.");
                }
            }
            catch (Exception e)
704         {
                printErrorMessageAndExit("The file " + logsessionFile + " wasn't found or hasn't read "
                        + "permissions.");
            }

709         try
            {
```

```
                if (!new File(configurationFile).exists())
                {
                    printErrorMessageAndExit("The file " + configurationFile + " wasn't found.");
714             }
            }
            catch (Exception e)
            {
                printErrorMessageAndExit("The file " + configurationFile
719                     + "wasn't found or hasn't read " + "permissions.");
            }
        }

        /**
724      * Prints the correct program usage and exits with an error.
         *
         * @param message The error message.
         */
        private static void printErrorMessageAndExit(String message)
729     {
            System.err.println(message + "\n\n");
            System.err.println("Try 'java cbr.CBRSystemMain --help' for more information.");
            System.exit(1);
        }
734
        /**
         * Prints the correct program usage and exits with an error.
         */
        private static void printUsageAndExit()
739     {
            System.err.println(USAGE);
            System.exit(1);
        }
    }
```

```
    /*
744  * Trabalho de Conclusão de Curso
     * Copyright (C) 2007  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
749  * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
754  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
759  * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr;

    import java.io.*;
764 import java.util.*;
    import java.util.logging.Level;
    import java.util.logging.Logger;

    import cbr.casebase.caze.Parameter;
769 import cbr.casebase.caze.UtilityFunction;

    import com.motorola.taflogger.events.KeyPressEvent;
    import com.motorola.taflogger.events.UFEventGroup;
    import com.motorola.taflogger.logmanagement.XMLLogElements;
774
    /**
     * TODO CBRSystemOutputBuilder.java description.
     */
    public class CBRSystemOutputBuilder extends Thread
779 {
        /**
         * The {@link CBRSystemOutputBuilder} logger.
```

```
     */
    private static final Logger logger = Logger.getLogger(CBRSystemOutputBuilder.class.getName());

    /**
     * The new line character.
     */
    private static final String NEWLINE = "\n";

    /**
     * The default encoding.
     */
    private static final String DEFAULT_ENCODING = "UTF-8";

    /**
     * The output file name.
     */
    private String outputFile;

    /**
     * The list with log events.
     */
    private List logEvents;

    /**
     * The buffer to write to file.
     */
    private StringBuffer buffer;

    /**
     * Constructor.
     *
     * @param outputFileName The output file name.
     * @param logEvents The list with the log events.
     */
    public CBRSystemOutputBuilder(String outputFileName, List logEvents)
    {
        this.outputFile = outputFileName;
        this.logEvents = logEvents;
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Thread#run()
     */
    public void run()
    {
        logger.fine("Writing CBRSytem output to file: " + this.outputFile + ".");

        // Build the buffer
        buildBuffer();

        // Write to file
        writeToFile();
    }

    /**
     * TODO CBRSystemOutputBuilder.buildBuffer() documentation.
     */
    private void buildBuffer()
    {
        // Create a new string buffer.
        buffer = new StringBuffer();

        buffer.append("<?xml version=\"1.0\" encoding=\"" + DEFAULT_ENCODING + "\" ?>");
        buffer.append(NEWLINE);
        buffer.append("<TESTCASE>");
        buffer.append(NEWLINE);
        buffer.append("<!-- Output generated at ");
        buffer.append(GregorianCalendar.getInstance().getTime().toString());
        buffer.append(". -->");

        // Verify the output.
```

```
            Object object = null;
            for (Iterator iter = logEvents.iterator(); iter.hasNext();/* empty */)
            {
                object = iter.next();
859
                if (object instanceof UFEventGroup)
                {
                    // Add utility function event group.
                    appendUtilityFunction(((UFEventGroup) object).getUtilityFunction());
864             }

                else if (object instanceof KeyPressEvent)
                {
                    // Add key press event
869                 buildUFAndAppend((KeyPressEvent) object);
                }
            }

            buffer.append("</TESTCASE>");
874     }

        /**
         * TODO CBRSystemOutputBuilder.writeToFile() documentation.
         */
879     private void writeToFile()
        {
            try
            {
                // Create the output writer.
884             BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(
                        outputFile), DEFAULT_ENCODING));

                // Write to file
                writer.write(buffer.toString());
889
                // Close the file
                writer.close();
            }
            catch (UnsupportedEncodingException e)
894         {
                logger.log(Level.SEVERE, "Unable to store the output on file.", e);
            }
            catch (FileNotFoundException e)
            {
899             logger.log(Level.SEVERE, "Unable to store the output on file.", e);
            }
            catch (IOException e)
            {
                logger.log(Level.SEVERE, "Unable to store the output on file.", e);
904         }
        }

        /**
         * TODO CBRSystemOutputBuilder.buildUFAndAppend() documentation.
909      *
         * @param event
         */
        private void buildUFAndAppend(KeyPressEvent event)
        {
914 //          String ufApi = "com.motorola.taf.utility.function.phone.api.PressKey";
    //          String ufImp = "com.motorola.taf.utility.function.phone.p2k.PressKeyImp";
    //          List parameters = new ArrayList(1);
    //          Parameter p = new Parameter("com.motorola.taf.frontend.option.PhoneHardKey",
    //                  "phoneHardKey", event.getPhoneKey().toString());
919 //          parameters.add(p);
    //          UtilityFunction uf = new UtilityFunction(ufApi, ufImp, parameters);

    //          appendUtilityFunction(uf);

924             buffer.append(NEWLINE);
                buffer.append("<!-- pb(0).phoneTk.pressKey(PhoneHardKey." + event.getPhoneKey().toString() + "
                    ) -->");
                buffer.append(NEWLINE);
```

```
        }

929     private void appendUtilityFunction(UtilityFunction uf)
        {
            buffer.append(NEWLINE);
            buffer.append("<UTILITY_FUNCTION");
            buffer.append(" " + XMLLogElements.UFAPI_ATTR + "=\"" + uf.getApiClassName() + "\"");
934         buffer.append(" " + XMLLogElements.UFIMP_ATTR + "=\"" + uf.getImpClassName() + "\">");

            for (Iterator iter = uf.getParameters().iterator(); iter.hasNext(); /* empty */)
            {
                appendParameter((Parameter) iter.next());
939         }
            buffer.append("</UTILITY_FUNCTION>");
        }

        private void appendParameter(Parameter p)
944     {
            buffer.append("<" + XMLLogElements.UFPARAM_ELEMENT + ">");

            buffer.append("<" + XMLLogElements.UFPARAM_TYPE_ELEMENT + ">");
            buffer.append(p.getType());
949         buffer.append("</" + XMLLogElements.UFPARAM_TYPE_ELEMENT + ">");

            buffer.append("<" + XMLLogElements.UFPARAM_NAME_ELEMENT + ">");
            buffer.append(p.getName());
            buffer.append("</" + XMLLogElements.UFPARAM_NAME_ELEMENT + ">");
954
            buffer.append("<" + XMLLogElements.UFPARAM_VALUE_ELEMENT + ">");
            buffer.append(p.getValue());
            buffer.append("</" + XMLLogElements.UFPARAM_VALUE_ELEMENT + ">");

959         buffer.append("</" + XMLLogElements.UFPARAM_ELEMENT + ">");
            buffer.append(NEWLINE);
        }

    }
```

```
    /*
964  * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
969  * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
974  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
979  * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr;

    import cbr.cycle.CBRCycleContext;
984
    /**
     * A CBRTask in one of the 4 R's (retrieve, reuse, revise and retain), or the tasks that make part
     * of one of these.
     */
989 public interface CBRTask
    {
        /**
         * Executes a CBR task.
         *
994      * @param context The CBR system context when this method is called.
         */
        public void execute(CBRCycleContext context);
    }
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.casebase;

import java.io.File;
import java.util.*;
import java.util.logging.Logger;

import cbr.casebase.caze.CBRCase;
import cbr.casebase.caze.UtilityFunction;
import cbr.similarity.utils.symbol.groups.*;

import com.motorola.taflogger.events.*;

/**
 * Holds all {@link CBRCase} into an indexed case bases.
 */
public class CBRCaseBase
{
    /**
     * The {@link CBRCaseBase} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRCaseBase.class.getName());

    /**
     * The possible initial screen types. While parsing the log session for new CBRCases, the
     * initial screen type of a valid CBRCase will be considered a possible one.
     */
    private Set possibleInitialScreenTypes;

    /**
     * The possible final screen types. While parsing the log session for new CBRCases, the final
     * screen type of a valid CBRCase will be considered a possible one.
     */
    private Set possibleFinalScreenTypes;

    /**
     * The mapping of all CBRCases. TODO explain how the indexing occurs.
     */
    public Map caseBase;

    /**
     * Maps all CBRCases to their index.
     */
    private Map indexing;

    /**
     * The last id used.
     */
    private static long lastId = Long.MIN_VALUE;

    /**
     * The singleton instance of this case case.
     */
    private static CBRCaseBase singleton;
```

```
1069    /**
         * The case base directory where the case base must be loaded.
         */
        private static String caseBaseDirectory = null;

1074    /**
         * Flag that holds whether to seek the case base directory recursively.
         */
        private static boolean recursiveLoadDirectory = false;

1079    /**
         * The minimum indexing percentage of phone keys of a phone key group on a CBRCase in order to
         * index the CBRCase under that group.
         */
        private static double minIndexPercentagePhoneKeyGroup = 0.225;
1084
        /**
         * The minimum indexing percentage of screen types of a screen type group on a CBRCase in order
         * to index the CBRCase under that group.
         */
1089    private static double minIndexPercentageScreenTypeGroup = 0.225;

        /**
         * The minimum indexing percentage of utility functions of a utility function group on a CBRCase
         * in order to index the CBRCase under that group.
1094     */
        private static double minIndexPercentageUtilityFunctionGroup = 0.225;

        /**
         * Sets the minimum indexing percentage of phone keys of a phone key group on a CBRCase in order
1099     * to index the CBRCase under that group.
         *
         * @param min The minimum indexing percentage.
         * @throws IllegalArgumentException If the minimum index percentage is lower than 0 or greater
         * than 1.
1104     */
        public static void setMinIndexPercentagePhoneKeyGroup(double min)
        {
            if (min < 0 || min > 1)
            {
1109            throw new IllegalArgumentException("The minimum indexing percentage must be greater"
                        + " than or equal 0 and lower than or equal 1. The percentage passed was: "
                        + min + ".");
            }

1114        minIndexPercentagePhoneKeyGroup = min;
        }

        /**
         * Sets the minimum indexing percentage of screen types of a screen type group on a CBRCase in
1119     * order to index the CBRCase under that group.
         *
         * @param min The minimum indexing percentage.
         * @throws IllegalArgumentException If the minimum index percentage is lower than 0 or greater
         * than 1.
1124     */
        public static void setMinIndexPercentageScreenTypeGroup(double min)
        {
            if (min < 0 || min > 1)
            {
1129            throw new IllegalArgumentException("The minimum indexing percentage must be greater"
                        + " than or equal 0 and lower than or equal 1. The percentage passed was: "
                        + min + ".");
            }

1134        minIndexPercentageScreenTypeGroup = min;
        }

        /**
         * Sets the minimum indexing percentage of utility functions of a utility function group on a
1139     * CBRCase in order to index the CBRCase under that group.
         *
```

```
         * @param min The minimum indexing percentage.
         * @throws IllegalArgumentException If the minimum index percentage is lower than 0 or greater
         * than 1.
1144     */
        public static void setMinIndexPercentageUtilityFunctionGroup(double min)
        {
            if (min < 0 || min > 1)
            {
1149            throw new IllegalArgumentException("The minimum indexing percentage must be greater"
                        + " than or equal 0 and lower than or equal 1. The percentage passed was: "
                        + min + ".");
            }

1154        minIndexPercentageUtilityFunctionGroup = min;
        }

        /**
         * Sets the case base directory where the case base must be loaded.
1159     *
         * @param dirName The case base directory where the case base must be loaded.
         * @throws IllegalArgumentException If the dirName is <code>null</code> or isn't a directory.
         */
        public static void setCaseBaseDirectory(String dirName)
1164    {
            if (dirName == null)
            {
                throw new IllegalArgumentException("The directory name cannot be null.");
            }
1169
            File directory = new File(dirName);

            if (!directory.isDirectory())
            {
1174            throw new IllegalArgumentException("Must be a directory. Passed directory was: "
                        + dirName);
            }

            caseBaseDirectory = dirName;
1179    }

        /**
         * Sets the recursiveLoadDirectory value.
         *
1184     * @param recursive The recursiveLoadDirectory to set.
         */
        public static void setRecursiveLoadDirectory(boolean recursive)
        {
            recursiveLoadDirectory = recursive;
1189    }

        /**
         * Returns the caseBaseDirectory value.
         *
1194     * @return Returns the caseBaseDirectory.
         */
        public static String getCaseBaseDirectory()
        {
            return caseBaseDirectory;
1199    }

        /**
         * Returns the recursiveLoadDirectory value.
         *
1204     * @return Returns the recursiveLoadDirectory.
         */
        public static boolean isRecursiveLoadDirectory()
        {
            return recursiveLoadDirectory;
1209    }

        /**
         * Constructor.
         */
```

```
1214    private CBRCaseBase()
        {
            this.caseBase = Collections.synchronizedMap(new HashMap(5));
            this.possibleInitialScreenTypes = Collections.synchronizedSet(new HashSet());
            this.possibleFinalScreenTypes = Collections.synchronizedSet(new HashSet());
1219        this.indexing = Collections.synchronizedMap(new HashMap(128));
        }

        /**
         * Returns the singleton instance of the case base.
1224     *
         * @return The singleton instance of the case base.
         */
        public static CBRCaseBase getInstance()
        {
1229        if (singleton == null)
            {
                singleton = new CBRCaseBase();
            }

1234        return singleton;
        }

        /**
         * Inserts a CBR case into the case base.
1239     *
         * @param caze The case to be inserted.
         */
        public synchronized void insertCBRCase(CBRCase caze)
        {
1244        if (!isValid(caze))
            {
                logger.finest("The CBRCase is a not valid case, so it won't be inserted  into the"
                        + " CBRCaseBase.");
                logger.finest("The CBRCase was: " + caze.getUtilityFuncion());
1249            return;
            }

            caze.setId(lastId++);

1254        CBRCaseIndex index = getCBRCaseIndex(caze);

            for (Iterator ufIter = index.utilityFunctionGroups.iterator(); ufIter.hasNext(); /* empty */)
            {
                Object uf = ufIter.next();
1259            Map ufMap = (Map) this.caseBase.get(uf);

                if (ufMap == null)
                {
                    ufMap = new HashMap(2);
1264                this.caseBase.put(uf, ufMap);
                }

                for (Iterator screenIter = index.screenTypeGroups.iterator(); screenIter.hasNext();/* empty
                      */)
                {
1269                Object display = screenIter.next();
                    Map keyMap = (Map) ufMap.get(display);

                    if (keyMap == null)
                    {
1274                    keyMap = new HashMap(5);
                        ufMap.put(display, keyMap);
                    }

                    for (Iterator keyIter = index.phoneKeyGroups.iterator(); keyIter.hasNext();/* empty */)
1279                {
                        Object key = keyIter.next();

                        Set current = (Set) keyMap.get(key);
                        if (current == null)
1284                    {
                            current = new HashSet();
```

```
                        keyMap.put(key, current);
                    }
                    current.add(caze);
                }
            }
        }

        System.out.println(caze);

        this.possibleInitialScreenTypes.add(caze.getInitialScreenType());
        this.possibleFinalScreenTypes.add(caze.getFinalScreenType());
    }

    /**
     * Returns the {@link CBRCaseIndex} of the CBRCase.
     *
     * @param caze The case on which the index shall be returned.
     * @return The index of the case.
     */
    private CBRCaseIndex getCBRCaseIndex(CBRCase caze)
    {
        CBRCaseIndex index = (CBRCaseIndex) this.indexing.get(caze);

        if (index == null)
        {
            // The case hasn't been cached yet, build and cache it.
            index = buildCBRIndex(caze);
            this.indexing.put(caze, index);
        }

        return index;
    }

    /**
     * Builds the CBRCaseIndex based of the case.
     *
     * @param caze The case to be based.
     * @return The CBRCaseIndex based of the case.
     */
    private CBRCaseIndex buildCBRIndex(CBRCase caze)
    {
        List transitions = caze.getTransitions().getTransitions();

        Map screensGroupMap = new HashMap();
        int phoneDisplaysSum = 0;

        Map keysGroupMap = new HashMap();
        int phoneKeysSum = 0;

        Map ufsGroupMap = new HashMap();
        int utilityFunctionsSum = 0;

        for (Iterator iter = transitions.iterator(); iter.hasNext();/* empty */)
        {
            LogEvent evt = (LogEvent) iter.next();

            if (evt instanceof PhoneScreenEvent)
            {
                increaseCounter(screensGroupMap, ScreenTypeGroup.getGroup(((PhoneScreenEvent) evt)
                        .getPhoneDisplay().getScreenType()));
                phoneDisplaysSum++;
            }
            else if (evt instanceof KeyPressEvent)
            {
                increaseCounter(keysGroupMap, PhoneKeyGroup.getGroup(((KeyPressEvent) evt)
                        .getPhoneKey()));
                phoneKeysSum++;
            }
            else if (evt instanceof UtilityFunction)
            {
                increaseCounter(ufsGroupMap, UtilityFunctionGroup.getGroup((UtilityFunction) evt));
                utilityFunctionsSum++;
            }
```

```
1359            else if (evt instanceof UFEventGroup)
                {
                    System.out.println(evt);
                }
            }
1364
            List screensGroupList = new ArrayList(screensGroupMap.size());
            for (Iterator iter = screensGroupMap.keySet().iterator(); iter.hasNext();/* empty */)
            {
                ScreenTypeGroup group = (ScreenTypeGroup) iter.next();
1369            int value = ((Integer) screensGroupMap.get(group)).intValue();

                if ((double) value / phoneDisplaysSum > minIndexPercentageScreenTypeGroup)
                {
                    screensGroupList.add(group);
1374            }
            }

            List ufsGroupList = new ArrayList(ufsGroupMap.size());
            for (Iterator iter = ufsGroupMap.keySet().iterator(); iter.hasNext();/* empty */)
1379        {
                UtilityFunctionGroup group = (UtilityFunctionGroup) iter.next();
                int value = ((Integer) ufsGroupMap.get(group)).intValue();

                if ((double) value / utilityFunctionsSum > minIndexPercentageUtilityFunctionGroup)
1384            {
                    ufsGroupList.add(group);
                }
            }

1389        if (ufsGroupList.isEmpty())
            {
                ufsGroupList.add(UtilityFunctionGroup.NONE);
            }

1394        List keysGroupList = new ArrayList(keysGroupMap.size());
            for (Iterator iter = keysGroupMap.keySet().iterator(); iter.hasNext();/* empty */)
            {
                PhoneKeyGroup group = (PhoneKeyGroup) iter.next();
                int value = ((Integer) keysGroupMap.get(group)).intValue();
1399
                if ((double) value / phoneKeysSum > minIndexPercentagePhoneKeyGroup)
                {
                    keysGroupList.add(group);
                }
1404        }

            return new CBRCaseIndex(screensGroupList, keysGroupList, ufsGroupList);
        }

1409    /**
         * Increases in one the counter of a symbol group.
         *
         * @param group The group mapping.
         * @param symbol The symbol group.
1414     */
        private void increaseCounter(Map group, SymbolGroup symbol)
        {
            Integer count = (Integer) group.get(symbol);

1419        if (count == null)
            {
                count = new Integer(1);
                group.put(symbol, count);
            }
1424        else
            {
                group.put(symbol, new Integer(count.intValue() + 1));
            }
        }
1429
        /**
         * Verifies if a CBRCase is valid. A valid CBRCase is an active one (dispatches at least one key
```

```
         * press event) and contains the initial and final phone screen.
         *
1434     * @param caze The case to be verified.
         * @return <code>true</code> if the case is valid; <code>false</code> otherwise.
         */
        private boolean isValid(CBRCase caze)
        {
1439        boolean valid = true;

            // Is an active case.
            valid &= !caze.getPhoneKeys().isEmpty();

1444        // Contains the initial phone screen.
            valid &= valid && caze.getInitialScreenType() != null;

            // Contains the final phone screen.
            valid &= valid && caze.getFinalScreenType() != null;
1449
            return valid;
        }

        /**
1454     * Returns all CBRCases in the case base that are indexed as the case, so they have an high
         * probability to be similar.
         *
         * @param caze The case to base the indexing.
         * @return All CBRCases in the case base that are indexed as the case.
1459     */
        public Set getCBRCases(CBRCase caze)
        {
            // Creates a new set.
            Set cases = new HashSet();
1464
            // The case index.
            CBRCaseIndex index = getCBRCaseIndex(caze);

            for (Iterator ufIter = index.getUtilityFunctionGroups().iterator(); ufIter.hasNext();/* empty
                 */)
1469        {
                // The key is an uf group.
                Object ufGroup = ufIter.next();

                // The value is the utility function mapping
1474            Map ufMap = (Map) this.caseBase.get(ufGroup);

                if (ufMap == null)
                {
                    // No indexing has been done with this uf group. Carrying on.
1479                continue;
                }

                for (Iterator screenIter = index.getScreenTypeGroups().iterator(); screenIter.hasNext();/*
                     empty */)
                {
1484                // The key is an screen type group
                    Object screenTypeGroup = screenIter.next();

                    // The value is the phone key group mapping
                    Map screenMap = (Map) ufMap.get(screenTypeGroup);
1489
                    if (screenMap == null)
                    {
                        // No indexing has been done with this screen type group. Carrying on.
                        continue;
1494                }

                    for (Iterator keyIter = index.getPhoneKeyGroups().iterator(); keyIter.hasNext();/*
                         empty */)
                    {
                        // The key is a phone key group.
1499                    Object phoneKeyGroup = keyIter.next();

                        // The value is the list of CBRCases indexed.
```

```
                    Set current = (Set) screenMap.get(phoneKeyGroup);

1504                if (current == null)
                    {
                        // No indexing has been done with this phone key group. Carrying on.
                        continue;
                    }
1509                cases.addAll(current);
                }
            }
        }

1514    return Collections.unmodifiableSet(cases);
    }

    /**
     * Returns the possible final screen types. While parsing the log session for new CBRCases, the
1519  * final screen type of a valid CBRCase will be considered a possible one.
     *
     * @return The possible final screen types.
     */
    public Set getPossibleFinalScreenTypes()
1524{
        return Collections.unmodifiableSet(possibleFinalScreenTypes);
    }

    /**
1529  * Returns the possible initial screen types. While parsing the log session for new CBRCases,
     * the initial screen type of a valid CBRCase will be considered a possible one.
     *
     * @return The possible initial screen types.
     */
1534public Set getPossibleInitialScreenTypes()
    {
        return Collections.unmodifiableSet(possibleInitialScreenTypes);
    }

1539/**
     * TODO CBRCaseBase.java description.
     */
    private static class CBRCaseIndex
    {
1544    /**
         * The list of screen type groups.
         */
        private List screenTypeGroups;

1549    /**
         * The list of phone key groups.
         */
        private List phoneKeyGroups;

1554    /**
         * The list of utility functions groups.
         */
        private List utilityFunctionGroups;

1559    /**
         * Constructor.
         *
         * @param displays The list of screen type groups.
         * @param keys The list of phone key groups.
1564     * @param ufs The list of utility function groups.
         */
        public CBRCaseIndex(List displays, List keys, List ufs)
        {
            this.screenTypeGroups = displays;
1569        this.phoneKeyGroups = keys;
            this.utilityFunctionGroups = ufs;
        }

        /**
1574     * Returns the phoneKeyGroups value.
```

```
             *
             * @return Returns the phoneKeyGroups.
             */
            public List getPhoneKeyGroups()
1579        {
                return phoneKeyGroups;
            }

            /**
1584         * Returns the screenTypeGroups value.
             *
             * @return Returns the screenTypeGroups.
             */
            public List getScreenTypeGroups()
1589        {
                return screenTypeGroups;
            }

            /**
1594         * Returns the utilityFunctionGroups value.
             *
             * @return Returns the utilityFunctionGroups.
             */
            public List getUtilityFunctionGroups()
1599        {
                return utilityFunctionGroups;
            }

            /*
1604         * (non-Javadoc)
             *
             * @see java.lang.Object#equals(java.lang.Object)
             */
            public boolean equals(Object obj)
1609        {
                if (obj instanceof CBRCaseIndex)
                {
                    CBRCaseIndex other = (CBRCaseIndex) obj;
                    boolean equals = this.screenTypeGroups.equals(other.screenTypeGroups);
1614                equals = equals && this.phoneKeyGroups.equals(other.phoneKeyGroups);
                    equals = equals && this.utilityFunctionGroups.equals(other.utilityFunctionGroups);

                    return equals;
                }
1619
                return false;
            }

            /*
1624         * (non-Javadoc)
             *
             * @see java.lang.Object#hashCode()
             */
            public int hashCode()
1629        {
                return this.screenTypeGroups.hashCode() ^ this.phoneKeyGroups.hashCode()
                        ^ this.utilityFunctionGroups.hashCode();
            }
        }
1634
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2007  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
1639 * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
1644 * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.casebase;

import java.io.File;
import java.util.Iterator;
import java.util.Stack;
import java.util.logging.Level;
import java.util.logging.Logger;

import cbr.casebase.caze.CBRCase;
import cbr.runner.CBRBarrier;

import com.motorola.taflogger.events.*;
import com.motorola.taflogger.exceptions.ParseException;
import com.motorola.taflogger.logmanagement.LogSession;
import com.motorola.taflogger.visitors.LogEventVisitor;

/**
 * Class responsible to parse a log session file and adds the CBRCases it founds into the case base.
 */
public class CBRCaseBaseBuilder extends Thread implements LogEventVisitor
{
    /**
     * The {@link CBRCaseBaseBuilder} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRCaseBaseBuilder.class.getName());

    /**
     * The XML file that contains the log session.
     */
    private File file;

    /**
     * The barrier to hit when finished.
     */
    private CBRBarrier barrier;

    /**
     * The stack of CBRCases.
     */
    private Stack cbrCasesStack;

    /**
     * Holds the current CBRCase.
     */
    private CBRCase currentCBRCase;

    /**
     * Holds the last parsed phone screen event.
     */
    private PhoneScreenEvent lastPhoneScreenEvent;

    /**
     * Holds the last parsed log event.
     */
    private LogEvent lastParsedLogEvent;

    /**
     * Constructor.
     *
     * @param file The file to build the case base from.
     * @param barrier The barrier to hit when finished.
     */
    public CBRCaseBaseBuilder(File file, CBRBarrier barrier)
    {
        super("CBRCaseBuilder for file: " + file.getName());
        this.file = file;
```

```
1719            this.barrier = barrier;
                this.cbrCasesStack = new Stack();
        }

        /*
1724     * (non-Javadoc)
         *
         * @see java.lang.Thread#run()
         */
        public void run()
1729    {
            LogSession logSession;
            try
            {
                // Builds the log session.
1734            logSession = new LogSession(this.file);

                // Creates the CBRCases and insert them into the CBRCaseBase.
                logSession.getEventList().getRootGroup().accept(this);
            }
1739        catch (ParseException e)
            {
                logger.log(Level.SEVERE, "Error parsing the file: '" + file.getName() + "'.", e);
            }

1744        this.barrier.waitForRealese();
        }

        /*
         * (non-Javadoc)
1749     *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitEventGroup(com.motorola.taflogger.
             events.EventGroup)
         */
        public void visitEventGroup(EventGroup group)
        {
1754        for (Iterator it = group.getLogEvents().iterator(); it.hasNext(); /* empty */)
            {
                ((LogEvent) it.next()).accept(this);
            }
        }
1759
        /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitKeyPressEvent(com.motorola.taflogger.
             events.KeyPressEvent)
1764     */
        public void visitKeyPressEvent(KeyPressEvent evt)
        {
            // FIXME This verification is due to duplicated event caught in the TAF. This issue must be
            // verified.
1769        if (this.lastParsedLogEvent == null
                    || (this.lastParsedLogEvent != null && this.lastParsedLogEvent.getDate().getTime() !=
                        evt
                            .getDate().getTime()))
            {
                this.currentCBRCase.add(evt);
1774            this.lastParsedLogEvent = evt;
            }
        }

        /*
1779     * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitPhoneScreenEvent(com.motorola.
             taflogger.events.PhoneScreenEvent)
         */
        public void visitPhoneScreenEvent(PhoneScreenEvent evt)
1784    {
            // Current event shouldn't be null at this point... But since sometimes a phone screen
            // event is caught without any utility function, it will just be ignored.
            if (this.currentCBRCase != null)
```

```
        {
1789        // FIXME This verification is due to duplicated event caught in the TAF. This issue must
            // be
            // verified.
            if (this.lastParsedLogEvent == null
                    || (this.lastParsedLogEvent != null && this.lastParsedLogEvent.getDate()
1794                        .getTime() != evt.getDate().getTime()))
            {
                this.currentCBRCase.add(evt);
                this.lastPhoneScreenEvent = evt;
                this.lastParsedLogEvent = evt;
1799        }
        }

        /*
1804     * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitUFEventGroup(com.motorola.taflogger.
             events.UFEventGroup)
         */
        public void visitUFEventGroup(UFEventGroup evt)
1809    {
            // If it's not the first case, add this UF as an event.
            if (this.currentCBRCase != null)
            {
                this.currentCBRCase.add(evt.getUtilityFunction());
1814        }

            // Push into the stack of CBRCases
            this.cbrCasesStack.push(this.currentCBRCase);

1819        // Create the CBRCase corresponding to this UF.
            this.currentCBRCase = new CBRCase();

            // The solution to the CBRCase is actually the UF.
            this.currentCBRCase.setUtilityFuncion(evt.getUtilityFunction());
1824
            if (this.lastPhoneScreenEvent != null)
            {
                // Add the last phone screen event as the first one in this new CBRCase
                this.currentCBRCase.add(this.lastPhoneScreenEvent);
1829        }

            for (Iterator it = evt.getLogEvents().iterator(); it.hasNext(); /* empty */)
            {
                ((LogEvent) it.next()).accept(this);
1834        }

            if (this.lastPhoneScreenEvent != null && lastParsedLogEvent != null
                    && this.lastPhoneScreenEvent != lastParsedLogEvent)
            {
1839            this.currentCBRCase.add(this.lastPhoneScreenEvent);
            }

            // Insert the parsed event into the CBRCaseBase
            CBRCaseBase.getInstance().insertCBRCase(this.currentCBRCase);
1844
            // Pop the CBRCase on the top of the stack;
            this.currentCBRCase = (CBRCase) this.cbrCasesStack.pop();
        }

1849    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitCommentedLogEvent(com.motorola.
             taflogger.events.CommentedLogEvent)
         */
1854    public void visitCommentedLogEvent(CommentedLogEvent evt)
        {
            // Empty.
        }
```

```
1859    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitIconMapper(com.motorola.taflogger.
             events.IconMapper)
         */
1864    public void visitIconMapper(IconMapper mapper)
        {
            // Empty.
        }

1869    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitLogSessionStateEvent(com.motorola.
             taflogger.events.LogSessionStateEvent)
         */
1874    public void visitLogSessionStateEvent(LogSessionStateEvent evt)
        {
            // Empty.
        }

1879    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitPhoneAddedEvent(com.motorola.taflogger
             .events.PhoneAddedEvent)
         */
1884    public void visitPhoneAddedEvent(PhoneAddedEvent evt)
        {
            // Empty.
        }

1889    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitPhoneConnectionChangedEvent(com.
             motorola.taflogger.events.PhoneConnectionChangedEvent)
         */
1894    public void visitPhoneConnectionChangedEvent(PhoneConnectionChangedEvent evt)
        {
            // Empty.
        }

1899    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitPhoneRemovedEvent(com.motorola.
             taflogger.events.PhoneRemovedEvent)
         */
1904    public void visitPhoneRemovedEvent(PhoneRemovedEvent evt)
        {
            // Empty.
        }

1909    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitScreenshotEvent(com.motorola.taflogger
             .events.ScreenshotEvent)
         */
1914    public void visitScreenshotEvent(ScreenshotEvent evt)
        {
            // Empty.
        }

1919    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitTextEvent(com.motorola.taflogger.
             events.TextEvent)
         */
1924    public void visitTextEvent(TextEvent evt)
```

```
        {
            // Empty.
        }
    }
```

```
    /*
1929 * Trabalho de Conclusão de Curso
     * Copyright (C) 2007  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
1934 * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
1939 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
1944 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.casebase;

    import java.io.File;
1949 import java.io.FileFilter;
    import java.io.FilenameFilter;
    import java.util.ArrayList;
    import java.util.Iterator;
    import java.util.List;
1954 import java.util.logging.Logger;

    import cbr.runner.CBRBarrier;

    /**
1959 * Class responsible to load the case base from files.
     */
    public class CBRCaseBaseLoader extends Thread
    {
        /**
1964     * The {@link CBRCaseBaseLoader} logger.
         */
        private static final Logger logger = Logger.getLogger(CBRCaseBaseLoader.class.getName());

        /**
1969     * The file filter for directories.
         */
        private static final FileFilter DIRECTORY_FILE_FILTER = new FileFilter()
        {
            /*
1974         * (non-Javadoc)
             *
             * @see java.io.FileFilter#accept(java.io.File)
             */
            public boolean accept(File file)
1979        {
                return file.isDirectory();
            }
        };

1984    /**
         * The file name filter for XML files.
         */
        private static final FilenameFilter XML_FILE_FILTER = new FilenameFilter()
        {
1989        /*
             * (non-Javadoc)
             *
             * @see java.io.FilenameFilter#accept(java.io.File, java.lang.String)
             */
1994        public boolean accept(File dir, String name)
            {
```

```
            return name.toLowerCase().endsWith(".xml");
        }
    };

    /**
     * The barrier to finished when finished.
     */
    private CBRBarrier barrier;

    /**
     * Constructor.
     */
    /**
     * Constructor.
     *
     * @param barrier The barrier to finished when finished.
     */
    public CBRCaseBaseLoader(CBRBarrier barrier)
    {
        super();
        this.barrier = barrier;
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Thread#run()
     */
    public void run()
    {
        load(CBRCaseBase.getCaseBaseDirectory(), CBRCaseBase.isRecursiveLoadDirectory());
    }

    /**
     * Loads the CBRCases parsed in log sessions found with the directory into the CBRCaseBase.
     *
     * @param dirName The directory that contains the log sessions.
     * @param recursive <code>true</code> if directory must be sought recursively;
     * <code>false</code> otherwise.
     */
    private void load(String dirName, boolean recursive) throws IllegalArgumentException
    {
        File directory = new File(dirName);

        long begin = System.currentTimeMillis();
        logger.fine("Loading case base from directory: " + directory.getName());

        // The list of files to load from
        List xmlFiles = getXMLFiles(directory, recursive);

        // Create a barrier to be released only all files were loaded
        CBRBarrier loadBarrier = new CBRBarrier(xmlFiles.size() + 1);

        // Starts the building of all file found
        for (Iterator iter = xmlFiles.iterator(); iter.hasNext();/* empty */)
        {
            new CBRCaseBaseBuilder((File) iter.next(), loadBarrier).start();
        }

        // Hold until all were loaded
        loadBarrier.waitForRealese();

        // All files were loaded, hit the outer barrier
        barrier.waitForRealese();

        logger.fine("Loaded from directory in " + (System.currentTimeMillis() - begin) + " ms.");
    }

    /**
     * Returns the list with all XML files within the directory.
     *
     * @param directory The directory to search for XML files.
     * @param recursive <code>true</code> if directory must be sought recursively;
```

```
2069        * <code>false</code> otherwise.
            * @return The list with all XML files within the directory.
            */
           private List getXMLFiles(File directory, boolean recursive)
           {
2074           List xmlFiles = new ArrayList();

               if (recursive)
               {
                   // Recursively return the XML files from the directory children.
2079               File[] dirChildren = directory.listFiles(DIRECTORY_FILE_FILTER);

                   if (dirChildren != null)
                   {
                       for (int i = 0; i < dirChildren.length; i++)
2084                   {
                           xmlFiles.addAll(getXMLFiles(dirChildren[i], recursive));
                       }
                   }
               }
2089
               // All directories were loaded, now load the children.
               File[] fileChildren = directory.listFiles(XML_FILE_FILTER);

               if (fileChildren != null)
2094           {
                   for (int i = 0; i < fileChildren.length; i++)
                   {
                       xmlFiles.add(fileChildren[i]);
                   }
2099           }

               return xmlFiles;
           }
       }
```

```
       /*
2104    * Trabalho de Conclusão de Curso
        * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
        *
        * This library is free software; you can redistribute it and/or
        * modify it under the terms of the GNU Lesser General Public
2109    * License as published by the Free Software Foundation; either
        * version 2.1 of the License, or (at your option) any later version.
        *
        * This library is distributed in the hope that it will be useful,
        * but WITHOUT ANY WARRANTY; without even the implied warranty of
2114    * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
        * Lesser General Public License for more details.
        *
        * You should have received a copy of the GNU Lesser General Public
        * License along with this library; if not, write to the Free Software
2119    * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
        */
       package cbr.casebase.caze;

       import java.util.HashSet;
2124   import java.util.Set;

       import cbr.similarity.CBRSimilarity;
       import cbr.similarity.centraltendencymeasures.WeightedMean;
       import cbr.similarity.utils.set.IntersectionSim;
2129   import cbr.similarity.utils.set.SetSimilarity;

       import com.motorola.taflogger.events.KeyPressEvent;
       import com.motorola.taflogger.events.PhoneScreenEvent;
       import com.motorola.taflogger.rendering.ScreenType;
2134
       /**
        * Represents a case record of utility function.
        */
       public class CBRCase implements CBRSimilarity
2139   {
```

```
        /**
         * The CBR case record (deterministic finite automaton).
         */
        private long id;
2144
        /**
         * The CBR case context on which this case was inserted when it was generated.
         */
        private CBRCaseContext caseContext;
2149
        /**
         * The set of phone displays types (states).
         */
        private Set screenTypes;
2154
        /**
         * The set of phone keys (symbols).
         */
        private Set phoneKeys;
2159
        /**
         * The initial display type (initial state).
         */
        private ScreenType initialScreenType;
2164
        /**
         * The final phone display type (final state).
         */
        private ScreenType finalScreenType;
2169
        /**
         * The transitions list.
         */
        private Transitions transitions;
2174
        /**
         * The CBR case solution, an Utility Function (UF).
         */
        private UtilityFunction utilityFunction;
2179
        /**
         * The displays weight when compared to another CBR case.
         */
        private static double displaysWeight = 2;
2184
        /**
         * The keys weight when compared to another CBR case.
         */
        private static double keysWeight = 2;
2189
        /**
         * The initial display weight when compared to another CBR case.
         */
        private static double initialDisplayWeight = 1;
2194
        /**
         * The final weight when compared to another CBR case.
         */
        private static double finalDisplayWeight = 1;
2199
        /**
         * The transitionList weight when compared to another CBR case..
         */
        private static double transitionsWeight = 10;
2204
        /**
         * The phone displays set comparison strategy.
         */
        private static SetSimilarity phonesDisplaysSimilarity = new IntersectionSim();
2209
        /**
         * The phone keys set comparison strategy.
         */
```

```
    private static SetSimilarity phoneKeysSimilarity = new IntersectionSim();
2214
    /**
     * Constructor.
     */
    public CBRCase()
2219 {
        this.transitions = new Transitions();
        this.screenTypes = new HashSet();
        this.phoneKeys = new HashSet();
    }
2224
    /**
     * Returns the utilityFunction value.
     *
     * @return Returns the utilityFunction.
2229  */
    public UtilityFunction getUtilityFuncion()
    {
        return utilityFunction;
    }
2234
    /**
     * Sets the utilityFunction value.
     *
     * @param utilityFunction The utilityFunction to set.
2239  */
    public void setUtilityFuncion(UtilityFunction utilityFunction)
    {
        this.utilityFunction = utilityFunction;
    }
2244
    public synchronized double similarityTo(Object a)
    {
        if (!(a instanceof CBRCase))
        {
2249        return 0d;
        }

        CBRCase other = (CBRCase) a;

2254    double[] data = new double[5];

        // Displays similarity.
        data[0] = phonesDisplaysSimilarity.similarity(getScreenTypes(), other.getScreenTypes());

2259    // Final display similarity.
        data[1] = getFinalScreenType().similarityTo(other.getFinalScreenType());

        // Initial display similarity.
        data[2] = getInitialScreenType().similarityTo(other.getInitialScreenType());
2264
        // Keys similarity.
        data[3] = phoneKeysSimilarity.similarity(getPhoneKeys(), other.getPhoneKeys());

        // Transitions similarity.
2269    data[4] = getTransitions().similarityTo(other.getTransitions());

        return WeightedMean.compute(data, new double[] { displaysWeight, finalDisplayWeight,
                initialDisplayWeight, keysWeight, transitionsWeight });
    }
2274
    /**
     * Returns the displays value.
     *
     * @return Returns the displays.
2279  */
    public Set getScreenTypes()
    {
        return screenTypes;
    }
2284
    /**
```

```
         * Returns the transitionsList value.
         *
         * @return Returns the transitionsList.
2289     */
        public Transitions getTransitions()
        {
            return transitions;
        }
2294
        /**
         * Returns the initialDisplay value.
         *
         * @return Returns the initialDisplay.
2299     */
        public ScreenType getInitialScreenType()
        {
            return initialScreenType;
        }
2304
        /**
         * Returns the keys value.
         *
         * @return Returns the keys.
2309     */
        public Set getPhoneKeys()
        {
            return phoneKeys;
        }
2314
        /**
         * Returns the finalDisplay value.
         *
         * @return Returns the finalDisplay.
2319     */
        public ScreenType getFinalScreenType()
        {
            return finalScreenType;
        }
2324
        /**
         * Adds an utility function to the case.
         *
         * @param evt The utility function.
2329     */
        public void add(UtilityFunction evt)
        {
            // Add the transition in the list.
            this.transitions.add(evt);
2334    }

        /**
         * Adds a phone screen event.
         *
2339     * @param evt The phone screen event.
         */
        public void add(PhoneScreenEvent evt)
        {
            // If its the first display, store.
2344        if (this.initialScreenType == null)
            {
                this.initialScreenType = evt.getPhoneDisplay().getScreenType();
            }

2349        // Add the transition in the list.
            this.transitions.add(evt);

            // Add the display in the list.
            this.screenTypes.add(evt.getPhoneDisplay().getScreenType());
2354
            // Store the final display;
            this.finalScreenType = evt.getPhoneDisplay().getScreenType();
        }
```

```
2359    /**
         * Adds a key press event.
         *
         * @param evt The key press event.
         */
2364    public void add(KeyPressEvent evt)
        {
            // Add the transition in the list.
            this.transitions.add(evt);

2369        // Add the key in the list.
            this.phoneKeys.add(evt.getPhoneKey());
        }

        /*
2374     * (non-Javadoc)
         *
         * @see java.lang.Object#toString()
         */
        public String toString()
2379    {
            StringBuffer sb = new StringBuffer();

            sb.append("- # TESTCASE @ " + Long.toHexString(id) + " # -");
            sb.append("-------------------------------------------------------------------------------");
2384        sb.append("\n");
            sb.append("Utility function    : " + this.utilityFunction);
            sb.append("\n");
            sb.append("Phone key types     : " + this.phoneKeys);
            sb.append("\n");
2389        sb.append("Screen types        : " + this.screenTypes);
            sb.append("\n");
            sb.append("Initial screen type: " + this.initialScreenType);
            sb.append("\n");
            sb.append("Final screen type   : " + this.finalScreenType);
2394        sb.append("\n");
            sb.append("Transitions         : " + this.transitions);
            sb.append("\n");
            sb.append("- ## ---------------------------------------------------------------");
            sb.append("-------------------------------------\n");
2399
            return sb.toString();
        }

        /**
2404     * Sets the displaysWeight value.
         *
         * @param weight The displaysWeight to set.
         * @throws IllegalArgumentException If the weight is lower than 0 or greater than 100.
         */
2409    public static void setDisplaysWeight(double weight)
        {
            if (weight < 0 || weight > 100)
            {
                throw new IllegalArgumentException("The weight must be greater than or equal 0 and "
2414                    + "lower than or equal 100. The weight passed was: " + weight + ".");
            }

            displaysWeight = weight;
        }
2419
        /**
         * Sets the finalDisplayWeight value.
         *
         * @param weight The finalDisplayWeight to set.
2424     * @throws IllegalArgumentException If the weight is lower than 0 or greater than 100.
         */
        public static void setFinalDisplayWeight(double weight)
        {
            if (weight < 0 || weight > 100)
2429        {
                throw new IllegalArgumentException("The weight must be greater than or equal 0 and "
                        + "lower than or equal 100. The weight passed was: " + weight + ".");
```

```
            }

2434        finalDisplayWeight = weight;
        }

        /**
         * Sets the initialDisplayWeight value.
2439     *
         * @param weight The initialDisplayWeight to set.
         * @throws IllegalArgumentException If the weight is lower than 0 or greater than 100.
         */
        public static void setInitialDisplayWeight(double weight)
2444    {
            if (weight < 0 || weight > 100)
            {
                throw new IllegalArgumentException("The weight must be greater than or equal 0 and "
                        + "lower than or equal 100. The weight passed was: " + weight + ".");
2449        }

            initialDisplayWeight = weight;
        }

2454    /**
         * Sets the keysWeight value.
         *
         * @param weight The keysWeight to set.
         * @throws IllegalArgumentException If the weight is lower than 0 or greater than 100.
2459     */
        public static void setKeysWeight(double weight)
        {
            if (weight < 0 || weight > 100)
            {
2464            throw new IllegalArgumentException("The weight must be greater than or equal 0 and "
                        + "lower than or equal 100. The weight passed was: " + weight + ".");
            }

            keysWeight = weight;
2469    }

        /**
         * Sets the transitionsWeight value.
         *
2474     * @param weight The transitionsWeight to set.
         * @throws IllegalArgumentException If the weight is lower than 0 or greater than 100.
         */
        public static void setTransitionsWeight(double weight)
        {
2479        if (weight < 0 || weight > 100)
            {
                throw new IllegalArgumentException("The weight must be greater than or equal 0 and "
                        + "lower than or equal 100. The weight passed was: " + weight + ".");
            }
2484
            transitionsWeight = weight;
        }

        /**
2489     * Sets the phoneKeysSimilarity value.
         *
         * @param setSimilarity The phoneKeysSimilarity to set.
         */
        public static void setPhoneKeysSimilarity(SetSimilarity setSimilarity)
2494    {
            phoneKeysSimilarity = setSimilarity;
        }

        /**
2499     * Sets the phonesDisplaysSimilarity value.
         *
         * @param setSimilarity The phonesDisplaysSimilarity to set.
         */
        public static void setPhonesDisplaysSimilarity(SetSimilarity setSimilarity)
2504    {
```

```
            phonesDisplaysSimilarity = setSimilarity;
        }

        /**
2509     * Returns the id value.
         *
         * @return Returns the id.
         */
        public long getId()
2514    {
            return id;
        }

        /**
2519     * Sets the id value.
         *
         * @param id The id to set.
         */
        public void setId(long id)
2524    {
            this.id = id;
        }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
2529 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
2534 * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
2539 * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
2544 */
    package cbr.casebase.caze;

    public class CBRCaseContext
    {
2549    /**
         * The TAF test case that generated the this case.
         */
        private String testCaseName;

2554    /**
         * The TAF version of the test case that generated this case.
         */
        private String tafVersion;

2559    /**
         * The PTF build version that was used to run the test case that generated this case.
         */
        private String ptfBuildVersion;

2564    /**
         * The PTF runtime version that was used to run the test case that generated this case.
         */
        private String ptfRuntimeVersion;

2569    /**
         * Sets the ptfVersion value.
         *
         * @param ptfVersion The ptfVersion to set.
         */
2574    public void setPtfBuildVersion(String ptfVersion)
        {
```

```
            this.ptfBuildVersion = ptfVersion;
        }

2579    /**
         * Sets the tafVersion value.
         *
         * @param tafVersion The tafVersion to set.
         */
2584    public void setTafVersion(String tafVersion)
        {
            this.tafVersion = tafVersion;
        }

2589    /**
         * Sets the testCaseName value.
         *
         * @param testCaseName The testCaseName to set.
         */
2594    public void setTestCaseName(String testCaseName)
        {
            this.testCaseName = testCaseName;
        }

2599    /**
         * Sets the ptfRuntimeVersion value.
         *
         * @param ptfRuntimeVersion The ptfRuntimeVersion to set.
         */
2604    public void setPtfRuntimeVersion(String ptfRuntimeVersion)
        {
            this.ptfRuntimeVersion = ptfRuntimeVersion;
        }
    }
```

```
    /*
2609 * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
2614 * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
2619 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
2624 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.casebase.caze;

    import cbr.similarity.CBRSimilarity;
2629
    /**
     * Defines a parameter of a UF.
     */
    public class Parameter implements CBRSimilarity
2634 {
        /**
         * The parameter's class.
         */
        private String type;
2639
        /**
         * The parameter's name;
         */
        private String name;
2644
        /**
         * The parameter's value.
```

```
         */
        private String value;
2649
        /**
         * Constructor.
         *
         * @param type The parameter's type.
2654     * @param name The parameter's name.
         * @param value The parameter's value.
         */
        public Parameter(String type, String name, String value)
        {
2659        this.type = type;
            this.name = name;
            this.value = value;
        }

2664    /**
         * Constructor.
         */
        public Parameter()
        {
2669        this("", "", "");
        }

        /**
         * Returns the name value.
2674     *
         * @return Returns the name.
         */
        public String getName()
        {
2679        return name;
        }

        /**
         * Sets the name value.
2684     *
         * @param name The name to set.
         */
        public void setName(String name)
        {
2689        this.name = name;
        }

        /**
         * Returns the type value.
2694     *
         * @return Returns the type.
         */
        public String getType()
        {
2699        return type;
        }

        /**
         * Sets the type value.
2704     *
         * @param type The type to set.
         */
        public void setType(String type)
        {
2709        this.type = type;
        }

        /**
         * Returns the value value.
2714     *
         * @return Returns the value.
         */
        public String getValue()
        {
2719        return value;
```

```
        }

        /**
         * Sets the value value.
2724     *
         * @param value The value to set.
         */
        public void setValue(String value)
        {
2729        this.value = value;
        }

        /*
         * (non-Javadoc)
2734     *
         * @see java.lang.Object#toString()
         */
        public String toString()
        {
2739        StringBuffer buffer = new StringBuffer();

            buffer.append(this.type);
            buffer.append(" ");
            buffer.append(this.name);
2744        buffer.append(" : ");
            buffer.append(this.value);

            return buffer.toString();
        }
2749
        /*
         * (non-Javadoc)
         *
         * @see cbr.similarity.CBRSimilarity#similarityTo(java.lang.Object)
2754     */
        public double similarityTo(Object a)
        {
            if (!(a instanceof Parameter))
            {
2759            return 0d;
            }

            Parameter other = (Parameter) a;

2764        boolean equals = this.name.equals(other.getName());
            equals = equals && this.type.equals(other.getType());
            equals = equals && !"".equals(this.value);
            equals = equals && this.value.equals(other.getValue());

2769        return equals ? 1 : 0;
        }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2007  Bruno Martins Petroski <petroski@inf.ufsc.br>
2774 *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
2779 *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
2784 *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
2789 package cbr.casebase.caze;
```

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import cbr.similarity.CBRSimilarity;

import com.motorola.taflogger.events.KeyPressEvent;
import com.motorola.taflogger.events.LogEvent;
import com.motorola.taflogger.events.PhoneScreenEvent;

/**
 * Holds the list of transitions of a case.
 */
public class Transitions implements CBRSimilarity
{
    /**
     * The list with the transitions.
     */
    private List transitions;

    /**
     * Constructor.
     */
    public Transitions()
    {
        this.transitions = new ArrayList(20);
    }

    /**
     * Adds a log event to the end of the log event list.
     *
     * @param evt The log event to be added.
     */
    public void add(LogEvent evt)
    {
        this.transitions.add(evt);
    }

    /**
     * Returns the transitions value.
     *
     * @return Returns the transitions.
     */
    public List getTransitions()
    {
        return transitions;
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Object#toString()
     */
    public String toString()
    {
        StringBuffer buffer = new StringBuffer();

        buffer.append("{");
        for (Iterator iter = transitions.iterator(); iter.hasNext();)
        {
            buffer.append("[" + iter.next() + "]");
            if (iter.hasNext())
            {
                buffer.append(" -> ");
            }
        }
        buffer.append("}");

        return buffer.toString();
    }

    /*
     * (non-Javadoc)
```

```
2864        *
            * @see cbr.similarity.CBRSimilarity#similarityTo(java.lang.Object)
            */
           public double similarityTo(Object a)
           {
2869           if (!(a instanceof Transitions))
               {
                   return 0;
               }

2874           return transitionComparison(transitions, ((Transitions) a).getTransitions());
           }

           /**
            * Compares two list of transitions and returns its similarity value.
2879        *
            * @param x The first transition.
            * @param y The second transition.
            * @return The similarity value between the two transitions.
            */
2884       private double transitionComparison(List x, List y)
           {
               int m = x.size();
               int n = y.size();

2889           if (m > n)
               {
                   return transitionComparison(y, x);
               }

2894           int maxCount = 0;
               for (int i = 0; i <= n - m; ++i)
               {
                   List subList = y.subList(i, n);
                   if (subList.get(0) instanceof KeyPressEvent)
2899               {
                       continue;
                   }

                   maxCount = Math.max(compareAutomaton(x, subList), maxCount);
2904           }

               return (double) maxCount / n;
           }

2909       /**
            * Compares to transitions and returns the longest sequence of equal transitions.
            *
            * @param x The first transition.
            * @param y The second transition.
2914        * @return The number of transitions of the longest sequence of equal transitions.
            */
           private int compareAutomaton(List x, List y)
           {
               double threshold = 0.75;
2919           int count = 0;
               int j = 0;
               PhoneScreenEvent lastDisplayX = null;
               for (int i = 0; i < x.size() && i + j < y.size(); i++)
               {
2924               double localSimilarity = 0;

                   UtilityFunction ufX = getUtilityFunction(x.get(i));
                   UtilityFunction ufY = getUtilityFunction(y.get(i + j));

2929               if (ufX != null && ufY != null)
                   {
                       localSimilarity = ufX.similarityTo(ufY);

                       if (localSimilarity > threshold)
2934                   {
                           count++;
                           continue;
```

```
                }
                break;
2939        }
            else if (ufX != null || ufY != null)
            {
                break;
            }
2944
            PhoneScreenEvent currentDisplayX = getPhoneScreenEvent(x.get(i));
            PhoneScreenEvent currentDisplayY = getPhoneScreenEvent(y.get(i + j));

            if (currentDisplayX != null && currentDisplayY != null)
2949        {
                lastDisplayX = currentDisplayX;
                // Both displays changed.
                localSimilarity = currentDisplayX.similarityTo(currentDisplayY);
                if (localSimilarity > threshold)
2954            {
                    count++;
                    continue;
                }
                break;
2959        }
            else if (currentDisplayX == null && currentDisplayY != null && lastDisplayX != null)
            {
                localSimilarity = lastDisplayX.similarityTo(currentDisplayY);
                i--;
2964            j++;
                if (localSimilarity > threshold)
                {
                    count++;
                    continue;
2969            }

                continue;
            }
            else if (currentDisplayX != null && currentDisplayY == null)
2974        {
                break;
            }

            KeyPressEvent currentKeyX = getKeyPressEvent(x.get(i));
2979        KeyPressEvent currentKeyY = getKeyPressEvent(y.get(i + j));

            if (currentKeyX != null && currentKeyY != null)
            {
                localSimilarity = currentKeyX.similarityTo(currentKeyY);
2984            if (localSimilarity > threshold)
                {
                    count++;
                    continue;
                }
2989            break;
            }

            // TODO Supposed to happen?
            throw new RuntimeException();
2994    }

        return count;
    }

2999    /**
     * Returns the phone screen event if the object is a phone screen event, or <code>null</code>
     * otherwise.
     *
     * @param a The object.
3004 * @return The phone screen event if the object is a phone screen event, or <code>null</code>
     * otherwise.
     */
    private PhoneScreenEvent getPhoneScreenEvent(Object a)
    {
3009        if (a instanceof PhoneScreenEvent)
```

```
            {
                return (PhoneScreenEvent) a;
            }

            return null;
        }

        /**
         * Returns the key press event if the object is a key press event, or <code>null</code>
         * otherwise.
         *
         * @param a The object.
         * @return The phone screen event if the object is a key press event, or <code>null</code>
         * otherwise.
         */
        private KeyPressEvent getKeyPressEvent(Object a)
        {
            if (a instanceof KeyPressEvent)
            {
                return (KeyPressEvent) a;
            }

            return null;
        }

        /**
         * Returns the utility function if the object is a utility function, or <code>null</code>
         * otherwise.
         *
         * @param a The object.
         * @return The utility function if the object is a utility function, or <code>null</code>
         * otherwise.
         */
        private UtilityFunction getUtilityFunction(Object a)
        {
            if (a instanceof UtilityFunction)
            {
                return (UtilityFunction) a;
            }

            return null;
        }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.casebase.caze;

    import java.util.*;

    import cbr.similarity.CBRSimilarity;
    import cbr.similarity.centraltendencymeasures.WeightedMean;
    import cbr.similarity.utils.text.CaseInsensitiveSim;
    import cbr.similarity.utils.text.TextSimilarity;

    import com.motorola.taflogger.events.LogEvent;
    import com.motorola.taflogger.visitors.LogEventVisitor;
```

```
    /**
     * Represents an Utility Function of the TAF (Test Automation Framework).
3084 */
    public class UtilityFunction extends LogEvent implements CBRSimilarity
    {
        /**
         * The Utility Function API (application programming interface) class name.
3089     */
        private String api;

        /**
         * The Utility Function implementation class name.
3094     */
        private String imp;

        /**
         * The Utility Function parameters.
3099     */
        private List parameters;

        /**
         * The API class name weight when compared to another utility function.
3104     */
        private static double apiWeight = 7;

        /**
         * The Imp class name weight when compared to another utility function.
3109     */
        private static double impWeight = 1;

        /**
         * The parameters weight when compared to another utility function.
3114     */
        private static double parametersWeight = 2;

        /**
         * The API class name comparison strategy.
3119     */
        private static TextSimilarity apiSimilarity = new CaseInsensitiveSim();

        /**
         * The Imp class name comparison strategy.
3124     */
        private static TextSimilarity impSimilarity = new CaseInsensitiveSim();

        /**
         * The minimum API similarity to compute the Imp and the parameters and consider them in the
3129     * utility function's local similarity.
         */
        private static double minAPISimilarity = 0.7;

        /**
3134     * Constructor.
         *
         * @param apiClassName The Utility Function API (application programming interface) class name.
         * @param impClassName The Utility Function implementation class name.
         * @param parameters The Utility Function parameters.
3139     */
        public UtilityFunction(String apiClassName, String impClassName, List parameters)
        {
            super(GregorianCalendar.getInstance().getTime(), -1);
            this.api = apiClassName;
3144        this.imp = impClassName;
            this.parameters = parameters;
        }

        /**
3149     * Returns the apiClassName value.
         *
         * @return Returns the apiClassName.
         */
        public String getApiClassName()
```

```
3154    {
            return this.api;
        }

        /*
3159     * (non-Javadoc)
         *
         * @see java.lang.Object#toString()
         */
        public String toString()
3164    {
            StringBuffer buffer = new StringBuffer();

            buffer.append("UF[");

3169        String[] parts = getImpClassName().split("\\.");
            for (int i = 5; i < parts.length; i++)
            {
                buffer.append(parts[i]);
                if (i < parts.length - 1)
3174            {
                    buffer.append(".");
                }
            }

3179        buffer.append("(");
            Parameter param = null;
            boolean addComma = false;
            for (Iterator iter = getParameters().iterator(); iter.hasNext(); /* empty */)
            {
3184            param = (Parameter) iter.next();
                if (param.getValue() != null && !param.getValue().equals("null"))
                {
                    if (addComma)
                    {
3189                    buffer.append(",");
                    }
                    buffer.append(param.getName());
                    buffer.append("=");
                    buffer.append(param.getValue());
3194                addComma = true;
                }
            }
            buffer.append(")]");

3199        return buffer.toString();
        }

        /**
         * Returns the impClassName value.
3204     *
         * @return Returns the impClassName.
         */
        public String getImpClassName()
        {
3209        return this.imp;
        }

        /**
         * Returns the parameters value.
3214     *
         * @return Returns the parameters.
         */
        public List getParameters()
        {
3219        return Collections.unmodifiableList(this.parameters);
        }

        /*
         * (non-Javadoc)
3224     *
         * @see cbr.similarity.CBRSimilarity#similarityTo(java.lang.Object)
         */
```

```
    public synchronized double similarityTo(Object a)
    {
3229        if (!(a instanceof UtilityFunction))
            {
                return 0;
            }

3234        double[] data = new double[3];

            UtilityFunction other = (UtilityFunction) a;

            // Compare api
3239        data[0] = apiSimilarity.similarity(getApiClassName(), other.getApiClassName());

            if (data[0] > minAPISimilarity)
            {
                // Compare imp
3244            data[1] = impSimilarity.similarity(getImpClassName(), other.getImpClassName());

                // Compare parameters
                List otherParams = other.getParameters();
                if (this.parameters.size() == otherParams.size())
3249            {
                    if ( this.parameters.size() > 0)
                    {
                        for (int i = 0; i < this.parameters.size(); ++i)
                        {
3254                        data[2] += ((Parameter) this.parameters.get(i))
                                    .similarityTo(otherParams.get(i));
                        }
                        data[2] = data[2] / this.parameters.size();
                    }
3259                else
                    {
                        data[2] = 1d;
                    }

3264            }
                else
                {
                    data[2] = 0;
                }
3269        }
            else
            {
                data[1] = 0;
                data[2] = 0;
3274        }

            return WeightedMean.compute(data, new double[] { apiWeight, impWeight, parametersWeight });
    }

3279    /**
     * Sets the API class name comparison strategy.
     *
     * @param sim The API class name comparison strategy.
     */
3284    public static void setApiSimilarity(TextSimilarity sim)
    {
        apiSimilarity = sim;
    }

3289    /**
     * Sets the API class name weight when compared to another utility function.
     *
     * @param weight the API class name weight when compared to another utility function.
     * @throws IllegalArgumentException If the weight is lower than 0 or greater than 100.
3294 */
    public static void setApiWeight(double weight)
    {
        if (weight < 0 || weight > 100)
        {
3299        throw new IllegalArgumentException("The weight must be greater than or equal 0 and "
```

```
                    + "lower than or equal 100. The weight passed was: " + weight + ".");
        }

        apiWeight = weight;
    }

    /**
     * Sets the Imp class name comparison strategy.
     *
     * @param sim the Imp class name comparison strategy.
     */
    public static void setImpSimilarity(TextSimilarity sim)
    {
        impSimilarity = sim;
    }

    /**
     * Sets the Imp class name weight when compared to another utility function.
     *
     * @param weight the Imp class name weight when compared to another utility function.
     * @throws IllegalArgumentException If the weight is lower than 0 or greater than 100.
     */
    public static void setImpWeight(double weight)
    {
        if (weight < 0 || weight > 100)
        {
            throw new IllegalArgumentException("The weight must be greater than or equal 0 and "
                    + "lower than or equal 100. The weight passed was: " + weight + ".");
        }
        impWeight = weight;
    }

    /**
     * Sets the minAPISimilarity value.
     *
     * @param min The minAPISimilarity to set.
     * @throws IllegalArgumentException If the minimum screen type similarity is lower than 0 or
     * greater than 1.
     */
    public static void setMinAPISimilarity(double min)
    {
        if (min < 0 || min > 1)
        {
            throw new IllegalArgumentException("The minimum similarity must be greater than or "
                    + "equal 0 and lower than or equal 1. The percentage passed was: " + min + ".");
        }

        minAPISimilarity = min;
    }

    /**
     * Sets the parameters weight when compared to another utility function.
     *
     * @param weight the parameters weight when compared to another utility function.
     * @throws IllegalArgumentException If the weight is lower than 0 or greater than 100.
     */
    public static void setParametersWeight(double weight)
    {
        if (weight < 0 || weight > 100)
        {
            throw new IllegalArgumentException("The weight must be greater than or equal 0 and "
                    + "lower than or equal 100. The weight passed was: " + weight + ".");
        }

        parametersWeight = weight;
    }

    /*
     * (non-Javadoc)
     *
     * @see com.motorola.taflogger.events.LogEvent#accept(com.motorola.taflogger.visitors.
          LogEventVisitor)
     */
```

```
        public void accept(LogEventVisitor visitor)
        {
3374        // TODO Auto-generated method stub
            throw new UnsupportedOperationException("UtilityFunction.accept()");
        }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
3379 * Copyright (C) 2007  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
3384 * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
3389 * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
3394 */
    package cbr.configurations;

    import java.util.logging.Logger;

3399 import cbr.casebase.caze.CBRCase;
    import cbr.casebase.caze.UtilityFunction;
    import cbr.similarity.utils.number.ThresholdSim;
    import cbr.similarity.utils.set.IntersectionSim;
    import cbr.similarity.utils.text.CaseInsensitiveSim;
3404
    import com.motorola.taflogger.events.KeyPressEvent;
    import com.motorola.taflogger.rendering.PhoneDisplay;
    import com.motorola.taflogger.rendering.TextItem;

3409 /**
     * Responsible to manage the system's configuration.
     */
    public class CBRConfigurationManager
    {
3414    /**
         * The {@link CBRConfigurationManager} logger.
         */
        private static final Logger logger = Logger.getLogger(CBRConfigurationManager.class.getName());

3419    /**
         * The singleton instance of the configuration manager.
         */
        private static CBRConfigurationManager singleton;

3424    /**
         * Constructor.
         */
        private CBRConfigurationManager()
        {
3429        // Empty.
        }

        /**
         * Returns the singleton instance of the configuration manager.
3434     *
         * @return The singleton instance of the configuration manager.
         */
        public static CBRConfigurationManager getInstance()
        {
3439        if (singleton == null)
            {
                singleton = new CBRConfigurationManager();
            }
```

```
3444            return singleton;
        }

        /**
         * Parses the configuration file and configures the system.
3449     *
         * @param configurationFile The CBR configuration file.
         */
        public synchronized void parseConfigurationFile(String configurationFile)
        {
3454        // Initialize the whole system to its defaults to avoid unwanted remaining (for other
            // configuration file) values.
            initializeToDefaults();

            // Reads the configuration file
3459        CBRConfigurationReader reader = new CBRConfigurationReader(configurationFile);
            reader.start();

            try
            {
3464            reader.join();
            }
            catch (InterruptedException e)
            {
                logger.severe("Unable to join the configuration reader thread: " + e.getMessage());
3469        }
        }

        /**
         * Initializes the system to its default values.
3474     */
        private void initializeToDefaults()
        {
            defaultsToCBRCase();

3479        defaultsToKeyPressEvent();

            defaultsToPhoneDisplay();

            defaultsToTextItem();
3484
            defaultsToUtilityFunction();
        }

        /**
3489     * Configures the {@link CBRCase} to its default values.
         */
        private void defaultsToCBRCase()
        {
            // Weight defaults
3494        CBRCase.setInitialDisplayWeight(1);
            CBRCase.setFinalDisplayWeight(1);
            CBRCase.setDisplaysWeight(2);
            CBRCase.setKeysWeight(2);
            CBRCase.setTransitionsWeight(10);
3499
            // Similarity defaults
            CBRCase.setPhoneKeysSimilarity(new IntersectionSim());
            CBRCase.setPhonesDisplaysSimilarity(new IntersectionSim());
        }
3504
        /**
         * Configures the {@link KeyPressEvent} to its default values.
         */
        private void defaultsToKeyPressEvent()
3509    {
            // Weight defaults
            KeyPressEvent.setPhoneKeyWeight(4);
            KeyPressEvent.setTimeHeldWeight(3);

3514        // Similarity defaults
            KeyPressEvent.setTimeHeldSimilarity(new ThresholdSim(2250));
```

```
        }

        /**
3519     * Configures the {@link PhoneDisplay} to its default values.
         */
        private void defaultsToPhoneDisplay()
        {
            // Weight defaults
3524        PhoneDisplay.setTitleWeight(1);
            PhoneDisplay.setScreenTypeWeight(4);
            PhoneDisplay.setScreenItemsWeight(3);

            // Similarity defaults
3529        PhoneDisplay.setTitleSimilarity(new CaseInsensitiveSim());

            // Other defaults
            PhoneDisplay.setMinScreenTypeSimilarity(0.7);
        }
3534
        /**
         * Configures the {@link TextItem} to its default values.
         */
        private void defaultsToTextItem()
3539    {
            // Weights defaults
            TextItem.setIsTitleWeight(1);
            TextItem.setTextWeight(4);

3544        // Similarity defaults.
            TextItem.setTextSimilarity(new CaseInsensitiveSim());
        }

        /**
3549     * Configures the {@link UtilityFunction} to its default values.
         */
        private void defaultsToUtilityFunction()
        {
            // Similarities defaults
3554        UtilityFunction.setApiSimilarity(new CaseInsensitiveSim());
            UtilityFunction.setImpSimilarity(new CaseInsensitiveSim());

            // Weight defaults
            UtilityFunction.setApiWeight(7);
3559        UtilityFunction.setImpWeight(1);
            UtilityFunction.setParametersWeight(2);

            // Other defaults
            UtilityFunction.setMinAPISimilarity(0.7);
3564    }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2007  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
3569 * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
3574 * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
3579 * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.configurations;
3584
    import java.io.File;
    import java.io.IOException;
```

```
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.xml.sax.SAXException;

import cbr.casebase.CBRCaseBase;
import cbr.casebase.caze.CBRCase;
import cbr.cycle.CBRCycle;
import cbr.similarity.utils.number.NumberSimilarity;
import cbr.similarity.utils.number.PercentToleranceSim;
import cbr.similarity.utils.number.ThresholdSim;
import cbr.similarity.utils.set.*;
import cbr.similarity.utils.text.CaseInsensitiveSim;
import cbr.similarity.utils.text.CaseSensitiveSim;
import cbr.similarity.utils.text.TextSimilarity;

import com.motorola.taflogger.events.KeyPressEvent;
import com.motorola.taflogger.rendering.PhoneDisplay;
import com.motorola.taflogger.rendering.TextItem;

/**
 * Responsible configuring the system, by read it from a file.
 */
public class CBRConfigurationReader extends Thread
{
    /**
     * The {@link CBRConfigurationReader} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRConfigurationReader.class.getName());

    /**
     * The configuration file name.
     */
    private String configFileName;

    /**
     * Constructor.
     *
     * @param configFileName The configuration file name.
     */
    public CBRConfigurationReader(String configFileName)
    {
        super("CBRConfiguratinReader for " + configFileName);
        this.configFileName = configFileName;
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Thread#run()
     */
    public void run()
    {
        File configurationFile = new File(this.configFileName);

        // Parses the configuration file.
        try
        {
            parseConfigurationFile(configurationFile);
        }
        catch (ParserConfigurationException e)
        {
            logger.log(Level.SEVERE, "Error parsing the configuration file.", e);
        }
        catch (SAXException e)
        {
            logger.log(Level.SEVERE, "Error parsing the configuration file.", e);
```

```
            }
            catch (IOException e)
            {
                logger.log(Level.SEVERE, "Error parsing the configuration file.", e);
3664        }
        }

        /**
         * Parses the configuration file from the root.
3669     *
         * @param configFile The configuration file.
         * @throws IOException If any IO errors occur.
         * @throws SAXException If any parse errors occur.
         * @throws ParserConfigurationException If a DocumentBuilder cannot be created which satisfies
3674     * the configuration requested.
         */
        private void parseConfigurationFile(File configFile) throws ParserConfigurationException,
                SAXException, IOException
        {
3679        // Retrieves the document
            Document document = getDocument(configFile);

            Node next = document.getFirstChild();

3684        // Flag that holds whether the root element was found.
            boolean rootFound = false;

            while (!rootFound && next != null)
            {
3689            if (next.getNodeType() == Node.ELEMENT_NODE)
                {
                    if (!CBRConfigurationTags.PCBR_ROOT_ELEM.equals(next.getNodeName()))
                    {
                        throw new ParserConfigurationException("Invalid root group found.");
3694                }
                    rootFound = true;
                    break;
                }

3699            next = next.getNextSibling();
            }

            if (next == null)
            {
3704            throw new ParserConfigurationException("Root group not found.");
            }

            next = next.getFirstChild();
            while (next != null)
3709        {
                if (next.getNodeType() == Node.ELEMENT_NODE)
                {
                    if (CBRConfigurationTags.GLOBAL_ELEM.equals(next.getNodeName()))
                    {
3714                    // Found the CBRSystem element, parse it.
                        parseGlobalConfigurationSection(next);
                    }
                    else if (CBRConfigurationTags.LOCAL_ELEM.equals(next.getNodeName()))
                    {
3719                    // Found the TAFLogger element, parse it.
                        parseLocalConfigurationSection(next);
                    }
                    else
                    {
3724                    logger.warning("The tag " + next.getNodeName() + " isn't recognizable from the"
                                + " parseConfigurationFile method.");
                    }
                }

3729            next = next.getNextSibling();
            }
        }
```

```
        /**
3734     * Returns the generated DOM document based on the XML file passed in.
         *
         * @return The DOM document representing the log session.
         * @throws IOException If any IO errors occur.
         * @throws SAXException If any parse errors occur.
3739     * @throws ParserConfigurationException If a DocumentBuilder cannot be created which satisfies
         * the configuration requested.
         */
        private Document getDocument(File source) throws ParserConfigurationException, SAXException,
                IOException
3744    {
            // The instance to obtain the parser that produces DOM object trees from XML documents
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

            // The document builder that produces DOM object trees from XML documents.
3749        DocumentBuilder builder = factory.newDocumentBuilder();

            // Represents the entire XML document. It is the root of the document tree, and provides the
            // primary access to the document's data
            Document document = builder.parse(source);
3754        document.normalize();

            return document;
        }

3759    /**
         * Parses general CBR system configuration.
         *
         * @param root The root node of the CBR system configuration.
         */
3764    private void parseGlobalConfigurationSection(Node root)
        {
            Node next = root.getFirstChild();

            while (next != null)
3769        {
                if (next.getNodeType() == Node.ELEMENT_NODE)
                {
                    if (parseCBRCase(next))
                    {
3774                    // Found a case configuration tag and has already parsed it.
                    }
                    else if (parseCBRCaseBase(next))
                    {
                        // Found a case base configuration tag and has already parsed it.
3779                }
                    else if (parseCBRCycle(next))
                    {
                        // Found a cycle configuration tag and has already parsed it.
                    }
3784                else
                    {
                        logger.warning("The tag " + next.getNodeName() + " isn't recognizable from the"
                                + " parseGlobalConfigurationSection method.");
                    }
3789            }

                next = next.getNextSibling();
            }
        }
3794
        /**
         * Parses the local configuration section.
         *
         * @param root The root node of the TAFLogger configuration.
3799     */
        private void parseLocalConfigurationSection(Node root)
        {
            Node next = root.getFirstChild();

3804        while (next != null)
            {
```

```
            if (next.getNodeType() == Node.ELEMENT_NODE)
            {
                if (parsePhoneDisplay(next))
3809            {
                    // Found a phone display configuration tag and has already parsed it.
                }
                else if (parseKeyPressEvent(next))
                {
3814                // Found a key press event configuration tag and has already parsed it.
                }
                else if (parseTextItem(next))
                {
                    // Found a text item configuration tag and has already parsed it.
3819            }
                else
                {
                    logger.warning("The tag " + next.getNodeName() + " isn't recognizable from the"
                            + " parseLocalConfigurationSection method.");
3824            }
            }

            next = next.getNextSibling();
        }
3829    }

        /**
         * Parses the CBRCase configuration.
         *
3834     * @param node The root node of the CBRCase configuration.
         * @return <code>true</code> if the node was parsed; <code>false</code> otherwise.
         */
        private boolean parseCBRCase(Node node)
        {
3839        if (CBRConfigurationTags.PHONE_DISPLAYS_SIM_ELEM.equals(node.getNodeName()))
            {
                SetSimilarity sim = parseSetSimilarity(node);

                if (sim != null)
3844            {
                    CBRCase.setPhonesDisplaysSimilarity(sim);
                }
                return true;
            }
3849        else if (CBRConfigurationTags.PHONE_DISPLAYS_WEIGHT_SIM.equals(node.getNodeName()))
            {
                double weight = parseDouble(node.getFirstChild());
                if (weight > -1)
                {
3854                try
                    {
                        CBRCase.setDisplaysWeight(weight);
                    }
                    catch (IllegalArgumentException e)
                    {
3859                    logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                    }
                }
                return true;
3864        }
            else if (CBRConfigurationTags.PHONE_KEYS_SIM_ELEM.equals(node.getNodeName()))
            {
                SetSimilarity sim = parseSetSimilarity(node);

3869            if (sim != null)
                {
                    CBRCase.setPhoneKeysSimilarity(sim);
                }
                return true;
3874        }
            else if (CBRConfigurationTags.PHONE_KEYS_WEIGHT_ELEM.equals(node.getNodeName()))
            {
                double weight = parseDouble(node.getFirstChild());
                if (weight > -1)
```

```
3879            {
                    try
                    {
                        CBRCase.setKeysWeight(weight);
                    }
3884                catch (IllegalArgumentException e)
                    {
                        logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                    }
                }
3889            return true;
            }
            else if (CBRConfigurationTags.INITIAL_DISPLAY_WEIGHT_ELEM.equals(node.getNodeName()))
            {
                double weight = parseDouble(node.getFirstChild());
3894            if (weight > -1)
                {
                    try
                    {
                        CBRCase.setInitialDisplayWeight(weight);
3899                }
                    catch (IllegalArgumentException e)
                    {
                        logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                    }
3904            }
                return true;
            }
            else if (CBRConfigurationTags.FINAL_DISPLAY_WEIGHT_ELEM.equals(node.getNodeName()))
            {
3909            double weight = parseDouble(node.getFirstChild());
                if (weight > -1)
                {
                    try
                    {
3914                    CBRCase.setFinalDisplayWeight(weight);
                    }
                    catch (IllegalArgumentException e)
                    {
                        logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
3919                }
                }
                return true;
            }
            else if (CBRConfigurationTags.TRANSITIONS_WEIGHT_ELEM.equals(node.getNodeName()))
3924        {
                double weight = parseDouble(node.getFirstChild());
                if (weight > -1)
                {
                    try
3929                {
                        CBRCase.setTransitionsWeight(weight);
                    }
                    catch (IllegalArgumentException e)
                    {
3934                    logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                    }
                }
                return true;
            }
3939
            return false;
        }

        /**
3944     * Parses the CBRCaseBase configuration.
         *
         * @param node The root node of the CBRCaseBase configuration.
         * @return <code>true</code> if the node was parsed; <code>false</code> otherwise.
         */
3949    private boolean parseCBRCaseBase(Node node)
        {
```

```
        if (CBRConfigurationTags.CASE_BASE_DIRECTORY_ELEM.equals(node.getNodeName()))
        {
3954        String directory = parseString(node.getFirstChild());

            if (directory != null)
            {
                try
3959            {
                    CBRCaseBase.setCaseBaseDirectory(directory.trim());
                }
                catch (IllegalArgumentException e)
                {
3964                logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                    logger.warning(CBRConfigurationTags.CASE_BASE_DIRECTORY_ELEM
                            + " is a mandatory configuration tag. System is exiting.");
                    System.exit(-1);
                }
3969        }

            Node attr = node.getAttributes().getNamedItem(
                    CBRConfigurationTags.CASE_BASE_DIRECTORY_RECURSIVE_ATTR);

3974        if (attr != null)
            {
                CBRCaseBase.setRecursiveLoadDirectory(Boolean.valueOf(attr.getNodeValue())
                        .booleanValue());
            }
3979
            return true;
        }
        else if (CBRConfigurationTags.MIN_INDEX_PHONEKEY_ELEM.equals(node.getNodeName()))
        {
3984        double min = parseDouble(node.getFirstChild());
            if (min > -1)
            {
                try
                {
3989                CBRCaseBase.setMinIndexPercentagePhoneKeyGroup(min);
                }
                catch (IllegalArgumentException e)
                {
                    logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
3994            }
            }
            return true;
        }
        else if (CBRConfigurationTags.MIN_INDEX_PHONEDISPLAY_ELEM.equals(node.getNodeName()))
3999    {
            double min = parseDouble(node.getFirstChild());
            if (min > -1)
            {
                try
4004            {
                    CBRCaseBase.setMinIndexPercentageScreenTypeGroup(min);
                }
                catch (IllegalArgumentException e)
                {
4009                logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                }
            }
            return true;
        }
4014    else if (CBRConfigurationTags.MIN_INDEX_UTILITYFUNCTION_ELEM.equals(node.getNodeName()))
        {
            double min = parseDouble(node.getFirstChild());
            if (min > -1)
            {
4019            try
                {
                    CBRCaseBase.setMinIndexPercentageUtilityFunctionGroup(min);
                }
                catch (IllegalArgumentException e)
4024            {
```

```
                    logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                }
            }
            return true;
        }

        return false;
    }

    /**
     * Parses the CBRCycle configuration.
     *
     * @param node The root node of the CBRCycle configuration.
     * @return <code>true</code> if the node was parsed; <code>false</code> otherwise.
     */
    private boolean parseCBRCycle(Node node)
    {
        if (CBRConfigurationTags.MIN_SOLUTION_SIM_ELEM.equals(node.getNodeName()))
        {
            double min = parseDouble(node.getFirstChild());
            if (min > -1)
            {
                try
                {
                    CBRCycle.setMinimumSimilarity(min);
                }
                catch (IllegalArgumentException e)
                {
                    logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                }
                return true;
            }
        }

        return false;
    }

    /**
     * Parses the KeyPressEvent configuration.
     *
     * @param root The root node of the KeyPressEvent configuration.
     * @return <code>true</code> if the node was parsed; <code>false</code> otherwise.
     */
    private boolean parseKeyPressEvent(Node node)
    {
        if (CBRConfigurationTags.TIME_HELD_SIM_ELEM.equals(node.getNodeName()))
        {
            NumberSimilarity sim = parseNumberSimilarity(node);

            if (sim != null)
            {
                KeyPressEvent.setTimeHeldSimilarity(sim);
            }
            return true;
        }
        else if (CBRConfigurationTags.TIME_HELD_WEIGHT_ELEM.equals(node.getNodeName()))
        {
            double weight = parseDouble(node.getFirstChild());
            if (weight > -1)
            {
                try
                {
                    KeyPressEvent.setTimeHeldWeight(weight);
                }
                catch (IllegalArgumentException e)
                {
                    logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                }
            }
            return true;
        }
        else if (CBRConfigurationTags.PHONE_KEY_WEIGHT_ELEM.equals(node.getNodeName()))
        {
```

```
            double weight = parseDouble(node.getFirstChild());
4099        if (weight > -1)
            {
                try
                {
                    KeyPressEvent.setPhoneKeyWeight(weight);
4104            }
                catch (IllegalArgumentException e)
                {
                    logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                }
4109        }
            return true;
        }
        return false;
    }
4114
    /**
     * Parses the PhoneDisplay configuration.
     *
     * @param node The root node of the PhoneDisplay configuration.
4119  * @return <code>true</code> if the node was parsed; <code>false</code> otherwise.
     */
    private boolean parsePhoneDisplay(Node node)
    {
        if (CBRConfigurationTags.DISPLAY_TITLE_SIM_ELEM.equals(node.getNodeName()))
4124    {
            TextSimilarity sim = parseTextSimilarity(node);
            if (sim != null)
            {
                PhoneDisplay.setTitleSimilarity(sim);
4129        }
            return true;
        }
        else if (CBRConfigurationTags.DISPLAY_TITLE_WEIGHT_ELEM.equals(node.getNodeName()))
        {
4134        double weight = parseDouble(node.getFirstChild());
            if (weight > -1)
            {
                try
                {
4139                PhoneDisplay.setTitleWeight(weight);
                }
                catch (IllegalArgumentException e)
                {
                    logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
4144            }
            }
            return true;
        }
        else if (CBRConfigurationTags.DISPLAY_TYPE_WEIGHT_ELEM.equals(node.getNodeName()))
4149    {
            double weight = parseDouble(node.getFirstChild());
            if (weight > -1)
            {
                try
4154            {
                    PhoneDisplay.setScreenTypeWeight(weight);
                }
                catch (IllegalArgumentException e)
                {
4159                logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                }
            }
            return true;
        }
4164    else if (CBRConfigurationTags.MIN_DISPLAY_TYPE_SIM_ELEM.equals(node.getNodeName()))
        {
            double min = parseDouble(node.getFirstChild());
            if (min > -1)
            {
4169            try
                {
```

```
                    PhoneDisplay.setMinScreenTypeSimilarity(min);
                }
                catch (IllegalArgumentException e)
4174            {
                    logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                }
            }
            return true;
4179        }
        else if (CBRConfigurationTags.DISPLAY_ITEMS_WEIGHT_ELEM.equals(node.getNodeName()))
        {
            double weight = parseDouble(node.getFirstChild());
            if (weight > -1)
4184        {
                try
                {
                    PhoneDisplay.setScreenItemsWeight(weight);
                }
4189            catch (IllegalArgumentException e)
                {
                    logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                }
            }
4194        return true;
        }

        return false;
    }
4199
    /**
     * Parses the TextItem configuration.
     *
     * @param root The root node of the TextItem configuration.
4204 * @return <code>true</code> if the node was parsed; <code>false</code> otherwise.
     */
    private boolean parseTextItem(Node node)
    {
        if (CBRConfigurationTags.TEXT_IS_TITLE_WEIGHT_ELEM.equals(node.getNodeName()))
4209    {
            TextSimilarity sim = parseTextSimilarity(node);

            if (sim != null)
            {
4214            TextItem.setTextSimilarity(sim);
            }
            return true;
        }
        else if (CBRConfigurationTags.TEXT_ITEM_TEXT_IS_TITLE_WEIGHT.equals(node.getNodeName()))
4219    {
            double weight = parseDouble(node.getFirstChild());
            if (weight > -1)
            {
                try
4224            {
                    TextItem.setIsTitleWeight(weight);
                }
                catch (IllegalArgumentException e)
                {
4229                logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                }
            }
            return true;
        }
4234    else if (CBRConfigurationTags.TEXT_WEIGHT_ELEM.equals(node.getNodeName()))
        {
            double weight = parseDouble(node.getFirstChild());
            if (weight > -1)
            {
4239            try
                {
                    TextItem.setTextWeight(4);
                }
                catch (IllegalArgumentException e)
```

```
4244                {
                        logger.log(Level.SEVERE, node.getNodeName() + " hasn't an valid value.", e);
                    }
                }
                return true;
4249        }

            return false;
        }

4254    /**
         * Parses a double.
         *
         * @param node The double node.
         * @return The double value of the node; or <code>-1</code> if the node is null or isn't a
4259     * valid double.
         */
        private double parseDouble(Node node)
        {
            if (node == null)
4264        {
                return -1;
            }

            try
4269        {
                return Double.parseDouble(node.getNodeValue());
            }
            catch (NumberFormatException e)
            {
4274            return -1;
            }
        }

        /**
4279     * Parses a string.
         *
         * @param node The string node.
         * @return The string value of the node; or <code>null</code> if the node is null.
         */
4284    private String parseString(Node node)
        {
            if (node == null)
            {
                return null;
4289        }

            return node.getNodeValue();
        }

4294    /**
         * Parses a number similarity.
         *
         * @param node The node that contains the number similarity.
         * @return The number similarity; or <code>null</code> if the node doesn't contain a valid
4299     * number similarity.
         */
        private NumberSimilarity parseNumberSimilarity(Node node)
        {
            Node next = node.getFirstChild();
4304
            while (next != null)
            {
                if (next.getNodeType() == Node.ELEMENT_NODE)
                {
4309                if (CBRConfigurationTags.PERCENT_TOLERANCE_SIM_ELEM.equals(next.getNodeName()))
                    {
                        Node attr = next.getAttributes().getNamedItem(
                                CBRConfigurationTags.PERCENT_TOLERANCE_SIM_TOLERANCE_ATTR);

4314                    if (attr != null)
                        {
                            double tolerance = parseDouble(attr);
```

```
                        if (tolerance > -1)
                        {
4319                        try
                            {
                                return new PercentToleranceSim(tolerance);
                            }
                            catch (IllegalArgumentException e)
4324                        {
                                logger.log(Level.SEVERE, node.getNodeName()
                                        + " hasn't an valid value.", e);
                            }
                        }
4329                }
                }
                else if (CBRConfigurationTags.THRESHOLD_SIM.equals(next.getNodeName()))
                {
                    Node attr = next.getAttributes().getNamedItem(
4334                            CBRConfigurationTags.THRESHOLD_SIM_THRESHOLD_ATTR);

                    if (attr != null)
                    {
                        double threshold = parseDouble(attr);
4339                    if (threshold > -1)
                        {
                            try
                            {
                                return new ThresholdSim(threshold);
4344                        }
                            catch (IllegalArgumentException e)
                            {
                                logger.log(Level.SEVERE, node.getNodeName()
                                        + " hasn't an valid value.", e);
4349                        }
                        }
                    }
                }
                else
4354            {
                    logger.warning("The tag " + next.getNodeName() + " isn't recognizable from the"
                            + " parseNumberSimilarity method.");
                }
            }
4359
            next = next.getNextSibling();
        }

        return null;
4364    }

        /**
         * Parses a set similarity.
         *
4369     * @param node The node that contains the set similarity.
         * @return The set similarity; or <code>null</code> if the node doesn't contain a valid set
         * number similarity.
         */
        private SetSimilarity parseSetSimilarity(Node node)
4374    {
            Node next = node.getFirstChild();

            while (next != null)
            {
4379            if (next.getNodeType() == Node.ELEMENT_NODE)
                {
                    if (CBRConfigurationTags.CASE_INCLUSION_SIM.equals(next.getNodeName()))
                    {
                        return new CaseInclusionSim();
4384                }
                    else if (CBRConfigurationTags.INTERSECTION_SIM.equals(next.getNodeName()))
                    {
                        return new IntersectionSim();
                    }
4389                else if (CBRConfigurationTags.QUERY_INCLUSION_SIM.equals(next.getNodeName()))
```

```
                {
                    return new QueryInclusionSim();
                }
                else
                {
                    logger.warning("The tag " + next.getNodeName() + " isn't recognizable from the"
                            + " parseSetSimilarity method.");
                }
            }

            next = next.getNextSibling();
        }

        return null;
    }

    /**
     * Parses a text similarity.
     *
     * @param node The node that contains the text similarity.
     * @return The text similarity; or <code>null</code> if the node doesn't contain a valid text
     * similarity.
     */
    private TextSimilarity parseTextSimilarity(Node node)
    {
        Node next = node.getFirstChild();

        while (next != null)
        {
            if (next.getNodeType() == Node.ELEMENT_NODE)
            {
                if (CBRConfigurationTags.CASE_INSENSITIVE_SIM.equals(next.getNodeName()))
                {
                    return new CaseInsensitiveSim();
                }
                else if (CBRConfigurationTags.CASE_SENSITIVE_SIM.equals(next.getNodeName()))
                {
                    return new CaseSensitiveSim();
                }
                else
                {
                    logger.warning("The tag " + next.getNodeName() + " isn't recognizable from the"
                            + " parseTextSimilarity method.");
                }
            }

            next = next.getNextSibling();
        }

        return null;
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2007  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.configurations;
```

```
    /**
     * Class that hold the configuration tags for the PCBRSystem.
     */
4464 public class CBRConfigurationTags
    {
        /**
         * The configuration root element.
         */
4469     public static final String PCBR_ROOT_ELEM = "pcbr";

        /**
         * The global configuration element.
         */
4474     public static final String GLOBAL_ELEM = "global";

        /**
         * The local configuration element.
         */
4479     public static final String LOCAL_ELEM = "local";

        /**
         * The phone displays set similarity element.
         */
4484     public static final String PHONE_DISPLAYS_SIM_ELEM = "phone_displays_similarity";

        /**
         * The phone displays set weight element.
         */
4489     public static final String PHONE_DISPLAYS_WEIGHT_SIM = "phone_displays_weight";

        /**
         * The phone keys set similarity element.
         */
4494     public static final String PHONE_KEYS_SIM_ELEM = "phone_keys_similarity";

        /**
         * The phone keys set weight element.
         */
4499     public static final String PHONE_KEYS_WEIGHT_ELEM = "phone_keys_weight";

        /**
         * The initial phone display weight element.
         */
4504     public static final String INITIAL_DISPLAY_WEIGHT_ELEM = "initial_display_weight";

        /**
         * The final phone display weight element.
         */
4509     public static final String FINAL_DISPLAY_WEIGHT_ELEM = "final_display_weight";

        /**
         * The transitions phone display weight element.
         */
4514     public static final String TRANSITIONS_WEIGHT_ELEM = "transitions_weight";

        /**
         * The case base directory to search for cases when building the case base element.
         */
4519     public static final String CASE_BASE_DIRECTORY_ELEM = "case_base_directory";

        /**
         * Whether the case base directory needs to be searched recursively for cases when building the
         * case base attribute.
4524     */
        public static final String CASE_BASE_DIRECTORY_RECURSIVE_ATTR = "recursive";

        /**
         * The minimum indexing percentage of a phone key group on a case in order to index it under
4529     * that group element.
         */
        public static final String MIN_INDEX_PHONEKEY_ELEM = "min_index_phonekey";

        /**
```

```
4534        * The minimum indexing percentage of a phone display group on a case in order to index it under
            * that group element.
            */
           public static final String MIN_INDEX_PHONEDISPLAY_ELEM = "min_index_phonedisplay";

4539       /**
            * The minimum indexing percentage of a utility function group on a case in order to index it
            * under that group element.
            */
           public static final String MIN_INDEX_UTILITYFUNCTION_ELEM = "min_index_utilityfunction";
4544
           /**
            * The minimum similarity to consider a solution valid element.
            */
           public static final String MIN_SOLUTION_SIM_ELEM = "min_solution_sim";
4549
           /**
            * The time held number comparison strategy element.
            */
           public static final String TIME_HELD_SIM_ELEM = "time_held_sim";
4554
           /**
            * The time held similarity weight on local key press event similarity element.
            */
           public static final String TIME_HELD_WEIGHT_ELEM = "time_held_weight";
4559
           /**
            * The phone key similarity weight on local key press event similarity element.
            */
           public static final String PHONE_KEY_WEIGHT_ELEM = "phone_key_weight";
4564
           /**
            * The phone display title text comparison strategy element.
            */
           public static final String DISPLAY_TITLE_SIM_ELEM = "display_title_sim";
4569
           /**
            * The title similarity weight on local phone display similarity element.
            */
           public static final String DISPLAY_TITLE_WEIGHT_ELEM = "display_title_weight";
4574
           /**
            * The screen type similarity weight on local phone display similarity element.
            */
           public static final String DISPLAY_TYPE_WEIGHT_ELEM = "display_type_weight";
4579
           /**
            * The display type similarity to compute the display items similarity and consider them in the
            * phone display's local similarity element.
            */
4584       public static final String MIN_DISPLAY_TYPE_SIM_ELEM = "min_display_type_sim";

           /**
            * The screen items similarity weight on local phone display similarity element.
            */
4589       public static final String DISPLAY_ITEMS_WEIGHT_ELEM = "display_items_weight";

           /**
            * The text is title similarity weight on local text item similarity element.
            */
4594       public static final String TEXT_IS_TITLE_WEIGHT_ELEM = "text_sim";

           /**
            * The text is title similarity weight on local text item similarity element.
            */
4599       public static final String TEXT_ITEM_TEXT_IS_TITLE_WEIGHT = "text_is_title_weight";

           /**
            * The text similarity weight on local text item similarity element.
            */
4604       public static final String TEXT_WEIGHT_ELEM = "text_weight";

           /**
```

```
         * The PercentToleranceSim set similarity element.
         */
4609     public static final String PERCENT_TOLERANCE_SIM_ELEM = "PercentToleranceSim";

        /**
         * The PercentToleranceSim tolerance attribute.
         */
4614     public static final String PERCENT_TOLERANCE_SIM_TOLERANCE_ATTR = "tolerance";

        /**
         * The ThresholdSim number similarity element.
         */
4619     public static final String THRESHOLD_SIM = "ThresholdSim";

        /**
         * The ThresholdSim threshold attribute.
         */
4624     public static final String THRESHOLD_SIM_THRESHOLD_ATTR = "threshold";

        /**
         * The CaseInclusionSim set similarity element.
         */
4629     public static final String CASE_INCLUSION_SIM = "CaseInclusionSim";

        /**
         * The IntersectionSim set similarity element.
         */
4634     public static final String INTERSECTION_SIM = "IntersectionSim";

        /**
         * The QueryInclusionSim set similarity element.
         */
4639     public static final String QUERY_INCLUSION_SIM = "QueryInclusionSim";

        /**
         * The CaseInsensitiveSim text similarity element.
         */
4644     public static final String CASE_INSENSITIVE_SIM = "CaseInsensitiveSim";

        /**
         * The CaseSensitiveSim text similarity element.
         */
4649     public static final String CASE_SENSITIVE_SIM = "CaseSensitiveSim";

        /**
         * Constructor.
         */
4654     private CBRConfigurationTags()
        {
            // Empty.
        }
    }
```

```
    /*
4659  * Trabalho de Conclusão de Curso
      * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
      *
      * This library is free software; you can redistribute it and/or
      * modify it under the terms of the GNU Lesser General Public
4664  * License as published by the Free Software Foundation; either
      * version 2.1 of the License, or (at your option) any later version.
      *
      * This library is distributed in the hope that it will be useful,
      * but WITHOUT ANY WARRANTY; without even the implied warranty of
4669  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
      * Lesser General Public License for more details.
      *
      * You should have received a copy of the GNU Lesser General Public
      * License along with this library; if not, write to the Free Software
4674  * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
      */
    package cbr.cycle;
```

```
import java.util.logging.Logger;
4679
import cbr.casebase.caze.CBRCase;
import cbr.casebase.caze.UtilityFunction;

/**
4684 * TODO CBRCycle.java description.
 */
public abstract class CBRCycle
{
    /**
4689  * The {@link CBRCycle} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRCycle.class.getName());

    /**
4694  * The last context the cycle was into.
     */
    private CBRCycleContext context;

    /**
4699  * The minimum similarity to consider a solution valid.
     */
    private static double minimumSimilarity = 0.725;

    /**
4704  * Constructor.
     */
    public CBRCycle()
    {
        // Empty.
4709  }

    /**
     * Set the minimum similarity to consider a solution valid.
     *
4714  * @param min The minimum similarity to consider a solution valid.
     */
    public static void setMinimumSimilarity(double min)
    {
        if (min < 0 || min > 1)
4719    {
            throw new IllegalArgumentException("The minimum similarity must be greater than or "
                    + "equal 0 and lower than or equal 1. The percentage passed was: " + min + ".");
        }

4724    minimumSimilarity = min;
    }

    /**
     * Returns the minimum similarity to consider a solution valid.
4729  *
     * @return The minimum similarity to consider a solution valid.
     */
    public static double getMinimumSimilarity()
    {
4734    return minimumSimilarity;
    }

    /**
     * Solves the problem by reasoning it into the case-based reasoner.
4739  *
     * @param problem The CBRCase problem that needs to be solved.
     */
    public void solveProblem(CBRCase problem)
    {
4744    logger.entering("CBRCycle", "solveProblem", "Begin solving a problem.");

        // Initialize a new cycle context.
        context = new CBRCycleContext(problem);

4749    // Retrieve.
        doRetrieve(context);
```

```
            // Reuse
            doReuse(context);

            // Revise
            doRevise(context);

            // Retain
            doRetain(context);

            logger.exiting("CBRCycle", "solveProblem", "Done solving a problem.");
        }

        /**
         * Returns the solution of the last solved problem, or <code>null</code> if no problem was
         * solved.
         *
         * @return The solution of the last solved problem, or <code>null</code> if no problem was
         * solved.
         */
        public UtilityFunction getSolution()
        {
            if (this.context == null)
            {
                return null;
            }

            return this.context.getSolution();
        }

        /**
         * Returns the solution case of the last solved problem, or <code>null</code> if no problem
         * was solved.
         *
         * @return The solution case of the last solved problem, or <code>null</code> if no problem
         * was solved.
         */
        public CBRCase getSolutionCase()
        {
            if (this.context == null)
            {
                return null;
            }

            return this.context.getCBRCaseSolution();
        }

        /**
         * Retrieve the case or cases that are more likely to do the reasoner. Its usually divided in 4
         * sub-steps: identify features, search, initially match and select.
         *
         * @param context The CBR system context when this method is called.
         */
        protected abstract void doRetrieve(CBRCycleContext context);

        /**
         * TODO CBRCycle.doReuse() documentation.
         *
         * @param context The CBR system context when this method is called.
         */
        protected abstract void doReuse(CBRCycleContext context);

        /**
         * TODO CBRCycle.doRevise() documentation.
         *
         * @param context The CBR system context when this method is called.
         */
        protected abstract void doRevise(CBRCycleContext context);

        /**
         * TODO CBRCycle.doRetain() documentation.
         *
         * @param context The CBR system context when this method is called.
```

```
4824        */
            protected abstract void doRetain(CBRCycleContext context);
        }
```

```
        /*
         * Trabalho de Conclusão de Curso
         * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
4829     *
         * This library is free software; you can redistribute it and/or
         * modify it under the terms of the GNU Lesser General Public
         * License as published by the Free Software Foundation; either
         * version 2.1 of the License, or (at your option) any later version.
4834     *
         * This library is distributed in the hope that it will be useful,
         * but WITHOUT ANY WARRANTY; without even the implied warranty of
         * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
         * Lesser General Public License for more details.
4839     *
         * You should have received a copy of the GNU Lesser General Public
         * License along with this library; if not, write to the Free Software
         * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
         */
4844    package cbr.cycle;

        import java.util.Collections;
        import java.util.Map;
        import java.util.Set;
4849
        import cbr.casebase.caze.CBRCase;
        import cbr.casebase.caze.UtilityFunction;
        import cbr.cycle.retrieve.initiallymatch.CBRRetrieveInitiallyMatchTask;
        import cbr.cycle.retrieve.search.CBRRetrieveSearchTask;
4854
        /**
         * Responsible to hold the state of the CBR system through its execution.
         */
        public class CBRCycleContext
4859    {
            /**
             * The CBRCase that doesn't have a solution yet.
             */
            private CBRCase problemCase;
4864
            /**
             * The solution for the case.
             */
            private UtilityFunction solution;
4869
            /**
             * The CBRCase solution for the case.
             */
            private CBRCase cbrCaseSolution;
4874
            /**
             * Contains the set of CBRCases found by the {@link CBRRetrieveSearchTask}.
             */
            private Set casesFound;
4879
            /**
             * Contains the map of similarities computed by the {@link CBRRetrieveInitiallyMatchTask}.
             */
            private Map similarities;
4884
            /**
             * Constructor.
             *
             * @param caze The CBRCase problem that needs to be solved.
4889         */
            public CBRCycleContext(CBRCase caze)
            {
                this.problemCase = caze;
            }
4894
```

```
        /**
         * Sets the casesFound value.
         *
         * @param casesFound The casesFound to set.
4899     */
        public void setCasesFound(Set casesFound)
        {
            this.casesFound = casesFound;
        }
4904
        /**
         * Returns the casesFound value.
         *
         * @return Returns the casesFound.
4909     */
        public Set getCasesFound()
        {
            return Collections.unmodifiableSet(casesFound);
        }
4914
        /**
         * Returns the problemCase value.
         *
         * @return Returns the problemCase.
4919     */
        public CBRCase getProblemCase()
        {
            return problemCase;
        }
4924
        /**
         * Returns the similarities value.
         *
         * @return Returns the similarities.
4929     */
        public Map getSimilarities()
        {
            return Collections.unmodifiableMap(similarities);
        }
4934
        /**
         * Returns the solution value.
         *
         * @return Returns the solution.
4939     */
        public UtilityFunction getSolution()
        {
            return solution;
        }
4944
        /**
         * Sets the solution value.
         *
         * @param solution The solution to set.
4949     */
        public void setSolution(UtilityFunction solution)
        {
            this.solution = solution;
        }
4954
        /**
         * Sets the CBRCase solution.
         *
         * @param caze The CBRCase solution.
4959     */
        public void setCBRCaseSolution(CBRCase caze)
        {
            this.cbrCaseSolution = caze;
        }
4964
        /**
         * Returns the CBRCase solution.
         *
```

```
         * @return The CBRCase solution.
4969     */
        public CBRCase getCBRCaseSolution()
        {
            return this.cbrCaseSolution;
        }
4974
        /**
         * Sets the similarities value.
         *
         * @param similarities The similarities to set.
4979     */
        public void setSimilarities(Map similarities)
        {
            this.similarities = similarities;
        }
4984 }
```

```
4984 /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
4989 * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
4994 * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
4999 * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.cycle;

5004 import cbr.cycle.retain.CBRRetainTask;
    import cbr.cycle.retain.CBRRetainTaskImp;
    import cbr.cycle.retrieve.CBRRetrieveTask;
    import cbr.cycle.retrieve.CBRRetrieveTaskImp;
    import cbr.cycle.reuse.CBRReuseTask;
5009 import cbr.cycle.reuse.CBRReuseTaskImp;
    import cbr.cycle.revise.CBRReviseTask;
    import cbr.cycle.revise.CBRReviseTaskImp;

    /**
5014 * TODO CBRCycleImp.java description.
     */
    public class CBRCycleImp extends CBRCycle
    {
        /**
5019     * The retrieve task.
         */
        private CBRRetrieveTask retrieveTask;

        /**
5024     * The reuse task.
         */
        private CBRReuseTask reuseTask;

        /**
5029     * The revise task.
         */
        private CBRReviseTask reviseTask;

        /**
5034     * The retain task.
         */
        private CBRRetainTask retainTask;

        /**
```

```
5039        * Constructor.
            */
           public CBRCycleImp()
           {
               super();
5044
               this.retrieveTask = new CBRRetrieveTaskImp();
               this.reuseTask = new CBRReuseTaskImp();
               this.reviseTask = new CBRReviseTaskImp();
               this.retainTask = new CBRRetainTaskImp();
5049       }

           /*
            * (non-Javadoc)
            *
5054        * @see cbr.cycle.CBRCycle#doRetain()
            */
           protected void doRetain(CBRCycleContext context)
           {
               retainTask.execute(context);
5059       }

           /*
            * (non-Javadoc)
            *
5064        * @see cbr.cycle.CBRCycle#doRetrieve()
            */
           protected void doRetrieve(CBRCycleContext context)
           {
               retrieveTask.execute(context);
5069       }

           /*
            * (non-Javadoc)
            *
5074        * @see cbr.cycle.CBRCycle#doReuse()
            */
           protected void doReuse(CBRCycleContext context)
           {
               reuseTask.execute(context);
5079       }

           /*
            * (non-Javadoc)
            *
5084        * @see cbr.cycle.CBRCycle#doRevise()
            */
           protected void doRevise(CBRCycleContext context)
           {
               reviseTask.execute(context);
5089       }
       }
```

```
       /*
        * Trabalho de Conclusão de Curso
        * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
        *
5094    * This library is free software; you can redistribute it and/or
        * modify it under the terms of the GNU Lesser General Public
        * License as published by the Free Software Foundation; either
        * version 2.1 of the License, or (at your option) any later version.
        *
5099    * This library is distributed in the hope that it will be useful,
        * but WITHOUT ANY WARRANTY; without even the implied warranty of
        * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
        * Lesser General Public License for more details.
        *
5104    * You should have received a copy of the GNU Lesser General Public
        * License along with this library; if not, write to the Free Software
        * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
        */
       package cbr.cycle.retain;
5109
```

```
import java.util.logging.Logger;

import cbr.CBRTask;
import cbr.cycle.CBRCycleContext;

/**
 * TODO CBRRetainTask.java description.
 */
public abstract class CBRRetainTask implements CBRTask
{
    /**
     * The {@link CBRRetainTask} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRRetainTask.class.getName());

    /**
     * TODO CBRRetainTask.doIntegrate() documentation.
     */
    protected abstract void doIntegrate();

    /**
     * TODO CBRRetainTask.doIndex() documentation.
     */
    protected abstract void doIndex();

    /**
     * TODO CBRRetainTask.doExtract() documentation.
     */
    protected abstract void doExtract();

    /*
     * (non-Javadoc)
     *
     * @see cbr.CBRTask#execute()
     */
    public void execute(CBRCycleContext context)
    {
        logger.entering("CBRRetainTask", "execute", "Begin retaining.");
        doIntegrate();
        doIndex();
        doExtract();
        logger.exiting("CBRRetainTask", "execute", "Done retaining.");
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.retain;

import cbr.cycle.retain.extract.CBRRetainExtractTask;
import cbr.cycle.retain.extract.ExtractSolutionMethodMethod;
import cbr.cycle.retain.index.CBRRetainIndexTask;
import cbr.cycle.retain.index.CBRRetainIndexTaskImp;
import cbr.cycle.retain.integrate.CBRRetainIntegrateTask;
import cbr.cycle.retain.integrate.CBRRetainIntegrateTaskImp;

/**
```

```
     * TODO CBRRetainTaskImp.java description.
     */
    public class CBRRetainTaskImp extends CBRRetainTask
5184 {
        /**
         * TODO integrateTask documentation.
         */
        protected CBRRetainIntegrateTask integrateTask;
5189
        /**
         * TODO extractTask documentation.
         */
        protected CBRRetainExtractTask extractTask;
5194
        /**
         * TODO indexTask documentation.
         */
        protected CBRRetainIndexTask indexTask;
5199
        /**
         * Constructor.
         */
        public CBRRetainTaskImp()
5204    {
            this.integrateTask = new CBRRetainIntegrateTaskImp();
            this.extractTask = new ExtractSolutionMethodMethod();
            this.indexTask = new CBRRetainIndexTaskImp();
        }
5209
        /*
         * (non-Javadoc)
         *
         * @see cbr.cycle.retain.CBRRetainTask#doExtract()
5214     */
        protected void doExtract()
        {
            extractTask.execute(null);
        }
5219
        /*
         * (non-Javadoc)
         *
         * @see cbr.cycle.retain.CBRRetainTask#doIndex()
5224     */
        protected void doIndex()
        {
            indexTask.execute(null);
        }
5229
        /*
         * (non-Javadoc)
         *
         * @see cbr.cycle.retain.CBRRetainTask#doIntegrate()
5234     */
        protected void doIntegrate()
        {
            integrateTask.execute(null);
        }
5239 }
```

```
5239 /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
5244  * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
5249  * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
```

```
     *
     * You should have received a copy of the GNU Lesser General Public
5254 * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.cycle.retain.extract;

5259 import java.util.logging.Logger;

    import cbr.CBRTask;
    import cbr.cycle.CBRCycleContext;

5264 /**
     * TODO CBRRetainExtractTask.java description.
     */
    public abstract class CBRRetainExtractTask implements CBRTask
    {
5269     /**
         * The {@link CBRRetainExtractTask} logger.
         */
        private static final Logger logger = Logger.getLogger(CBRRetainExtractTask.class.getName());

5274     /**
         * TODO CBRRetainExtractTask.extract() documentation.
         */
        protected abstract void extract();

5279     /*
         * (non-Javadoc)
         *
         * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
         */
5284     public void execute(CBRCycleContext context)
        {
            logger.entering("CBRRetainExtractTask", "execute", "Begin extracting.");
            extract();
            logger.exiting("CBRRetainExtractTask", "execute", "Done extracting.");
5289    }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
5294 * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
5299 * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
5304 * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.cycle.retain.extract;
5309
    /**
     * TODO ExtractSolutionsMethod.java description.
     */
    public class ExtractSolutionsMethod extends CBRRetainExtractTask
5314 {
        /*
         * (non-Javadoc)
         *
         * @see cbr.cycle.retain.extract.CBRRetainExtractTask#extract()
5319     */
        protected void extract()
        {
            throw new UnsupportedOperationException("ExtractSolutionsMethod.extract()");
```

```
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.retain.index;

import java.util.logging.Logger;

import cbr.CBRTask;
import cbr.cycle.CBRCycleContext;

/**
 * TODO CBRDetermineIndexesTask.java description.
 */
public class CBRDetermineIndexesTask implements CBRTask
{
    /**
     * The {@link CBRDetermineIndexesTask} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRDetermineIndexesTask.class.getName());

    /*
     * (non-Javadoc)
     *
     * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
     */
    public void execute(CBRCycleContext context)
    {
        logger.entering("CBRDetermineIndexesTask", "execute", "Begin determining indexes task.");
        logger.exiting("CBRDetermineIndexesTask", "execute", "Done determining indexes task.");
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.retain.index;

import java.util.logging.Logger;
```

```
import cbr.CBRTask;
import cbr.cycle.CBRCycleContext;

/**
 * TODO CBRGeneralizeIndexesTask.java description.
 */
public class CBRGeneralizeIndexesTask implements CBRTask
{
    /**
     * The {@link CBRGeneralizeIndexesTask} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRGeneralizeIndexesTask.class.getName());

    /*
     * (non-Javadoc)
     *
     * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
     */
    public void execute(CBRCycleContext context)
    {
        logger.entering("CBRGeneralizeIndexesTask", "execute", "Begin generalizing indexes task.");
        logger.exiting("CBRGeneralizeIndexesTask", "execute", "Done generalizing indexes task.");
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.retain.index;

import java.util.logging.Logger;

import cbr.CBRTask;
import cbr.cycle.CBRCycleContext;

/**
 * TODO CBRRetainIndexTask.java description.
 */
public abstract class CBRRetainIndexTask implements CBRTask
{
    /**
     * The {@link CBRRetainIndexTask} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRRetainIndexTask.class.getName());

    /**
     * TODO CBRRetainIndexTask.doGeneralizeIndexes() documentation.
     */
    protected abstract void doGeneralizeIndexes();

    /**
     * TODO CBRRetainIndexTask.doDetermineIndexes() documentation.
     */
    protected abstract void doDetermineIndexes();

    /*
     * (non-Javadoc)
     *
```

```
         * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
         */
5464     public void execute(CBRCycleContext context)
        {
            logger.entering("CBRRetainIndexTask", "execute", "Begin retaining.");
            doGeneralizeIndexes();
            doDetermineIndexes();
5469        logger.exiting("CBRRetainIndexTask", "execute", "Done retaining.");
        }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
5474 *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
5479 *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
5484 *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
5489
    package cbr.cycle.retain.index;

    /**
     * TODO CBRRetainIndexTaskImp.java description.
5494 */
    public class CBRRetainIndexTaskImp extends CBRRetainIndexTask
    {
        /**
         * TODO cBRGeneralizeIndexesTask documentation.
5499     */
        private CBRGeneralizeIndexesTask cBRGeneralizeIndexesTask;

        /**
         * TODO cBRDetermineIndexesTask documentation.
5504     */
        private CBRDetermineIndexesTask cBRDetermineIndexesTask;

        /**
         * Constructor.
5509     */
        public CBRRetainIndexTaskImp()
        {
            this.cBRGeneralizeIndexesTask = new CBRGeneralizeIndexesTask();
            this.cBRDetermineIndexesTask = new CBRDetermineIndexesTask();
5514    }

        /*
         * (non-Javadoc)
         *
5519     * @see cbr.cycle.retain.index.CBRRetainIndexTask#doDetermineIndexes()
         */
        protected void doDetermineIndexes()
        {
            cBRDetermineIndexesTask.execute(null);
5524    }

        /*
         * (non-Javadoc)
         *
5529     * @see cbr.cycle.retain.index.CBRRetainIndexTask#doGeneralizeIndexes()
         */
        protected void doGeneralizeIndexes()
        {
```

```
        cBRGeneralizeIndexesTask.execute(null);
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.retain.integrate;

import java.util.logging.Logger;

import cbr.CBRTask;
import cbr.cycle.CBRCycleContext;

/**
 * TODO CBRAdjustIndexesTask.java description.
 */
public class CBRAdjustIndexesTask implements CBRTask
{
    /**
     * The {@link CBRAdjustIndexesTask} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRAdjustIndexesTask.class.getName());

    /*
     * (non-Javadoc)
     *
     * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
     */
    public void execute(CBRCycleContext context)
    {
        logger.entering("CBRAdjustIndexesTask", "execute", "Begin adjusting indexes.");
        logger.exiting("CBRAdjustIndexesTask", "execute", "Done adjusting indexes.");
    }

}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.retain.integrate;
```

```
import java.util.logging.Logger;

import cbr.CBRTask;
import cbr.cycle.CBRCycleContext;

/**
 * TODO CBRRerunProblemTask.java description.
 */
public class CBRRerunProblemTask implements CBRTask
{
    /**
     * The {@link CBRRerunProblemTask} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRRerunProblemTask.class.getName());

    /*
     * (non-Javadoc)
     *
     * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
     */
    public void execute(CBRCycleContext context)
    {
        logger.entering("CBRRerunProblemTask", "execute", "Begin rerunning the problem.");
        logger.exiting("CBRRerunProblemTask", "execute", "Done rerunning the problem.");
    }

}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.retain.integrate;

import java.util.logging.Logger;

import cbr.CBRTask;
import cbr.cycle.CBRCycleContext;

/**
 * TODO CBRRetainIntegrateTask.java description.
 */
public abstract class CBRRetainIntegrateTask implements CBRTask
{
    /**
     * The {@link CBRRetainIntegrateTask} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRRetainIntegrateTask.class.getName());

    /**
     * TODO CBRRetainIntegrateTask.doRerunProblem() documentation.
     */
    protected abstract void doRerunProblem();

    /**
     * TODO CBRRetainIntegrateTask.doUpdateGeneralKnowledge() documentation.
     */
    protected abstract void doUpdateGeneralKnowledge();
```

```
        /**
         * TODO CBRRetainIntegrateTask.doAdjustIndexes() documentation.
5674     */
        protected abstract void doAdjustIndexes();

        /*
         * (non-Javadoc)
5679     *
         * @see cbr.CBRTask#execute()
         */
        public void execute(CBRCycleContext context)
        {
5684        logger.entering("CBRRetainIntegrateTask", "execute", "Begin integrating.");
            doRerunProblem();
            doUpdateGeneralKnowledge();
            doAdjustIndexes();
            logger.exiting("CBRRetainIntegrateTask", "execute", "Done integrating.");
5689    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
5694 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
5699 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
5704 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.retain.integrate;
5709
/**
 * TODO CBRRetainIntegrateTaskImp.java description.
 */
public class CBRRetainIntegrateTaskImp extends CBRRetainIntegrateTask
5714 {
    /**
     * TODO rerunProblemTask documentation.
     */
    private CBRRerunProblemTask rerunProblemTask;
5719
    /**
     * TODO updateGeneralKnowledgeTask documentation.
     */
    private CBRUpdateGeneralKnowledgeTask updateGeneralKnowledgeTask;
5724
    /**
     * TODO adjustIndexesTask documentation.
     */
    private CBRAdjustIndexesTask adjustIndexesTask;
5729
    /**
     * Constructor.
     */
    public CBRRetainIntegrateTaskImp()
5734 {
        this.rerunProblemTask = new CBRRerunProblemTask();
        this.updateGeneralKnowledgeTask = new CBRUpdateGeneralKnowledgeTask();
        this.adjustIndexesTask = new CBRAdjustIndexesTask();
    }
5739
    /*
     * (non-Javadoc)
     *
```

```
         * @see cbr.cycle.retain.integrate.CBRRetainIntegrateTask#doAdjustIndexes()
5744     */
        protected void doAdjustIndexes()
        {
            adjustIndexesTask.execute(null);
        }
5749
        /*
         * (non-Javadoc)
         *
         * @see cbr.cycle.retain.integrate.CBRRetainIntegrateTask#doRerunProblem()
5754     */
        protected void doRerunProblem()
        {
            rerunProblemTask.execute(null);
        }
5759
        /*
         * (non-Javadoc)
         *
         * @see cbr.cycle.retain.integrate.CBRRetainIntegrateTask#doUpdateGeneralKnowledge()
5764     */
        protected void doUpdateGeneralKnowledge()
        {
            updateGeneralKnowledgeTask.execute(null);
        }
5769 }
```

```
5769 /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
5774 * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
5779 * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
5784 * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.cycle.retain.integrate;

5789 import java.util.logging.Logger;

    import cbr.CBRTask;
    import cbr.cycle.CBRCycleContext;

5794 /**
     * TODO CBRUpdateGeneralKnowledgeTask.java description.
     */
    public class CBRUpdateGeneralKnowledgeTask implements CBRTask
    {
5799    /**
         * The {@link CBRUpdateGeneralKnowledgeTask} logger.
         */
        private static final Logger logger = Logger.getLogger(CBRUpdateGeneralKnowledgeTask.class
                .getName());
5804
        /*
         * (non-Javadoc)
         *
         * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
5809     */
        public void execute(CBRCycleContext context)
        {
            logger.entering("CBRUpdateGeneralKnowledgeTask", "execute",
                    "Begin updating general knowledge.");
```

```
5814            logger.exiting("CBRUpdateGeneralKnowledgeTask", "execute",
                        "Done updating general knowledge.");
        }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
5819  * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
5824  * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
5829  * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
5834  */
    package cbr.cycle.retrieve;

    import java.util.logging.Logger;

5839 import cbr.CBRTask;
    import cbr.cycle.CBRCycleContext;

    /**
     * TODO CBRRetrieveTask.java description.
5844  */
    public abstract class CBRRetrieveTask implements CBRTask
    {
        /**
         * The {@link CBRRetrieveTask} logger.
5849      */
        private static final Logger logger = Logger.getLogger(CBRRetrieveTask.class.getName());

        /**
         * Identify the features of the CBRCase.
5854      *
         * @param context The CBR system context when this method is called.
         */
        protected abstract void doIdentifyFeatures(CBRCycleContext context);

5859    /**
         * Search the CBRCaseBase.
         *
         * @param context The CBR system context when this method is called.
         */
5864    protected abstract void doSearch(CBRCycleContext context);

        /**
         * Initially match a sub-set of similar CBRCases.
         *
5869      * @param context The CBR system context when this method is called.
         */
        protected abstract void doInitiallyMatch(CBRCycleContext context);

        /**
5874      * Select the best CBRCase.
         *
         * @param context The CBR system context when this method is called.
         */
        protected abstract void doSelect(CBRCycleContext context);
5879
        /*
         * (non-Javadoc)
         *
         * @see cbr.CBRTask#execute()
5884      */
```

```
        public void execute(CBRCycleContext context)
        {
            /**
             * The identification task basically comes up with a set of relevant problem descriptors,
5889         * the goal of the matching task is to return a set of cases that are sufficiently similar
             * to the new case – given a similarity threshold of some kind, and the selection task works
             * on this set of cases and chooses the best match (or at least a first case to try out).
             */
            logger.entering("CBRRetrieveTask", "execute", "Begin retrieving.");
5894
            doIdentifyFeatures(context);

            doSearch(context);

5899        doInitiallyMatch(context);

            doSelect(context);

            logger.exiting("CBRRetrieveTask", "execute", "Done retrieving.");
5904    }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
5909 * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
5914 * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
5919 * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */

5924 package cbr.cycle.retrieve;

    import cbr.cycle.CBRCycleContext;
    import cbr.cycle.retrieve.identifyfeatures.CBRIdentifyFeaturesTask;
    import cbr.cycle.retrieve.identifyfeatures.CBRIdentifyFeaturesTaskImp;
5929 import cbr.cycle.retrieve.initiallymatch.CBRRetrieveInitiallyMatchTask;
    import cbr.cycle.retrieve.initiallymatch.CalculateSimilarityMethod;
    import cbr.cycle.retrieve.search.CBRRetrieveSearchTask;
    import cbr.cycle.retrieve.search.SearchGeneralKnowledgeMethod;
    import cbr.cycle.retrieve.select.CBRRetrieveSelectTask;
5934 import cbr.cycle.retrieve.select.UseSelectionCriteriaMethod;

    /**
     * TODO CBRRetrieveTaskImp.java description.
     */
5939 public class CBRRetrieveTaskImp extends CBRRetrieveTask
    {
        /**
         * Task responsible to search.
         */
5944    private CBRRetrieveSearchTask searchTask;

        /**
         * Task responsible to initially match.
         */
5949    private CBRRetrieveInitiallyMatchTask initiallyMatchTask;

        /**
         * Task responsible to select.
         */
5954    private CBRRetrieveSelectTask selectTask;
```

```
        /**
         * Task responsible to identify the features.
         */
5959    private CBRIdentifyFeaturesTask identifyFeaturesTask;

        /**
         * Constructor.
         */
5964    public CBRRetrieveTaskImp()
        {
            this.searchTask = new SearchGeneralKnowledgeMethod();
            this.initiallyMatchTask = new CalculateSimilarityMethod();
            this.selectTask = new UseSelectionCriteriaMethod();
5969        this.identifyFeaturesTask = new CBRIdentifyFeaturesTaskImp();
        }

        /*
         * (non-Javadoc)
5974     *
         * @see cbr.cycle.retrieve.CBRRetrieveTask#doIdentifyFeatures(cbr.cycle.CBRCycleContext)
         */
        protected void doIdentifyFeatures(CBRCycleContext context)
        {
5979        identifyFeaturesTask.execute(context);
        }

        /*
         * (non-Javadoc)
5984     *
         * @see cbr.cycle.retrieve.CBRRetrieveTask#doInitiallyMatch(cbr.cycle.CBRCycleContext)
         */
        protected void doInitiallyMatch(CBRCycleContext context)
        {
5989        initiallyMatchTask.execute(context);
        }

        /*
         * (non-Javadoc)
5994     *
         * @see cbr.cycle.retrieve.CBRRetrieveTask#doSearch(cbr.cycle.CBRCycleContext)
         */
        protected void doSearch(CBRCycleContext context)
        {
5999        searchTask.execute(context);
        }

        /*
         * (non-Javadoc)
6004     *
         * @see cbr.cycle.retrieve.CBRRetrieveTask#doSelect(cbr.cycle.CBRCycleContext)
         */
        protected void doSelect(CBRCycleContext context)
        {
6009        selectTask.execute(context);
        }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
6014 *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
6019 *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
6024 *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
```

```
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.retrieve.identifyfeatures;

import cbr.CBRTask;
import cbr.cycle.CBRCycleContext;

/**
 * TODO CBRCollectDescriptorsTask.java description.
 */
public class CBRCollectDescriptorsTask implements CBRTask
{
    /*
     * (non-Javadoc)
     *
     * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
     */
    public void execute(CBRCycleContext context)
    {
        // Empty.
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.retrieve.identifyfeatures;

import java.util.logging.Logger;

import cbr.CBRTask;
import cbr.cycle.CBRCycleContext;

/**
 * TODO CBRIdentifyFeaturesTask.java description.
 */
public abstract class CBRIdentifyFeaturesTask implements CBRTask
{
    /**
     * The {@link CBRIdentifyFeaturesTask} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRIdentifyFeaturesTask.class.getName());

    /**
     * Collect the descriptors.
     *
     * @param context
     */
    protected abstract void doCollectDescriptors(CBRCycleContext context);

    /**
     * TODO CBRIdentifyFeaturesTask.doInterpretProblem() documentation.
     *
     * @param context
     */
    protected abstract void doInterpretProblem(CBRCycleContext context);

    /**
```

```
         * TODO CBRIdentifyFeaturesTask.doInferDescriptors() documentation.
6099     *
         * @param context
         */
        protected abstract void doInferDescriptors(CBRCycleContext context);

6104    /*
         * (non-Javadoc)
         *
         * @see cbr.CBRTask#execute()
         */
6109    public void execute(CBRCycleContext context)
        {
            logger.entering("CBRIdentifyFeaturesTask", "execute", "Begin identifying features.");
            doCollectDescriptors(context);
            doInterpretProblem(context);
6114        doInferDescriptors(context);
            logger.exiting("CBRIdentifyFeaturesTask", "execute", "Done identifying features.");
        }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
6119 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
6124 * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
6129 * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
6134 */
    package cbr.cycle.retrieve.identifyfeatures;

    import cbr.cycle.CBRCycleContext;

6139 /**
     * TODO CBRIdentifyFeaturesTaskImp.java description.
     */
    public class CBRIdentifyFeaturesTaskImp extends CBRIdentifyFeaturesTask
    {
6144    /**
         * Task responsible to collect the descriptors.
         */
        private CBRCollectDescriptorsTask collectDescriptorsTask;

6149    /**
         * Task responsible to interpret the problem.
         */
        private CBRInterpretProblemTask interpretProblemTask;

6154    /**
         * Task responsible to infer the descriptors.
         */
        private CBRInferDescriptorsTask inferDescriptorsTask;

6159    /**
         * Constructor.
         */
        public CBRIdentifyFeaturesTaskImp()
        {
6164        this.collectDescriptorsTask = new CBRCollectDescriptorsTask();
            this.interpretProblemTask = new CBRInterpretProblemTask();
            this.inferDescriptorsTask = new CBRInferDescriptorsTask();
        }
```

```
6169        /*
             * (non-Javadoc)
             *
             * @see cbr.cycle.retrieve.identifyfeatures.CBRIdentifyFeaturesTask#doCollectDescriptors(cbr.cycle.
                 CBRCycleContext)
             */
6174        protected void doCollectDescriptors(CBRCycleContext context)
            {
                collectDescriptorsTask.execute(context);
            }

6179        /*
             * (non-Javadoc)
             *
             * @see cbr.cycle.retrieve.identifyfeatures.CBRIdentifyFeaturesTask#doInferDescriptors(cbr.cycle.
                 CBRCycleContext)
             */
6184        protected void doInferDescriptors(CBRCycleContext context)
            {
                inferDescriptorsTask.execute(context);
            }

6189        /*
             * (non-Javadoc)
             *
             * @see cbr.cycle.retrieve.identifyfeatures.CBRIdentifyFeaturesTask#doInterpretProblem(cbr.cycle.
                 CBRCycleContext)
             */
6194        protected void doInterpretProblem(CBRCycleContext context)
            {
                interpretProblemTask.execute(context);
            }
        }
```

```
        /*
6199     * Trabalho de Conclusão de Curso
         * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
         *
         * This library is free software; you can redistribute it and/or
         * modify it under the terms of the GNU Lesser General Public
6204     * License as published by the Free Software Foundation; either
         * version 2.1 of the License, or (at your option) any later version.
         *
         * This library is distributed in the hope that it will be useful,
         * but WITHOUT ANY WARRANTY; without even the implied warranty of
6209     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
         * Lesser General Public License for more details.
         *
         * You should have received a copy of the GNU Lesser General Public
         * License along with this library; if not, write to the Free Software
6214     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
         */
        package cbr.cycle.retrieve.identifyfeatures;

        import cbr.CBRTask;
6219    import cbr.cycle.CBRCycleContext;

        /**
         * TODO CBRInferDescriptorsTask.java description.
         */
6224    public class CBRInferDescriptorsTask implements CBRTask
        {
            /*
             * (non-Javadoc)
             *
6229         * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
             */
            public void execute(CBRCycleContext context)
            {
                // Empty.
6234        }
        }
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.retrieve.identifyfeatures;

import cbr.CBRTask;
import cbr.cycle.CBRCycleContext;

/**
 * TODO CBRInterpretProblemTask.java description.
 */
public class CBRInterpretProblemTask implements CBRTask
{
    /*
     * (non-Javadoc)
     *
     * @see cbr.CBRTask#execute()
     */
    public void execute(CBRCycleContext context)
    {
        // Empty.
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.retrieve.initiallymatch;

import java.util.logging.Logger;

import cbr.CBRTask;
import cbr.cycle.CBRCycleContext;

/**
 * TODO CBRRetrieveInitiallyMatchTask.java description.
 */
public abstract class CBRRetrieveInitiallyMatchTask implements CBRTask
{
    /**
     * The {@link CBRRetrieveInitiallyMatchTask} logger.
     */
```

```
        private static final Logger logger = Logger.getLogger(CBRRetrieveInitiallyMatchTask.class
                .getName());

        /**
6309     * Initially match the cases found.
         */
        protected abstract void initiallyMatch(CBRCycleContext context);

        /*
6314     * (non-Javadoc)
         *
         * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
         */
        public void execute(CBRCycleContext context)
6319    {
            logger.entering("CBRRetrieveInitiallyMatchTask", "execute", "Begin initially match.");
            initiallyMatch(context);
            logger.exiting("CBRRetrieveInitiallyMatchTask", "execute", "Done initially match.");
        }
6324 }
```

```
6324 /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
6329 * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
6334 * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
6339 * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.cycle.retrieve.initiallymatch;

6344 import java.util.*;
    import java.util.logging.Logger;

    import cbr.casebase.caze.CBRCase;
    import cbr.cycle.CBRCycleContext;
6349
    /**
     * TODO CalculateSimilarityMethod.java description.
     */
    public class CalculateSimilarityMethod extends CBRRetrieveInitiallyMatchTask
6354 {
        /**
         * The {@link CalculateSimilarityMethod} logger.
         */
        private static final Logger logger = Logger
6359            .getLogger(CalculateSimilarityMethod.class.getName());

        /**
         * The descending comparator for comparable objects.
         */
6364    private static final Comparator DESCENDING_COMPARATOR = new Comparator()
        {
            public int compare(Object arg0, Object arg1)
            {
                return ((Comparable) arg1).compareTo(arg0);
6369        }
        };

        /**
         * Holds all similarities between the cases found and the problem case.
6374     */
        private Map similarities;
```

```
        /*
         * (non-Javadoc)
6379     *
         * @see cbr.cycle.retrieve.initiallymatch.CBRRetrieveInitiallyMatchTask#initiallyMatch(cbr.cycle.
             CBRCycleContext)
         */
        protected synchronized void initiallyMatch(CBRCycleContext context)
        {
6384        logger.entering("CalculateSimilarityMethod", "initiallyMatch",
                    "Begin calculate similarity.");

            // Retrieve the problem case.
            CBRCase problem = context.getProblemCase();
6389
            // Retrieve the cases found by the search found.
            Set cases = context.getCasesFound();

            similarities = new TreeMap(DESCENDING_COMPARATOR);
6394
            CBRCase current;
            for (Iterator iter = cases.iterator(); iter.hasNext(); /* empty */)
            {
                current = (CBRCase) iter.next();
6399            addSimilarity(current, problem.similarityTo(current));
            }

            context.setSimilarities(similarities);

6404        logger
                    .entering("CalculateSimilarityMethod", "initiallyMatch",
                            "Done calculate similarity.");
        }

6409    private void addSimilarity(CBRCase caze, double similarity)
        {
            Double key = new Double(similarity);

            List cases = (List) similarities.get(key);
6414
            if (cases == null)
            {
                cases = new ArrayList();
            }
6419
            cases.add(caze);
            similarities.put(key, cases);
        }
    }
```

```
    /*
6424 * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
6429 * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
6434 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
6439 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.cycle.retrieve.search;

    import java.util.logging.Logger;
6444
    import cbr.CBRTask;
```

```
import cbr.cycle.CBRCycleContext;

/**
 * TODO CBRRetrieveSearchTask.java description.
 */
public abstract class CBRRetrieveSearchTask implements CBRTask
{
    /**
     * The {@link CBRRetrieveSearchTask} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRRetrieveSearchTask.class.getName());

    /**
     * Search the case base.
     */
    protected abstract void search(CBRCycleContext context);

    /*
     * (non-Javadoc)
     *
     * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
     */
    public void execute(CBRCycleContext context)
    {
        logger.entering("CBRRetrieveSearchTask", "execute", "Begin searching.");
        search(context);
        logger.exiting("CBRRetrieveSearchTask", "execute", "Done searching.");
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.retrieve.search;

import java.util.logging.Logger;

import cbr.casebase.CBRCaseBase;
import cbr.cycle.CBRCycleContext;

/**
 * TODO SearchGeneralKnowledgeMethod.java description.
 */
public class SearchGeneralKnowledgeMethod extends CBRRetrieveSearchTask
{
    /**
     * The {@link SearchGeneralKnowledgeMethod} logger.
     */
    private static final Logger logger = Logger.getLogger(SearchGeneralKnowledgeMethod.class
            .getName());

    /*
     * (non-Javadoc)
     *
     * @see cbr.cycle.retrieve.search.CBRRetrieveSearchTask#search(cbr.cycle.CBRCycleContext)
     */
    protected void search(CBRCycleContext context)
    {
```

```
            logger.entering("SearchGeneralKnowledgeMethod", "search",
                    "Begin searching general knowledge.");

            // Let the case base decide what's close enough.
            context.setCasesFound(CBRCaseBase.getInstance().getCBRCases(context.getProblemCase()));

            logger.exiting("SearchGeneralKnowledgeMethod", "search",
                    "Done searching general knowledge.");
        }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.cycle.retrieve.select;

    import java.util.logging.Logger;

    import cbr.CBRTask;
    import cbr.cycle.CBRCycleContext;

    /**
     * TODO CBRRetrieveSelectTask.java description.
     */
    public abstract class CBRRetrieveSelectTask implements CBRTask
    {
        /**
         * The {@link CBRRetrieveSelectTask} logger.
         */
        private static final Logger logger = Logger.getLogger(CBRRetrieveSelectTask.class.getName());

        /**
         * Select the best case.
         */
        protected abstract void select(CBRCycleContext context);

        /*
         * (non-Javadoc)
         *
         * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
         */
        public void execute(CBRCycleContext context)
        {
            logger.entering("CBRRetrieveSelectTask", "execute", "Begin selecting.");
            select(context);
            logger.exiting("CBRRetrieveSelectTask", "execute", "Done selecting.");
        }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
```

```
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.retrieve.select;

import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.logging.Logger;

import cbr.casebase.caze.CBRCase;
import cbr.cycle.CBRCycle;
import cbr.cycle.CBRCycleContext;

/**
 * TODO UseSelectionCriteriaMethod.java description.
 */
public class UseSelectionCriteriaMethod extends CBRRetrieveSelectTask
{
    /**
     * The {@link UseSelectionCriteriaMethod} logger.
     */
    private static final Logger logger = Logger.getLogger(UseSelectionCriteriaMethod.class
            .getName());

    /*
     * (non-Javadoc)
     *
     * @see cbr.cycle.retrieve.select.CBRRetrieveSelectTask#select(cbr.cycle.CBRCycleContext)
     */
    protected synchronized void select(CBRCycleContext context)
    {
        logger.entering("UseSelectionCriteriaMethod", "select", "Begin using selection criteria.");

        // Retrieve the similarities computed.
        Map similarites = context.getSimilarities();

        // Holds the list of best cases.
        List bestCases = null;

        // Holds the best case.
        double similarityOfTheBest = 0;

        for (Iterator iter = similarites.keySet().iterator(); iter.hasNext(); /* empty */)
        {
            Double key = (Double) iter.next();

            similarityOfTheBest = key.doubleValue();

            bestCases = (List) similarites.get(key);

            // We're only interested in the first one.
            break;
        }

        CBRCase bestCase;
        if (bestCases != null && similarityOfTheBest > CBRCycle.getMinimumSimilarity())
        {
            bestCase = (CBRCase)bestCases.get(0);
            context.setSolution(bestCase.getUtilityFuncion());
            context.setCBRCaseSolution(bestCase);
        }

        logger.exiting("UseSelectionCriteriaMethod", "select", "Done using selection criteria.");
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.reuse;

import java.util.logging.Logger;

import cbr.CBRTask;
import cbr.cycle.CBRCycleContext;

/**
 * TODO CBRReuseTask.java description.
 */
public abstract class CBRReuseTask implements CBRTask
{
    /**
     * The {@link CBRReuseTask} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRReuseTask.class.getName());

    /**
     * TODO CBRReuseTask.doCopy() documentation.
     */
    protected abstract void doCopy();

    /**
     * TODO CBRReuseTask.doAdapt() documentation.
     */
    protected abstract void doAdapt();

    /*
     * (non-Javadoc)
     *
     * @see cbr.CBRTask#execute()
     */
    public void execute(CBRCycleContext context)
    {
        logger.entering("CBRReuseTask", "execute", "Begin reusing.");
        doCopy();
        doAdapt();
        logger.exiting("CBRReuseTask", "execute", "Done reusing.");
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
```

```
 *
 * You should have received a copy of the GNU Lesser General Public
6729 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.reuse;

6734 import cbr.cycle.reuse.adapt.CBRReuseAdaptTask;
import cbr.cycle.reuse.adapt.ModifySolutionMethod;
import cbr.cycle.reuse.copy.CBRReuseCopyTask;
import cbr.cycle.reuse.copy.CopySolutionMethod;

6739 /**
 * TODO CBRReuseTaskImp.java description.
 */
public class CBRReuseTaskImp extends CBRReuseTask
{
6744     /**
      * TODO copyTask documentation.
      */
     protected CBRReuseCopyTask copyTask;

6749     /**
      * TODO adaptTask documentation.
      */
     protected CBRReuseAdaptTask adaptTask;

6754     /**
      * Constructor.
      */
     public CBRReuseTaskImp()
     {
6759         this.copyTask = new CopySolutionMethod();
         this.adaptTask = new ModifySolutionMethod();
     }

     /*
6764  * (non-Javadoc)
      *
      * @see cbr.cycle.reuse.CBRReuseTask#doAdapt()
      */
     protected void doAdapt()
6769 {
         adaptTask.execute(null);
     }

     /*
6774  * (non-Javadoc)
      *
      * @see cbr.cycle.reuse.CBRReuseTask#doCopy()
      */
     protected void doCopy()
6779 {
         copyTask.execute(null);
     }
}
```

```
/*
 * Trabalho de Conclusão de Curso
6784 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
6789 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
6794 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
```

```
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
6799  */

    package cbr.cycle.reuse.adapt;

    import java.util.logging.Logger;
6804
    import cbr.CBRTask;
    import cbr.cycle.CBRCycleContext;

    /**
6809  * TODO CBRReuseAdaptTask.java description.
     */
    public abstract class CBRReuseAdaptTask implements CBRTask
    {
        /**
6814     * The {@link CBRReuseAdaptTask} logger.
         */
        private static final Logger logger = Logger.getLogger(CBRReuseAdaptTask.class.getName());

        /**
6819     * TODO CBRReuseAdaptTask.adapt() documentation.
         */
        protected abstract void adapt();

        /*
6824     * (non-Javadoc)
         *
         * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
         */
        public void execute(CBRCycleContext context)
6829    {
            logger.entering("CBRReuseAdaptTask", "execute", "Begin adapting.");
            adapt();
            logger.exiting("CBRReuseAdaptTask", "execute", "Done adapting.");
        }
6834 }
```

```
6834 /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
6839  * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
6844  * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
6849  * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.cycle.reuse.adapt;

6854 import java.util.logging.Logger;

    /**
     * TODO ModifySolutionMethod.java description.
     */
6859 public class ModifySolutionMethod extends CBRReuseAdaptTask
    {
        /**
         * The {@link ModifySolutionMethod} logger.
         */
6864    private static final Logger logger = Logger.getLogger(ModifySolutionMethod.class.getName());

        /*
         * (non-Javadoc)
         *
```

```
6869        * @see cbr.cycle.reuse.adapt.CBRReuseAdaptTask#adapt()
            */
        protected void adapt()
        {
            logger.entering("ModifySolutionMethod", "adapt", "Begin modifying solution.");
6874        logger.exiting("ModifySolutionMethod", "adapt", "Done modifying solution.");
        }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
6879 *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
6884 *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
6889 *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
6894 package cbr.cycle.reuse.copy;

    import java.util.logging.Logger;

    import cbr.CBRTask;
6899 import cbr.cycle.CBRCycleContext;

    /**
     * TODO CBRReuseCopyTask.java description.
     */
6904 public abstract class CBRReuseCopyTask implements CBRTask
    {
        /**
         * The {@link CBRReuseCopyTask} logger.
         */
6909    private static final Logger logger = Logger.getLogger(CBRReuseCopyTask.class.getName());

        /**
         * TODO CBRReuseCopyTask.copy() documentation.
         */
6914    protected abstract void copy();

        /*
         * (non-Javadoc)
         *
6919     * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
         */
        public void execute(CBRCycleContext context)
        {
            logger.entering("CBRReuseCopyTask", "execute", "Begin copying.");
6924        copy();
            logger.exiting("CBRReuseCopyTask", "execute", "Done copying.");
        }
    }
```

```
    /*
     * Trabalho de Conclusão de Curso
6929 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
6934 * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
```

```
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
6939 * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
6944 */
    package cbr.cycle.reuse.copy;

    import java.util.logging.Logger;

6949 /**
     * TODO CopySolutionMethod.java description.
     */
    public class CopySolutionMethod extends CBRReuseCopyTask
    {
6954    /**
         * The {@link CopySolutionMethod} logger.
         */
        private static final Logger logger = Logger.getLogger(CopySolutionMethod.class.getName());

6959    /*
         * (non-Javadoc)
         *
         * @see cbr.cycle.reuse.copy.CBRReuseCopyTask#copy()
         */
6964    protected void copy()
        {
            logger.entering("CopySolutionMethod", "copy", "Begin copying solution.");
            logger.exiting("CopySolutionMethod", "copy", "Done copying solution.");
        }
6969 }
```

```
6969 /*
     * Trabalho de Conclusão de Curso
     * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
6974 * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
6979 * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
6984 * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.cycle.revise;

6989 import java.util.logging.Logger;

    import cbr.CBRTask;
    import cbr.cycle.CBRCycleContext;

6994 /**
     * TODO CBRReviseTask.java description.
     */
    public abstract class CBRReviseTask implements CBRTask
    {
6999    /**
         * The {@link CBRReviseTask} logger.
         */
        private static final Logger logger = Logger.getLogger(CBRReviseTask.class.getName());

7004    /**
         * TODO CBRReviseTask.doEvaluateSolution() documentation.
         */
        protected abstract void doEvaluateSolution();
```

```
7009        /**
             * TODO CBRReviseTask.doRepairFault() documentation.
             */
            protected abstract void doRepairFault();

7014        /*
             * (non-Javadoc)
             *
             * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
             */
7019        public void execute(CBRCycleContext context)
            {
                logger.entering("CBRReviseTask", "execute", "Begin revising.");
                doEvaluateSolution();
                doRepairFault();
7024            logger.exiting("CBRReviseTask", "execute", "Done revising.");
            }
        }
```

```
        /*
         * Trabalho de Conclusão de Curso
         * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
7029     *
         * This library is free software; you can redistribute it and/or
         * modify it under the terms of the GNU Lesser General Public
         * License as published by the Free Software Foundation; either
         * version 2.1 of the License, or (at your option) any later version.
7034     *
         * This library is distributed in the hope that it will be useful,
         * but WITHOUT ANY WARRANTY; without even the implied warranty of
         * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
         * Lesser General Public License for more details.
7039     *
         * You should have received a copy of the GNU Lesser General Public
         * License along with this library; if not, write to the Free Software
         * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
         */
7044    package cbr.cycle.revise;

        import cbr.cycle.revise.evaluatesolution.CBRReviseEvaluateSolutionTask;
        import cbr.cycle.revise.evaluatesolution.EvaluateByTeacherMethod;
        import cbr.cycle.revise.repairfault.CBRReviseRepairFaultTask;
7049    import cbr.cycle.revise.repairfault.UserRepairMethod;

        /**
         * TODO CBRReviseTaskImp.java description.
         */
7054    public class CBRReviseTaskImp extends CBRReviseTask
        {
            /**
             * TODO evaluateSolutionTask documentation.
             */
7059        private CBRReviseEvaluateSolutionTask evaluateSolutionTask;

            /**
             * TODO repairFaultTask documentation.
             */
7064        private CBRReviseRepairFaultTask repairFaultTask;

            /**
             * Constructor.
             */
7069        public CBRReviseTaskImp()
            {
                this.evaluateSolutionTask = new EvaluateByTeacherMethod();
                this.repairFaultTask = new UserRepairMethod();
            }
7074
            /*
             * (non-Javadoc)
             *
             * @see cbr.cycle.revise.CBRReviseTask#doEvaluateSolution()
```

```
7079        */
            protected void doEvaluateSolution()
            {
                evaluateSolutionTask.execute(null);
            }
7084
            /*
             * (non-Javadoc)
             *
             * @see cbr.cycle.revise.CBRReviseTask#doRepairFault()
7089         */
            protected void doRepairFault()
            {
                repairFaultTask.execute(null);
            }
7094    }
```

```
7094    /*
         * Trabalho de Conclusão de Curso
         * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
         *
         * This library is free software; you can redistribute it and/or
7099     * modify it under the terms of the GNU Lesser General Public
         * License as published by the Free Software Foundation; either
         * version 2.1 of the License, or (at your option) any later version.
         *
         * This library is distributed in the hope that it will be useful,
7104     * but WITHOUT ANY WARRANTY; without even the implied warranty of
         * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
         * Lesser General Public License for more details.
         *
         * You should have received a copy of the GNU Lesser General Public
7109     * License along with this library; if not, write to the Free Software
         * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
         */
        package cbr.cycle.revise.evaluatesolution;

7114    import java.util.logging.Logger;

        import cbr.CBRTask;
        import cbr.cycle.CBRCycleContext;

7119    /**
         * TODO CBRReviseEvaluateSolutionTask.java description.
         */
        public abstract class CBRReviseEvaluateSolutionTask implements CBRTask
        {
7124        /**
             * The {@link CBRReviseEvaluateSolutionTask} logger.
             */
            private static final Logger logger = Logger.getLogger(CBRReviseEvaluateSolutionTask.class
                    .getName());
7129
            /**
             * TODO CBRReviseEvaluateSolutionTask.evaluateSolution() documentation.
             */
            protected abstract void evaluateSolution();
7134
            /*
             * (non-Javadoc)
             *
             * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
7139         */
            public void execute(CBRCycleContext context)
            {
                logger.entering("CBRReviseEvaluateSolutionTask", "execute", "Begin evaluating solution.");
                evaluateSolution();
7144            logger.exiting("CBRReviseEvaluateSolutionTask", "execute", "Done evaluating solution.");
            }
        }
```

```
        /*
```

```
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.revise.evaluatesolution;

import java.util.logging.Logger;

/**
 * TODO EvaluateByTeacherMethod.java description.
 */
public class EvaluateByTeacherMethod extends CBRReviseEvaluateSolutionTask
{
    /**
     * The {@link EvaluateByTeacherMethod} logger.
     */
    private static final Logger logger = Logger.getLogger(EvaluateByTeacherMethod.class.getName());

    /*
     * (non-Javadoc)
     *
     * @see cbr.cycle.revise.evaluatesolution.CBRReviseEvaluateSolutionTask#evaluateSolution()
     */
    protected void evaluateSolution()
    {
        logger.entering("EvaluateByTeacherMethod", "evaluateSolution",
                "Begin evaluating by teacher.");
        logger
                .exiting("EvaluateByTeacherMethod", "evaluateSolution",
                        "Done evaluating by teacher.");
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.revise.repairfault;

import java.util.logging.Logger;

import cbr.CBRTask;
import cbr.cycle.CBRCycleContext;

/**
 * TODO CBRReviseRepairFaultTask.java description.
```

```
 */
public abstract class CBRReviseRepairFaultTask implements CBRTask
{
    /**
     * The {@link CBRReviseRepairFaultTask} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRReviseRepairFaultTask.class.getName());

    /**
     * TODO CBRReviseRepairFaultTask.repairFault() documentation.
     */
    protected abstract void repairFault();

    /*
     * (non-Javadoc)
     *
     * @see cbr.CBRTask#execute(cbr.cycle.CBRCycleContext)
     */
    public void execute(CBRCycleContext context)
    {
        logger.entering("CBRReviseRepairFaultTask", "execute", "Begin repairing fault.");
        repairFault();
        logger.exiting("CBRReviseRepairFaultTask", "execute", "Done repairing fault.");
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.cycle.revise.repairfault;

import java.util.logging.Logger;

/**
 * TODO UserRepairMethod.java description.
 */
public class UserRepairMethod extends CBRReviseRepairFaultTask
{
    /**
     * The {@link UserRepairMethod} logger.
     */
    private static final Logger logger = Logger.getLogger(UserRepairMethod.class.getName());

    /*
     * (non-Javadoc)
     *
     * @see cbr.cycle.revise.repairfault.CBRReviseRepairFaultTask#repairFault()
     */
    protected void repairFault()
    {
        logger.entering("UserRepairMethod", "repairFault", "Begin user repairing.");
        logger.exiting("UserRepairMethod", "repairFault", "Done user repairing.");
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
```

```
     * Copyright (C) 2007  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
7289 * modify it under the terms of the GNU Lesser General Public
     * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
7294 * but WITHOUT ANY WARRANTY; without even the implied warranty of
     * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
7299 * License along with this library; if not, write to the Free Software
     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.parsers;

7304 import java.io.File;
    import java.util.ArrayList;
    import java.util.Iterator;
    import java.util.List;
    import java.util.logging.Logger;
7309
    import cbr.runner.CBRBarrier;

    import com.motorola.taflogger.events.*;
    import com.motorola.taflogger.exceptions.ParseException;
7314 import com.motorola.taflogger.logmanagement.LogSession;
    import com.motorola.taflogger.visitors.LogEventVisitor;

    /**
     * TODO LogSessionToLogEventsList.java description.
7319 */
    public class LogSessionBuilder extends Thread
    {
        /**
         * The {@link LogSessionBuilder} logger.
7324     */
        private static final Logger logger = Logger.getLogger(LogSessionBuilder.class.getName());

        /**
         * The log session file name.
7329     */
        private String logSessionFileName;

        /**
         * The log session itself.
7334     */
        private LogSession logSession;

        /**
         * The log events list.
7339     */
        private List logEvents;

        /**
         * The barrier to wait after has finished.
7344     */
        private CBRBarrier barrier;

        /**
         * Constructor.
7349     *
         * @param fileName The log session file name.
         * @param barrier The barrier to hit when finish.
         */
        public LogSessionBuilder(String fileName, CBRBarrier barrier)
7354    {
            super("Log session builder of " + fileName);
            this.logSessionFileName = fileName;
            this.barrier = barrier;
        }
```

```
7359
        /**
         * Returns the log events.
         *
         * @return Returns the log events.
7364     */
        public List getLogEvents()
        {
            return logEvents;
        }
7369
        /*
         * (non-Javadoc)
         *
         * @see java.lang.Thread#run()
7374     */
        public void run()
        {
            long init = System.currentTimeMillis();

7379        logger.fine("Building the log session.");
            buildLogSession();

            logger.fine("Building the log event list.");
            buildLogEventList();
7384
            logger.fine("Log events were parsed in " + (System.currentTimeMillis() - init) + " ms.");
            barrier.waitForRealese();
        }

7389    /**
         * Builds the log session based on the file.
         */
        private void buildLogSession()
        {
7394        try
            {
                File logSessionFile = new File(this.logSessionFileName);
                logSession = new LogSession(logSessionFile);
            }
7399        catch (ParseException e)
            {
                throw new RuntimeException("Error building the log session", e);
            }
        }
7404
        /**
         * Builds the log event list based on the log session.
         */
        private void buildLogEventList()
7409    {
            LogEventListVisitor visitor = new LogEventListVisitor();
            this.logSession.getEventList().getRootGroup().accept(visitor);

            this.logEvents = visitor.getLogEvents();
7414    }

        /**
         * Visits log events and hold them in a list. The hold events are the {@link KeyPressEvent} and
         * the {@link PhoneScreenEvent} (all other events are just ignored). The log events are
7419     * 'normalized', i.e., if two KeyPressEvents are found consecutively, the last known
         * PhoneScreenEvent if added between the two key press events.
         */
        private static class LogEventListVisitor implements LogEventVisitor
        {
7424        /**
             * Holds the last parsed screen event.
             */
            private PhoneScreenEvent lastPhoneScreenEvent;

7429        /**
             * Flag stating if the last parsed event was an phone screen event.
             */
```

```
        boolean wasPhoneScreenEvent;

7434    /**
         * The list containing the log events.
         */
        private List logEvents;

7439    /**
         * Constructor.
         */
        public LogEventListVisitor()
        {
7444        this.lastPhoneScreenEvent = null;
            this.wasPhoneScreenEvent = false;
            this.logEvents = new ArrayList(100);
        }

7449    /**
         * Returns the log events value.
         *
         * @return Returns the log events.
         */
7454    public List getLogEvents()
        {
            return logEvents;
        }

7459    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitEventGroup(com.motorola.taflogger.
               events.EventGroup)
         */
7464    public void visitEventGroup(EventGroup group)
        {
            for (Iterator iter = group.getLogEvents().iterator(); iter.hasNext();)
            {
                ((LogEvent) iter.next()).accept(this);
7469        }
        }

        /*
         * (non-Javadoc)
7474     *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitKeyPressEvent(com.motorola.
               taflogger.events.KeyPressEvent)
         */
        public void visitKeyPressEvent(KeyPressEvent evt)
        {
7479        if (this.wasPhoneScreenEvent)
            {
                this.wasPhoneScreenEvent = false;
            }
            else if (this.lastPhoneScreenEvent != null)
7484        {
                // Avoid to key press events to be add consecutively by adding last known phone
                // screen event before it.
                this.logEvents.add(lastPhoneScreenEvent);
            }
7489        this.logEvents.add(evt);
        }

        /*
         * (non-Javadoc)
7494     *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitPhoneScreenEvent(com.motorola.
               taflogger.events.PhoneScreenEvent)
         */
        public void visitPhoneScreenEvent(PhoneScreenEvent evt)
        {
7499        this.lastPhoneScreenEvent = evt;
            this.wasPhoneScreenEvent = true;
            this.logEvents.add(evt);
```

```
        }

7504    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitCommentedLogEvent(com.motorola.
             taflogger.events.CommentedLogEvent)
         */
7509    public void visitCommentedLogEvent(CommentedLogEvent evt)
        {
            // Empty.
        }

7514    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitIconMapper(com.motorola.taflogger.
             events.IconMapper)
         */
7519    public void visitIconMapper(IconMapper mapper)
        {
            // Empty.
        }

7524    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitLogSessionStateEvent(com.motorola.
             taflogger.events.LogSessionStateEvent)
         */
7529    public void visitLogSessionStateEvent(LogSessionStateEvent evt)
        {
            // Empty.
        }

7534    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitPhoneAddedEvent(com.motorola.
             taflogger.events.PhoneAddedEvent)
         */
7539    public void visitPhoneAddedEvent(PhoneAddedEvent evt)
        {
            // Empty.
        }

7544    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitPhoneConnectionChangedEvent(com.
             motorola.taflogger.events.PhoneConnectionChangedEvent)
         */
7549    public void visitPhoneConnectionChangedEvent(PhoneConnectionChangedEvent evt)
        {
            // Empty.
        }

7554    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitPhoneRemovedEvent(com.motorola.
             taflogger.events.PhoneRemovedEvent)
         */
7559    public void visitPhoneRemovedEvent(PhoneRemovedEvent evt)
        {
            // Empty.
        }

7564    /*
         * (non-Javadoc)
         *
         * @see com.motorola.taflogger.visitors.LogEventVisitor#visitScreenshotEvent(com.motorola.
             taflogger.events.ScreenshotEvent)
```

```
         */
7569     public void visitScreenshotEvent(ScreenshotEvent evt)
         {
             // Empty.
         }

7574     /*
          * (non-Javadoc)
          *
          * @see com.motorola.taflogger.visitors.LogEventVisitor#visitTextEvent(com.motorola.taflogger.
              events.TextEvent)
          */
7579     public void visitTextEvent(TextEvent evt)
         {
             // Empty.
         }

7584     /*
          * (non-Javadoc)
          *
          * @see com.motorola.taflogger.visitors.LogEventVisitor#visitUFEventGroup(com.motorola.
              taflogger.events.UFEventGroup)
          */
7589     public void visitUFEventGroup(UFEventGroup evt)
         {
             // Empty.
         }
     }
7594
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2007  Bruno Martins Petroski <petroski@inf.ufsc.br>
 *
7599 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
7604 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
7609 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
package cbr.runner;
7614
import java.util.logging.Logger;

/**
 * TODO CBRBarrier.java description.
7619 */
public class CBRBarrier
{
    /**
     * The {@link CBRBarrier} logger.
7624 */
    private static final Logger logger = Logger.getLogger(CBRBarrier.class.getName());

    /**
     * The number of hits to wait before releasing the threads.
7629 */
    private final int totalNumberOfThreads;

    /**
     * The current number of hits to wait before releasing the threads.
7634 */
    private int numberOfThreadsReady;
```

```
        /**
         * Constructor method.
7639     *
         * @param totalNumberOfThreads The number of hits to wait before releasing the threads.
         */
        public CBRBarrier(int totalNumberOfThreads)
        {
7644        this.totalNumberOfThreads = totalNumberOfThreads;
            this.numberOfThreadsReady = 0;
        }

        /**
7649     * Provides a synchronization to the threads.
         */
        public synchronized void waitForRealese()
        {
            // Increase the number of threads finished.
7654        this.numberOfThreadsReady++;

            // Verify if all threads finished.
            if (this.numberOfThreadsReady == this.totalNumberOfThreads)
            {
7659            // Reset the counter.
                this.numberOfThreadsReady = 0;

                // Notify all threads they're read to continue.
                this.notifyAll();
7664        }
            else
            {
                try
                {
7669                // Wait all threads to finish.
                    this.wait();
                }
                catch (InterruptedException e)
                {
7674                logger.fine("Error waiting all threads to finish: " + e.getMessage());
                }
            }
        }
    }
```

```
    /*
7679 * Trabalho de Conclusão de Curso
     * Copyright (C) 2007  Bruno Martins Petroski <petroski@inf.ufsc.br>
     *
     * This library is free software; you can redistribute it and/or
     * modify it under the terms of the GNU Lesser General Public
7684 * License as published by the Free Software Foundation; either
     * version 2.1 of the License, or (at your option) any later version.
     *
     * This library is distributed in the hope that it will be useful,
     * but WITHOUT ANY WARRANTY; without even the implied warranty of
7689 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
     * Lesser General Public License for more details.
     *
     * You should have received a copy of the GNU Lesser General Public
     * License along with this library; if not, write to the Free Software
7694 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
     */
    package cbr.runner;

    import java.util.Iterator;
7699 import java.util.List;
    import java.util.logging.Logger;

    import cbr.casebase.caze.CBRCase;
    import cbr.casebase.caze.UtilityFunction;
7704 import cbr.cycle.CBRCycle;
    import cbr.cycle.CBRCycleImp;

    import com.motorola.taflogger.events.*;
```

```
import com.motorola.taflogger.visitors.LogEventVisitor;

/**
 * TODO CBRCycleRunner.java description.
 */
public class CBRCycleRunner extends Thread
{
    /**
     * The {@link CBRCycleRunner} logger.
     */
    private static final Logger logger = Logger.getLogger(CBRCycleRunner.class.getName());

    /**
     * The barrier to wait after has finished.
     */
    private CBRBarrier barrier;

    /**
     * The beginning index of the log event list.
     */
    private int begin;

    /**
     * The end index of the log event list.
     */
    private int end;

    /**
     * The list of log events.
     */
    private List logEvents;

    /**
     * This threads run result;
     */
    private UtilityFunction solution;

    /**
     * Constructor.
     *
     * @param begin The beginning index of the log event list.
     * @param end The end index of the log event list.
     * @param logEvents The list of log events.
     */
    public CBRCycleRunner(int begin, int end, List logEvents)
    {
        super("CBRCycleRunner[" + begin + "-" + end + "]");
        this.begin = begin;
        this.end = end;
        this.logEvents = logEvents;
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Runnable#run()
     */
    public void run()
    {
        CBRCycle cycle = new CBRCycleImp();

        CBRCase problem = build(logEvents);

        cycle.solveProblem(problem);

        solution = cycle.getSolution();

        if (solution != null)
        {
            System.out.println("************************************************************");
            System.out.println("PROBLEM: " + problem.toString());
            System.out.println("\nSOLUTION: " + cycle.getSolutionCase());
            System.out.println("\n************************************************************");
```

```
            }

            barrier.waitForRealese();
7784    }

        /**
         * Returns the solution value.
         *
7789     * @return Returns the solution.
         */
        public UtilityFunction getSolution()
        {
            return solution;
7794    }

        /**
         * Sets the barrier value.
         *
7799     * @param barrier The barrier to set.
         */
        public void setBarrier(CBRBarrier barrier)
        {
            this.barrier = barrier;
7804    }

        /**
         * Returns the begin value.
         *
7809     * @return Returns the begin.
         */
        public int getBegin()
        {
            return begin;
7814    }

        /**
         * Returns the end value.
         *
7819     * @return Returns the end.
         */
        public int getEnd()
        {
            return end;
7824    }

        /**
         * Verify if other runner can run in parallel with this thread. I.e., they parse different parts
         * of the log event list.
7829     *
         * @param other The other runner.
         * @return <code>true</code> if the other thread can run in parallel with this thread.
         */
        public boolean isMutuallyExclusive(CBRCycleRunner other)
7834    {
            return this.end <= other.begin || this.begin >= other.end;
        }

        /**
7839     * Builds a CBRCase based on the list of log events.
         *
         * @param logEvents The list of log events.
         * @return The CBRCase built base on the list of log events.
         */
7844    private CBRCase build(List logEvents)
        {
            CBRCaseBuilderVisitor visitor = new CBRCaseBuilderVisitor(logEvents.subList(begin, end + 1));

            // Start the building.
7849        visitor.start();

            try
            {
                // Waits the build to finish.
```

```
7854                visitor.join();
                }
                catch (InterruptedException e)
                {
                    logger.fine("Error waiting the build of the CBRCase: " + e.getMessage());
7859            }

                return visitor.getCBRCaseBuilt();
            }

7864        /**
             * TODO CBRCycleRunner.java description.
             */
            private static class CBRCaseBuilderVisitor extends Thread implements LogEventVisitor
            {
7869            /**
                 * The case that will be built.
                 */
                private CBRCase caze;

7874            /**
                 * The list of events to visit.
                 */
                private List listOfEvents;

7879            /**
                 * Constructor.
                 */
                /**
                 * Constructor.
7884             *
                 * @param listOfEvents The list of events to visit.
                 */
                public CBRCaseBuilderVisitor(List listOfEvents)
                {
7889                this.caze = new CBRCase();
                    this.listOfEvents = listOfEvents;
                }

                /**
7894             * Returns the built CBRCase.
                 *
                 * @return the built CBRCase.
                 */
                public CBRCase getCBRCaseBuilt()
7899            {
                    return this.caze;
                }

                /*
7904             * (non-Javadoc)
                 *
                 * @see java.lang.Thread#run()
                 */
                public void run()
7909            {
                    for (Iterator it = this.listOfEvents.iterator(); it.hasNext(); /* empty */)
                    {
                        ((LogEvent) it.next()).accept(this);
                    }
7914            }

                /*
                 * (non-Javadoc)
                 *
7919             * @see com.motorola.taflogger.visitors.LogEventVisitor#visitKeyPressEvent(com.motorola.
                       taflogger.events.KeyPressEvent)
                 */
                public void visitKeyPressEvent(KeyPressEvent evt)
                {
                    caze.add(evt);
7924            }
```

```
        /*
         * (non-Javadoc)
         *
7929     * @see com.motorola.taflogger.visitors.LogEventVisitor#visitPhoneScreenEvent(com.motorola.
             taflogger.events.PhoneScreenEvent)
         */
        public void visitPhoneScreenEvent(PhoneScreenEvent evt)
        {
            caze.add(evt);
7934    }

        /*
         * (non-Javadoc)
         *
7939     * @see com.motorola.taflogger.visitors.LogEventVisitor#visitUFEventGroup(com.motorola.
             taflogger.events.UFEventGroup)
         */
        public void visitUFEventGroup(UFEventGroup evt)
        {
            caze.add(evt.getUtilityFunction());
7944    }

        /*
         * (non-Javadoc)
         *
7949     * @see com.motorola.taflogger.visitors.LogEventVisitor#visitCommentedLogEvent(com.motorola.
             taflogger.events.CommentedLogEvent)
         */
        public void visitCommentedLogEvent(CommentedLogEvent evt)
        {
            // Empty.
7954    }

        /*
         * (non-Javadoc)
         *
7959     * @see com.motorola.taflogger.visitors.LogEventVisitor#visitEventGroup(com.motorola.taflogger.
             events.EventGroup)
         */
        public void visitEventGroup(EventGroup group)
        {
            // Empty.
7964    }

        /*
         * (non-Javadoc)
         *
7969     * @see com.motorola.taflogger.visitors.LogEventVisitor#visitIconMapper(com.motorola.taflogger.
             events.IconMapper)
         */
        public void visitIconMapper(IconMapper mapper)
        {
            // Empty.
7974    }

        /*
         * (non-Javadoc)
         *
7979     * @see com.motorola.taflogger.visitors.LogEventVisitor#visitLogSessionStateEvent(com.motorola.
             taflogger.events.LogSessionStateEvent)
         */
        public void visitLogSessionStateEvent(LogSessionStateEvent evt)
        {
            // Empty.
7984    }

        /*
         * (non-Javadoc)
         *
7989     * @see com.motorola.taflogger.visitors.LogEventVisitor#visitPhoneAddedEvent(com.motorola.
             taflogger.events.PhoneAddedEvent)
         */
        public void visitPhoneAddedEvent(PhoneAddedEvent evt)
```

```
        {
            // Empty.
7994        }

        /*
         * (non-Javadoc)
         *
7999     * @see com.motorola.taflogger.visitors.LogEventVisitor#visitPhoneConnectionChangedEvent(com.
             motorola.taflogger.events.PhoneConnectionChangedEvent)
         */
        public void visitPhoneConnectionChangedEvent(PhoneConnectionChangedEvent evt)
        {
            // Empty.
8004        }

        /*
         * (non-Javadoc)
         *
8009     * @see com.motorola.taflogger.visitors.LogEventVisitor#visitPhoneRemovedEvent(com.motorola.
             taflogger.events.PhoneRemovedEvent)
         */
        public void visitPhoneRemovedEvent(PhoneRemovedEvent evt)
        {
            // Empty.
8014        }

        /*
         * (non-Javadoc)
         *
8019     * @see com.motorola.taflogger.visitors.LogEventVisitor#visitScreenshotEvent(com.motorola.
             taflogger.events.ScreenshotEvent)
         */
        public void visitScreenshotEvent(ScreenshotEvent evt)
        {
            // Empty.
8024        }

        /*
         * (non-Javadoc)
         *
8029     * @see com.motorola.taflogger.visitors.LogEventVisitor#visitTextEvent(com.motorola.taflogger.
             events.TextEvent)
         */
        public void visitTextEvent(TextEvent evt)
        {
            // Empty.
8034        }
    }
}
```

```
/*
 * Trabalho de Conclusão de Curso
 * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
8039 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
8044 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
8049 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */
8054 package cbr.similarity;

/**
 * Interface that's responsible to compute the similarity of two objects.
 */
```

```
8059 public interface CBRSimilarity
     {
         /**
          * Computes the similarity of two objects.
          * <p>
8064      * The <code>compute</code> method implements an equivalence relation on non-null object
          * references:
          * <ul>
          * <li>It is <i>reflexive</i>: for any non-null reference value <code>x</code>,
          * <code>compute(x,x)</code> should return <code>1d</code>.
8069      * <li>It is <i>symmetric</i>: for any non-null reference values <code>x</code> and
          * <code>y</code>, <code>compute(x,y)</code> should return the same value as
          * <code>compute(y,x)</code>.
          * <li>It is <i>consistent</i>: for any non-null reference values <code>x</code> and
          * <code>y</code>, multiple invocations of <tt>compute(x,y)</tt> consistently return the
8074      * same value, provided no information used in <code>compute</code> comparisons on the objects
          * is modified.
          * <li>For any non-null reference value <code>x</code>, <code>compute(x,null)</code>
          * should return <code>0d</code>.
          * </ul>
8079      * <p>
          *
          * @param a The object to be computed its similarity.
          * @return A <tt>double</tt> value between <code>0d</code> and <code>1d</code> determining
          * the similarity of the two objects. Thus, the value <code>0d</code> of similarity means the
8084      * objects are (or can be considered) totally different and <code>1d</code> means they are (or
          * can be considered) the same.
          */
         double similarityTo(Object a);
     }
```

```
     /*
8089  * Trabalho de Conclusão de Curso
      * Copyright (C) 2006  Bruno Martins Petroski <petroski@inf.ufsc.br>
      *
      * This library is free software; you can redistribute it and/or
      * modify it under the terms of the GNU Lesser General Public
8094  * License as published by the Free Software Foundation; either
      * version 2.1 of the License, or (at your option) any later version.
      *
      * This library is distributed in the hope that it will be useful,
      * but WITHOUT ANY WARRANTY; without even the implied warranty of
8099  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
      * Lesser General Public License for more details.
      *
      * You should have received a copy of the GNU Lesser General Public
      * License along with this library; if not, write to the Free Software
8104  * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
      */
     package cbr.similarity.centraltendencymeasures;

     /**
8109  * Utility class that responsible to compute the weighted mean of a data and weights array.
      */
     public class WeightedMean
     {
         /**
8114      * Computes the weighted mean of the data set.
          *
          * @param data The data.
          * @param weights The weights of all data.
          * @return The weighted mean of the data set.
8119      */
         public static double compute(double[] data, double[] weights)
         {
             if (data.length != weights.length)
             {
8124             return 0d;
             }

             double sumData = 0d;
             double sumWeight = 0d;
8129         for (int i = 0; i < data.length; i++)
```

```
            {
                sumData += weights[i] * data[i];
                sumWeight += weights[i];
            }

8134
            return sumData / sumWeight;
        }
    }
```

APÊNDICE B – Artigo

Geração automática de casos de teste automatizados no contexto de uma suite de testes em telefones celulares

Bruno Martins Petroski
<petroski@inf.ufsc.br>

2006/2

Resumo

Atualmente, as empresas estão recorrendo à automatização de testes visando diminuir o ciclo de desenvolvimento de *softwares*, o qual é um processo muito eficiente que elimina a necessidade de execução manual dos casos de teste automatizados. Contudo, o processo de gerar casos de teste automatizados não é trivial, e é responsável por grande parte do tempo despendido no processo de automatização. Em contrapartida ao processo de execução automatizada, temos o teste exploratório que. Esta técnica de teste de *software* consiste em testar o sistema de uma forma não-sistemática, e pode ser bastante eficaz em identificar erros não facilmente capturados por técnicas formais. Os resultados de uma sessão de teste exploratório não são necessariamente totalmente diferentes dos testes de roteiro, e essas duas abordagens de teste são compatíveis e comumente utilizadas no meio corporativo. Este trabalho apresenta um método que possibilita a geração automática de casos de teste automatizados, a partir dos *logs* de uma sessão exploratória e de *logs* provindos da execução automatizada de casos de testes. Tal método combina a eficácia do teste exploratório e a eficiência do teste automatizado, diminuindo assim, o ciclo de desenvolvimento de *software*.

1 Introdução

Atualmente a indústria de desenvolvimento de *software* enfrenta o seguinte paradoxo: os *softwares* estão cada vez mais complexos, dispõem de menos tempo para seu desenvolvimento e existe uma maior exigência em relação à qualidade. O teste de *software* freqüentemente corresponde por cerca de metade do custo total do desenvolvimento de *software* [6]. Consiste em uma técnica para avaliar a qualidade do produto e, indiretamente, melhorá-lo – através da identificação de defeitos e falhas.

Teste de roteiro[1] é o processo de teste cujos passos são definidos em um documento formal e podem ser executados de forma manual ou automatizada. Quando realizado manualmente, é comum que o processo de teste se torne suscetível a erros e altamente custoso, tanto financeiramente e quanto em relação ao consumo de tempo [7]. O processo automatizado de teste de *software* vem atacar este problema e, se aplicado corretamente, pode aumentar significativamente a quantidade de testes realizados durante um período de tempo ou reduzir o tempo de cada teste [11]. Automação dos casos de teste é um processo muito eficiente, pois reduz o tempo gasto no ciclo de desenvolvimento do *software* embutido nos telefones, eliminando a necessidade de execução manual destes. Contudo, o processo de gerar casos de teste automatizados não é trivial, e é responsável por grande parte do tempo despendido no processo de automatização do processo de teste de *software*.

Em contrapartida ao processo de teste utilizando roteiros, temos o teste exploratório. Teste exploratório consiste em testar o sistema de uma forma não-sistemática, e pode ser bastante eficaz em identificar casos especiais de teste, os quais não são facilmente capturados por técnicas formais.

Os resultados de uma sessão de teste exploratório não são necessariamente totalmente diferentes dos testes de roteiro, e essas duas abordagens de teste são compatíveis. Empresas como Nortel, Microsoft

[1] Do inglês *scripted test*.

e Motorola comumente utilizam estas duas abordagens no mesmo projeto [4]. Juntamente com a automatização do testes, a utilização destas duas abordagens em conjunto tem como objetivo combinar a eficácia do teste exploratório juntamente com a eficiência do teste automatizado. Este artigo propõe uma metodologia que possibilita a geração automática de casos de teste automatizados, a partir dos *logs* de uma sessão de teste exploratório e de *logs* provindos da execução automatizada de casos de testes.

Este artigo está organizado da seguinte maneira: as secções 2 e 3 apresentam a fundamentação teórica; a descrição da metodologia e os resultados obtidos estão na secção 4 e finalmente as conclusões são apresentadas na secção 6.

2 Raciocínio Baseado em Casos

O raciocínio baseado em casos [2] é uma abordagem para a resolução de problemas e aprendizado que tornou-se muito popular nos últimos anos. Mostrou-se bastante útil em uma grande variedade de aplicações, por exemplo: diagnósticos médicos [2, 19], aplicações *help–desk* on-line [9, 14], comércio eletrônico [20], tutor inteligente [17], pesquisa de defeitos em motores de aeronaves a jato [16] e controle militar [15, 10].

Segundo [13], raciocínio baseado em casos pode significar adaptar soluções antigas para ajustar-se a novas demandas; utilizar casos antigos para criticar novas soluções; ou raciocinar sobre precedentes para interpretar uma nova solução ou criar uma solução semelhante para o novo problema.

A qualidade das soluções advindas de um raciocinador baseado em casos depende de quatro fatores [13]:

- da sua experiência;
- da habilidade de entender novas situações baseadas nas experiências passadas;
- da adequação da adaptação; e
- da adequação da avaliação.

[2] Do inglês, *Case-Based Reasoning* (CBR).

2.1 O Ciclo de RBC

De modo geral, o RBC é descrito através do ciclo proposto por [1] e composto por quatro processos [3]:

1. **recuperar** o(s) caso(s) mais similar(es);
2. **reutilizar** a informação e o conhecimento neste caso(s) para resolver o problema;
3. **revisar** a solução proposta;
4. **reter** as partes desta experiência que potencialmente podem ser úteis na resolução de problemas futuros.

Quando um novo problema é proposto, sua descrição define um novo caso. Um ou mais casos passados similares a este novo são recuperados e reutilizados de alguma forma. A solução é revisada baseando-se na reutilização do caso passado, e por fim essa nova experiência é retida através de sua incorporação em uma base de conhecimento (base de casos) já existente.

2.1.1 Representação dos casos

Casos podem representar diferentes tipos de conhecimento e ser armazenado em uma base de casos em vários formatos de representação. Um sistema RBC é altamente dependente da estrutura e do conteúdo dos casos armazenados em sua base. O conteúdo dos casos é determinando pelo domínio de aplicação específico e dos objetivos do raciocinador. O problema de representação de um caso é basicamente o problema da decisão de o quê armazenar, achar uma estrutura apropriada para descrever o seu conteúdo, e decidir qual a organização e indexação do caso na memória para que a recuperação e a reutilização sejam efetivas. Suas representações variam muito e ass formas de representação de casos mais usuais são: atributo-valor, orientada a objetos, redes semânticas, árvores e grafos.

2.1.2 Indexação dos casos

Atribuir índices aos casos para futuras recuperações e comparações é o ponto central da indexação dos casos. Índice nada mais é do que um conjunto

[3] 4R como também é conhecido.

de atributos que melhor identifica um caso. A relevância do atributo no que diz respeito a sua dissimilaridade entre os casos deve ser considerada para atribuir este atributo como parte do índice. A indexação ordena e armazena através do agrupamento dos casos através dos índices. Métodos manuais e automáticos são utilizados no processo de atribuição dos índices relevantes. O processo manual baseia-se no conhecimento de especialistas sobre domínio de aplicação do raciocinador.

3 Teste de *Software*

Segundo [5], teste de *software* é:

> *a verificação dinâmica do comportamento de um programa em um conjunto finito de casos de teste, selecionado apropriadamente de um domínio de execuções comumente infinito, através do comportamento esperado.*

O comportamento observado pode ser verificado, por exemplo, segundo as expectativas do usuário (teste de validação) ou segundo as especificações do sistema (teste de verificação). Testar para identificar defeitos tem seu objetivo cumprido quando levam à uma falha do sistema. O que é bem diferente de testes que têm por objetivo demonstrar que o *software* está de acordo com suas especificações, onde o teste é um sucesso se falhas não são observadas durante sua execução. Uma famosa citação sobre o teste de *software* é a do cientista em computação holandês Edsger Dijkstra: "*teste de software pode ser apenas usado para mostrar a presença de bugs, mas nunca sua ausência*", a qual faz uma alusão ao fato de que testar completamente é inviável em sistemas reais.

3.1 Casos de teste e suites

Um caso de teste é um documento de teste que descreve a entrada, ações, eventos e resultados esperados para determinar se uma aplicação está funcionando corretamente ou não. É constituído basicamente por uma série de passos e seus resultados esperados. Casos de teste maiores também podem conter pré-requisitos à sua execução, como um estado ou passos a serem executados. Uma suite de testes é basicamente uma coleção de casos de teste. Usualmente também contém instruções ou objetivos mais detalhados para cada coleção de casos teste. Assim como um caso de teste individual, também pode conter pré-requisitos à sua execução.

3.2 Teste exploratório

Segundo [3] teste exploratório é: aprendizado, projeto e execução de testes simultaneamente. Teste exploratório tem como meta utilizar os recursos disponíveis, habilidades e conhecimento da melhor maneira possível, a fim de encontrar o maior número de erros críticos dentro do tempo disponível. É baseado na habilidade e conhecimento do testador sobre o processo e técnicas de teste. Consiste em testar o sistema de uma forma não-sistemática, podendo ser útil para identificar casos especiais de teste, os quais não são facilmente capturados por técnicas formais.

3.3 Teste automatizado

À medida que os *softwares* se tornam cada vez maiores e mais complexos, fica praticamente inviável de alocar recursos para realizar testes e verificações detalhadas. Automatizar uma ou mais partes do processo de teste de *software* está sendo apontado como uma solução para contornar tal problema. Dentre seus potenciais benefícios estão: substituir o trabalho de testar manualmente, aumentar a repetibilidade e a precisão dos testes, diminuindo os custos de desenvolvimento de *software*. Contudo, não existem somente benefícios acerca da automação do processo e, outras questões devem ser levadas em consideração quando da sua implementação, tais como: custos iniciais podem ser elevados, a abordagem exploratória do teste manual pode ser perdida, além de riscos inerentes ao desenvolvimento de qualquer *software*.

3.4 *Test Automation Framework*

O *Test Automation Framework* (TAF) é um *framework* projetado para suportar a automação de testes funcionais dos *softwares* embutidos em telefones celulares produzidos e desenvolvidos pela Motorola Industrial Ltda [12]. O *framework* foi projetado após observar-se que diversos modelos de telefone celular possuem as mesmas funcionalidades e que uma elevada quantidade de casos de teste em

comum são executadas nesses aparelhos [18]. Os mesmos casos de teste automatizados são reutilizados em diversos modelos que implementam as mesmas funcionalidades, assim a reutilização de código a partir da portabilidade dos testes é maximizada com a utilização do TAF. A automação dos casos de teste reduz o tempo gasto no ciclo de desenvolvimento do *software* embutido nos telefones, pois elimina a necessidade de execução manual destes.

Toda a interação entre o TAF e o telefone é realizada através da utilização de uma biblioteca de funções proprietária chamada de *Phone Test Framework* (PTF). O PTF comunica-se com o telefone via interface USB (*Universal Serial Bus*) e provê funções que, por exemplo, retornam o conteúdo que está sendo mostrado na tela e que simulam o pressionamento de teclas do telefone [8]. Contudo a utilização direta do PTF para automatizar um caso de teste produz *scripts* de teste ilegíveis e difíceis de serem portados para serem executados em outro modelo de telefone, pois as funções providas pelo PTF são de baixo nível de abstração.

3.4.1 Visão geral da arquitetura

A fim de aumentar o nível de abstração das funções providas pelo PTF, o TAF foi projetado. No TAF, chamadas diretas à biblioteca de funções do PTF são encapsuladas em *Utility Functions* (UFs). Uma UF é a implementação de um passo de alto nível que é executado em um caso de teste. Sendo assim os casos de teste automatizados são escritos em termos de UFs de alto nível de abstração, promovendo a reutilização de código. Alguns exemplos de UFs que representam um passo de alto nível em um caso de teste são: *ComposeMessage* (compõe uma mensagem de texto ou de multimídia), *LaunchApp* (abre uma aplicação qualquer, como o editor de mensagens ou a lista de contatos) e *DeleteAllPictures* (remove todas as figuras do telefone).

A interface *Step* é implementada por todas as UFs no TAF. Segundo [18]: "*Sob certo ponto de vista, um caso de teste é uma lista de Steps; tudo o que um caso de teste faz é invocar o método execute que é definido na interface Step e implementado nas UFs concretas. Cada UF, entretanto, além de implementar a interface Step, possui uma API (Application Programming Interface) bem definida que provê métodos que possibilitam a interação com esta UF*".

3.4.2 Controle do estado do telefone

Cada execução de uma UF potencialmente produz uma mudança no estado do telefone. Antes da UF ser executada o telefone encontra-se no estado inicial, a UF é executada e, após sua execução, o telefone está no estado final, comportamento análogo à um autômato finito. Contudo, nem todas as UFs mudam o estado do telefone. Em geral, são UFs responsáveis por realizar verificações do estado do telefone, do tipo: garantir que o telefone encontra-se em alguma aplicação ou que existem um número de mensagens não lidas na caixa de entrada. Cada UF possui uma pré-condição para que sua execução seja possível, a qual geralmente está vinculada ao estado em que o telefone se encontra. E, após o término da execução da UF, o telefone estará no estado especificado pela pós-condição desta UF.

Mudanças no estado do telefone são geralmente vinculadas a estímulos no teclado do telefone. Assim, cada vez que uma tecla do telefone é estimulada, em mais baixo nível pelo PTF, o telefone transiciona para um novo estado.

3.5 TAFLogger

O TAFLogger é um *software* projetado para auxiliar a execução de testes exploratórios dos *softwares* embutidos em telefones celulares produzidos e desenvolvidos pela Motorola. Nas sessões de teste exploratório realizadas com o auxílio do TAFLogger, todas as ações realizadas sobre o telefone pelo testador são registradas. Dentre os principais registros de uma sessão de testes do TAFLogger estão: horários de início e fim da sessão, notas de texto realizados pelo testador (incluindo comentários genéricos, possíveis problemas no telefone e *bugs*), teclas pressionadas e estados do telefone.

Assim como o TAF, toda a interação entre TAFLogger e o telefone é realizada através da utilização do PTF via interface USB [8]. Mas, ao contrário do que era feito no TAF, a função do PTF no TAFLogger é somente reportar eventos que acontecem no telefone, *e.g.* o pressionamento de teclas, e não estimular o telefone de alguma maneira. Isso deve-se ao fato de que todas as ações realizadas

sobre o telefone são oriundas do testador durante a sessão.

3.6 Correlação entre teste de *software*, TAF e TAFLogger

Ambos, TAF e TAFLogger, tem como objetivo auxiliar o processo de teste dos *softwares* embutidos em telefones celulares produzidos e desenvolvidos pela Motorola. O TAF é um *framework* que automatiza os testes funcionais e de regressão nos telefones celulares. Já o TAFLogger é responsável por auxiliar o processo manual de testes exploratórios.

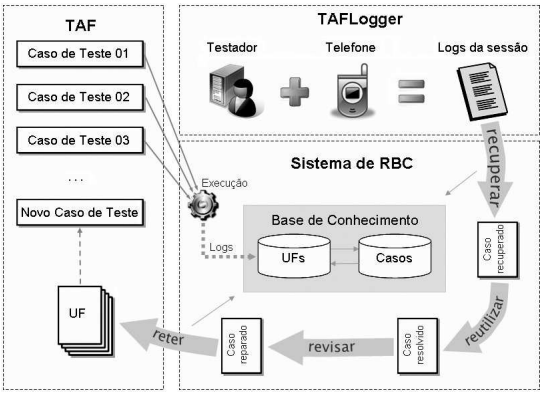

Figura 1: Visão geral do processo de geração de casos de teste automatizados a partir de testes exploratórios

4 Geração de Casos de Teste Automatizados a Partir de Testes Exploratórios

4.1 Visão geral

O processo de geração de casos de teste automatizados a partir de testes exploratórios é ilustrado na Figura 1. Cada execução de um caso de teste automatizado gera um log que é armazenado. Posteriormente todos os logs gerados das execuções automatizadas são analisados à procura de chamadas de UF. Cada chamada de UF é então convertida à um caso do raciocinador baseado em casos, e serve para alimentar sua base de conhecimento. Uma sessão de teste exploratório qualquer utilizando o TAFLogger serve como entrada para o sistema implementado. Tal sessão é dividida em potenciais problemas (UFs) que são a entrada do sistema RBC. Cada problema é então submetido ao ciclo do raciocinador que tem por objetivo encontrar uma solução (UF correspondente à esta parte da sessão). Após todas as partes da sessão serem analisadas à procura por UFs correspondentes, um novo caso de teste compatível com o TAF é criado. A lista de passos (*Steps*) do novo caso de teste é montada a partir das UFs encontradas pelo raciocinador. Este novo caso de teste pode ser incluído em alguma suite do TAF e ser executado de forma automatizada.

4.1.1 Adaptação dos *frameworks* e aplicações disponíveis

Para que o TAF coletasse os logs de forma aproveitável, foram necessárias algumas modificações. A principal delas foi utilizar o próprio TAFLogger para coletar os eventos que o *framework* realiza sobre os telefones que estão sendo testados. A utilização do TAFLogger foi implementada utilizando o padrão de projeto *Façade*. Este padrão de projeto visa promover um baixo acoplamento entre o TAF e o TAFLogger. Isso acontece pois a comunicação e as dependências entre os dois sistemas é diminuída, tornando-os mais fácil de utilizar.

A adaptação necessária ao TAFLogger foi mínima: apenas a adesão de parte do sistema à outro padrão de projeto, o MVC (Modelo-Visão-Controlador [4]). Para tal adequação, a separação da interface gráfica com o usuário e a parte lógica foi implementada em determinadas partes do código fonte.

4.2 Aplicação da metodologia de RBC

4.2.1 Representação do caso do RBC

Conforme descrito na Seção 2.1.1, existem diversas formas de representar casos. O formato adotado foi o de uma dupla problema e solução. O pro-

[4] Em inglês, *Model-View-Controller*.

blema é a seqüência de estados, teclas estimuladas e UFs chamadas durante a execução de uma UF, modelada como um autômato finito. A solução é justamente a UF que originou tal seqüência, juntamente com os valores dos parâmetros definidos em sua API. Formalmente um autômato finito é representado por uma 5-tupla (Q, Σ, δ, $s0$, F), onde: Q é o conjunto finito de estados; Σ é o conjunto finito de símbolos (alfabeto); δ é a função de transição; $s0$ é o estado inicial; e F é o conjunto de estados aceitos (ou finais). Portanto, o autômato finito modelado pelo problema é o seguinte:

Q – conjunto finito de estados (telas) capturadas e de UFs invocadas;

Σ – conjunto de teclas estimuladas durante a execução de uma UF;

δ – tabela que representa todas as transições realizadas pela UF;

$s0$ – é o estado em que o telefone se encontrava quando a UF foi chamada;

F – é o conjunto de estados composto apenas pelo estado em que o telefone se encontrava quando terminou a execução dessa da UF.

4.2.2 Base de casos e indexação

Definida a representação de um caso, um recipiente para armazená-los foi modelado – a base de casos. A base de casos nada mais é do que um repositório indexado de casos. Os índices de um caso são combinações de seus atributos mais importantes, que permitem distinguí-lo de outros. No caso modelado, os atributos mais importantes são: conjunto de estados capturados, UFs invocadas, e o conjunto de teclas estimuladas.

A estrutura de indexação adotada foi uma árvore *k-d*. Uma árvore *k-d* é uma árvore de pesquisa binária *k*-dimensional, que decompõe a base de casos iterativamente em partes menores [21]. Em cada nível desse tipo de árvore utiliza-se uma chave diferente dentre suas k chaves (equivalente ao número de atributos indexáveis). Cada nodo da árvore *k-d* representa um subconjunto dos casos na base de casos, e a raiz representa toda a base de casos. As folhas da árvore contém um subconjunto específico da base de casos.

A Figura 2 mostra a estrutura de indexação com alguns índices e casos a título de exemplificação. No nível 0 está a base de casos. Logo abaixo, no nível 1, estão os grupos de UFs (`uf_grp_1`, `uf_grp_2`, `none_uf_grp`, etc.). Os casos que não invocam nenhuma UF são agrupados no grupo `none_uf_grp` [5]. No nível 2 da árvore, estão os índices relativos aos grupos de estados capturados (`scrn_grp_1`, `scrn_grp_2`, `scrn_grp_n`, etc.). No nível imediatamente abaixo (3) os casos ficam agrupados segundo às teclas estimuladas. Os nodos ilustrados na Figura 2 que estão nesse nível são: (`key_grp_1`, `key_grp_2`, `key_grp_n`, etc.). Finalmente, os casos, que são as folhas da árvore, ficam armazenados no nível 4. Dentre as folhas estão os casos `case_1`, `case_2`, `case_n`, etc.

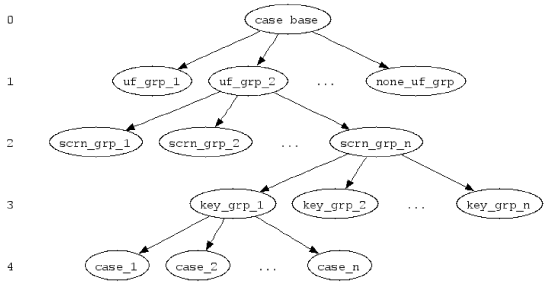

Figura 2: Estrutura de índices da base de casos

4.2.3 Implementação do ciclo de RBC

Para implementar o ciclo 4R da metodologia de RBC, foi definido um esqueleto do algoritmo na classe `CBRCycle` utilizando o padrão de projeto *template method*. Cada um dos 4 processos do ciclo é delegado à outras classes, que são responsáveis pelos passos específicos de seu processo (`CBRCycleRetrieve`, `CBRCycleReuse`, `CBRCycleRevise`, `CBRCycleRetain`). O diagrama de classes do ciclo 4R do sistema de RBC é apresentado na Figura 3. Observa-se no diagrama de classes que um objeto da classe `CBRCyleContext` é passado para todos os processos. Esse objeto é responsável por armazenar o contexto durante os quatro processos do ciclo.

[5] Em geral, os casos que não invocam nenhuma UF correspondem às UFs de mais baixo nível.

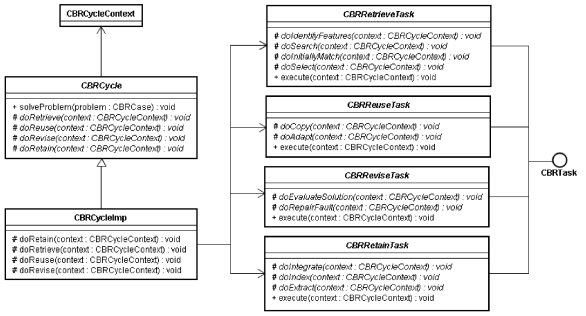

Figura 3: Diagrama de classes geral do ciclo RBC

5 Resultados obtidos

Com o intuito de analisar e validar a aplicação implementada, a seguinte metodologia foi seguida:

1. Foi escolhido um caso de teste automatizados simplificados e responsável por testar uma determinada funcionalidade, *e.g.*, a agenda de telefones ou mensagens;

2. O caso de teste automatizado foi executado e seus logs foram armazenados;

3. Uma sessão de teste exploratório serão realizadas seguindo a seguinte regra: seguirá a risca os passos de um dos casos teste automatizado;

4. A sessão de teste exploratório terá seu log coletado;

5. Os logs coletados tanto da execução automatizada quanto na sessão de teste exploratório servirá de entrada para aplicação implementada e uma saída será produzidas;

6. A saída da aplicação será confrontada com seu resultado esperado.

O primeiro caso de teste escolhido foi o `TC_DELETE_EMAIL` e seus passos de mais alto nível são:

1. `navigationTk.launchApp(PhoneApplication.MESSAGES);` – ir para a aplicação de e-mail;

2. `emailTk.deleteEmailMsg(EmailContent.SUBJECT_HELLO);` – remover o e-mail que contenha como assunto da mensagem: "Hello";

3. `navigationTk.goToIdle();` – voltar para a tela principal.

Com o logs do caso de teste automatizado, foi a vez de realizar a sessão de teste exploratória e armazenar os logs. Então a aplicação implementada foi executada, tendo como entrada os logs do caso de teste automatizado e os logs da sessão de teste exploratório. O resultado pode ser visualizado na Tabela 1.

Comparando os resultados obtidos pela sessão de teste exploratório e o caso de teste automatizado `TC_DELETE_EMAIL` pode-se ver que existem consideráveis semelhanças. A análise feita por parte da ferramenta implementada foi realizada com sucesso até a navegação que abre a aplicação de mensagens. Os dois passos seguintes realizam entrada efetiva na caixa de entrada dos e-mails, que fica dentro da aplicação de mensagens. Seguindo o caso de teste, deveria ser encontrado uma chamada para a UF que remove um e-mail da pasta corrente. Tal UF utiliza-se de mais outras 3 UFs em ordem: *ScrollToMessage*, *GoToAndSelectMenuItem* e *VerifyDialog*. As duas primeiras foram encontradas pela ferramenta. Contudo, com o nível de precisão adotado pela aplicação, não foi possível inferir que somente 2 das 3 UFs responsáveis para remover um e-mail do telefone sejam suficientes para tal análise. No final do caso de teste, o retorno para tela principal do telefone foi encontrado.

Tabela 1: Resultado da sessão de teste exploratório

#	UFs encontradas
1	`phoneTk.pressKey(PhoneHardKey.END)`
2	`navigationTk.launchApp(PhoneApplication.MESSAGES)`
3	`phoneTk.pressKey(PhoneHardKey.DOWN)`
4	`phoneTk.pressKey(PhoneHardKey.CENTER)`
5	`msgTk.scrollToMessage(EmailContent.SUBJECT_HELLO)`
6	`navigationTk.goToAndSelectMenuItem(SmsEmsMmsItem.DELETE)`
7	`navigationTk.launchApp(PhoneApplication.IDLE)`

6 Conclusões

Teste de *software* é uma atividade crucial no atual processo de desenvolvimento de *softwares* de

alta qualidade. Também contribui com uma grande parcela do custo total do desenvolvimento de *software* – freqüentemente cerca de 50% [6].

Casos de teste podem ser tanto executados manualmente ou de forma automatizada. Quando realizado de forma manual pode tornar-se suscetível a erros e altamente custoso, então o processo automatizado surge como uma das melhores alternativas. Contudo criar casos de teste passíveis de automação não é uma tarefa trivial e é responsável por grande parte do tempo despendido no processo de automatização de teste de *software*. Outra técnica de teste de software comumente difundida é o teste exploratório, e consiste em testar o sistema de uma forma não-sistemática, podendo ser útil para identificar casos especiais de teste, os quais não são facilmente capturados por técnicas formais.

Durante um ciclo de testes automatizados, apesar de eficiente, erros podem se infiltrar sem serem descobertos. Sessões de teste exploratório mostram-se bastante eficazes em identificar tais erros. A aplicação implementada durante a elaboração do presente trabalho obteve êxito combinando a eficácia do teste exploratório juntamente com a eficiência do teste automatizado.

O processo de geração de casos de teste automatizados tornou-se muito mais rápido e fácil pois os casos são gerados automaticamente a partir de uma sessão de teste exploratório. Isso porque executar uma sessão de teste exploratório utilizando um telefone celular e gerar o caso de teste automaticamente é mais fácil do que elaborá-lo programaticamente. Assim, o teste gerado automaticamente é incorporado aos próximos ciclos de execução automatizada. Então os erros descobertos durante a sessão exploratória podem ser reproduzidos automatizadamente, não somente para o modelo de telefone onde a sessão foi executada mas também para outros modelos devido a portabilidade dos casos teste do TAF.

O ciclo de desenvolvimento de *software* pôde ser diminuído aplicando a metodologia de raciocínio baseado em casos para gerar automaticamente casos de teste automatizados. Uma aplicação foi implementada para validar a metodologia e tem como entrada apenas os *logs* de uma ferramenta de auxílio a testes exploratórios e dos *logs* de um *framework* de automação de testes dos *softwares* embutidos em telefones celulares.

Referências

[1] Agnar Aamodt and Enric Plaza. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.

[2] Klaus-Dieter Althoff, Ralph Bergmann, Stefan Wess, Michel Manago, Eric Auriol, Oleg. I. Larichev, Alexander Bolotov, Yurii I. Zhuravlev, and Serge I. Gurov. Case-based reasoning for medical decision support tasks: the inreca approach. *Artificial Intelligence in Medicine*, 12(1):25–41, jan 1998.

[3] James Bach. Exploratory testing explained, 2003.

[4] James Bach. What is exploratory testing?, 2003.

[5] Pierre Bourque, Robert Dupuis, Alain Abran, James W. Moore, and Leonard Tripp. *Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society Press, Los Alamitos, CA, USA, 2001.

[6] S. K. Chernonozhkin. Automated test generation and static analysis. *Programming and Computer Software*, 27(2):86–94, mar 2001.

[7] N.S. Eickelmann and D.J. Richardson. An evaluation of software test environment architectures. In *Proceedings of the 18th International Conference*, pages 353–364, Berlim, mar 1996. Software Engineering, 1996.

[8] I. Esipchuk and D. Validov. Ptf-based test automation for java applications on mobile phones. In *IEEE 10th International Symposium on Consumer Electronics*, pages 1 – 3, 2006.

[9] Mehmet H. Göker and Thomas Roth-Berghofe. The development and utilization of the case-based help-desk support system homer. *Engineering Application of Artificial Intelligence*, 12(6):665–680, dec 1999.

[10] M. Goodman. Cbr in battle planning. In *Proceedings of the Second Workshop on Case-Based Reasoning*, Pensacola Beach, US, 1989.

[11] I D Hicks, G J South, and A O Oshisanwo. Automated testing as an aid to systems integration. *BT Technology Journal*, 15(3):26–36, jul 1997.

[12] Luiz Kawakami, André Knabben, Douglas Rechia, Denise Bastos, Otavio Pereira, and Ricardo Pereira Silvae e Luiz Santos. A test automation framework for mobile phones. In *VIII IEEE Latin-American Test WorkShop*, mar 2007 (to appear).

[13] Janet L. Kolodner. An introduction to case-based reasoning. *Artificial Intelligence Review*, 6(1):3–34, mar 1992.

[14] Mark Kriegsman and Ralph Barletta. Building a case-based help desk application. In *IEEE Expert*, volume 8, pages 18–26, dec 1993.

[15] Shu Liao. Case-based decision support system: Architecture for simulating military command and control. *European Journal of Operational Research*, 123(3):558–567, jun 2000.

[16] R. V. Magaldi. Cbr for troubleshooting on the flightline. In *IEE Colloquium on Case-Based Reasoning: Prospects for Applications*, pages 6/1–6/9, London, UK, mar 1994.

[17] Simon Masterton. The virtual participant: Lessons to be learned from a case-based tutor's assistant. In *Proceedings of Computer Supported Collaborative Learning*, Toronto, Canada, 1997.

[18] Douglas Nascimento Rechia. Especificação formal de restrições de projeto para frameworks orientados a objetos. Master's thesis, Universidade Federal de Santa Catarina, dezembro 2005.

[19] Rainer SCHMIDT, Stefania MONTANI, Riccardo BELLAZZI, Luigi PORTINALE, Lothar GIERL, Bernd BLOBEL, Joachim DUDECK, Rolf ENGELBRECHT, Gunther GELL, and Hans-Ulrich PROKOSCH. Cased-based reasoning for medical knowledge-based systems. *International journal of medical informatics*, 64(2–3):355–367, 2001.

[20] Ivo Vollrath, Wolfgang Wilke, and Ralph Bergmann. Case-based reasoning support for online catalog sales. *IEEE Internet Computing online*, 2(4):47–54, 1998.

[21] Christiane Gresse von Wangenheim and Aldo von Wangenheim. *Raciocínio Baseado em Casos*. Manole, Barueri, São Paulo, Brasil, 2003.