JOICE BARBOSA MENDES

UM FRAMEWORK DE RACIOCÍNIO BASEADO EM CASOS APLICADO PARA ESTRUTURAR A BASE DE CONHECIMENTO EM SISTEMAS TUTORES INTELIGENTES

Itajubá (MG), Fevereiro de 2012

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Joice Barbosa Mendes

UM FRAMEWORK DE RACIOCÍNIO BASEADO EM CASOS APLICADO PARA ESTRUTURAR A BASE DE CONHECIMENTO EM SISTEMAS TUTORES INTELIGENTES

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do Título de Mestre em Ciências em Ciência e Tecnologia da Computação

Área de Concentração: Inteligência Artificial

Orientador: Dr. Alexandre Carlos Brandão Ramos

Co-orientadora: Dra. Isabela Neves Drummond

Fevereiro de 2012

Itajubá - MG

Ficha catalográfica elaborada pela Biblioteca Mauá ï
Bibliotecária Cristiane Carpinteiro- CRB_6/1702

M538u

Mendes, Joice Barbosa

Um framework de raciocínio baseado em casos aplicados para estruturar a base de conhecimento em sistemas tutores inteligentes. / por Joice Barbosa Mendes. -- Itajubá (MG) : [s.n.], 2012.

147 p. : il.

Orientador : Prof. Dr. Alexandre Carlos Brandão Ramos.
Coorientadora : Profa. Dra. Isabela Neves Drummond.
Dissertação (Mestrado) ï Universidade Federal de Itajubá.

1. Inteligência artificial. 2. Raciocínio baseado em casos. 3. Framework para desenvolvimento. I. Ramos, Alexandre Carlos Brandão, orient. II. Drummond, Isabela Neves, coorient. III. Universidade Federal de Itajubá. IV. Título.

Resumo

Atualmente, existe a necessidade de se oferecer ensino personalizado, segundo o perfil dos usuários. Nesse contexto, surgiram os sistemas tutores inteligentes, que usam de técnicas de inteligência artificial para prover ensino dinâmico, considerando as habilidades e deficiências dos alunos. O raciocínio baseado em casos (RBC) é uma técnica de IA que tenta simular o funcionamento do cérebro humano, buscando solucionar um novo problema através da recuperação e adaptação de casos passados armazenados na base de conhecimento. Essa técnica utiliza diferentes cálculos de medida de similaridade como meio de mensurar o quão semelhante um caso é de outro, considerando seus atributos e pesos associados a eles. Uma das maiores dificuldades da aplicação do RBC é a modelagem dos dados na forma de casos, que é totalmente dependente do domínio da aplicação. O framework proposto visa auxiliar desenvolvedores de sistemas RBC na modelagem de dados e na recuperação de casos semelhantes através de medidas de similaridade; o framework tem como objetivo estruturar a base de conhecimento do sistema em construção. A ferramenta foi testada através de um estudo de caso, utilizando um sistema tutor para novos pilotos de helicóptero. Foram modelados casos de dois tipos distintos: casos teóricos e práticos. Para ambos os casos, foi possível inserir novos episódios e mensurar o quão semelhante os outros casos existentes são do caso em estudo.

Palavras Chaves: inteligência artificial, raciocínio baseado em casos, framework para desenvolvimento.

Abstract

Currently there is a need to offer personalized instruction, according to the users profile. In this context emerged the intelligent tutoring systems, which use artificial intelligence techniques to provide dynamic teaching, considering the students' abilities and disabilities. The case-based reasoning (CBR) is an AI technique that try simulate the human brain function, seeking to solve a new problem through the restoration and adaptation of past cases stored in the knowledge base. This technique uses different calculations of similarity measure as a means of measuring how similar a case is another, considering its attributes and weights associated with them. A major difficulty of the implementation of CBR is the data model, which is totally dependent on the application domain. The proposed framework aims to assist developers of CBR systems in data modeling and retrieval of similar cases by the similarity measures, the framework aims to structure the knowledge base system under construction. The tool was tested through a case study using a mentor system for new helicopter pilots. We modeled two distinct types of cases, theoretical and practical cases. In both cases it was possible to insert new episodes and measure how similar are other cases existing in the case study.

Key words: artificial intelligence, case-based reasoning, framework for development.

Sumário

Lista de Tabelas

Lista de Figuras

1 Introdução

A inteligência artificial é definida por (FERNANDES, 2003) como uma área da Ciência da Computação que pesquisa a forma de representação do conhecimento e a capacidade das máquinas assimilarem e manipularem dados tentando solucionar problemas.

Ainda segundo (FERNANDES, 2003), a inteligência humana é fruto de suas habilidades cogniticas e conotativas, ou seja, o ser humano é capaz de se expressar e reconhecer diferentes ações. Pesquisas na área de IA tentam implementar as habilidades humanas em máquinas inteligentes, buscando adaptá-las de forma que adquiram capacidade de interagir com o meio externo, desenvolvendo formas de obter conhecimento, aprendizado e realizar reconhecimento de padrões.

Existem diferentes técnicas que implementam essas funcionalidades, como redes bayesianas, redes neurais, lógica fuzzy e também o raciocínio baseado em casos (RBC). O raciocínio baseado em casos é definido por (WANGENHEIM; WANGENHEIM, 2003) como um enfoque para a solução de problemas e para o aprendizado baseado em experiência passada. Sistemas RBC resolvem problemas ao recuperar e adaptar experiências passadas - chamadas casos - armazenadas em uma base de casos. Um novo problema é resolvido com base na adaptação de soluções de problemas similares já conhecidas.

O RBC foi inspirado nos trabalhos de Schank e Abelson de Memória Dinâmica, em 1977. A proposta do trabalho era armazenar o conhecimento geral das pessoas na forma de roteiros. O primeiro sistema desenvolvido com essa filosofia foi denominado de CYRUS, criado por Janet Kolodner. Esse sistema contribuiu para a criação posterior de sistemas baseados em RBC e utilizou a abordagem de Schank na organização do conhecimento (AAMODT; PLAZA, 1994).

Segundo (KOLODNER, 1992) um sistema de raciocínio baseado em casos resolve novos problemas através da reutilização de conhecimento e experiências que são recuperadas de uma situação já ocorrida que mais se assemelhe ao novo caso. Um ou mais casos com características semelhantes podem ser recuperados da base de conhecimento. Esses casos

são adaptados de forma a encontrar uma solução satisfatória para o problema em estudo e, posteriormente, esse mesmo caso é armazenado na memória para servir de consulta.

A técnica de RBC é composta por quatro elementos, a representação do conhecimento, medidas de similaridade, adaptação e aprendizado. A representação do conhecimento descreve casos já experimentados. A medida de similaridade define os cálculos que serão realizados para medir a semelhança entre dois casos diferentes. A adaptação define mecanismos para adaptar casos recuperados segundo o caso em questão. O aprendizado garante que o sistema esteja em constante evolução, atualizando sempre sua base de conhecimento. (WANGENHEIM; WANGENHEIM, 2003).

Uma das maiores dificuldades na modelagem de um sistema RBC é a representação do conhecimento na forma de caso. Um caso deve ser modelado de forma que descreva o episódio ocorrido e suas informações sejam suficientes para definir a similaridade entre dois episódios distintos.

1.1 Problema de Pesquisa

Os sistemas RBC desenvolvidos são dependentes do domínio no qual serão aplicados. Essa depêndencia dificulta a elaboração de uma ferramenta genérica para a criação de novos sistemas que utilizam essa técnica. Além disso, desenvolvedores de aplicação RBC devem ter pleno domínio da técnica para que o sistema funcione eficazmente.

A modelagem dos casos consiste em um dos maiores problemas na elaboração de um novo sistema, bem como a definição das medidas de similaridade apropriadas para a recuperação dos casos na base de conhecimento e posterior utilização para encontrar a solução do novo caso estudado.

Os atributos de um caso podem ser descritos de diferentes formas, visto que seus valores variam desde dados numéricos, definidos dentro de uma faixa aceitável, até valores simbólicos que caracterizam os valores válidos dentro de um conjunto de **strings**. Ou seja, a introdução de um novo caso no sistema deve ser realizada de forma que cada atributo caracterizador assuma um valor válido já definido na modelagem.

A modelagem e a inserção de um novo caso no sistema só podem ser feitas após a definição dos atributos que aquele dado caso conterá. Portanto, a especificação dos casos, com seus diferentes atributos, pode se tornar um gargalo na construção de sistemas RBC, pois é a partir dos casos que todo o ciclo RBC é realizado e o sistema passa a funcionar

e a aprender.

A disponibilização de uma ferramenta de desenvolvimento para a modelagem de dados pode simplificar a tarefa de construção de novos sistemas RBC, visto que através do mesmo os atributos dos casos podem ser especificados, tanto sobre o tipo quanto sobre os valores válidos ligados a ele.

Levando em consideração as deficiências encontradas na modelagem de atributos de casos para sistemas RBC, um framework genérico para estruturação da base de conhecimento é proposto, visando facilitar o processo de especificação dos atributos e, posteriormene, de caracterização de casos.

1.2 Solução Proposta

Levando em consideração as características de sistemas RBC e as dificuldades que um desenvolvedor encontra em modelar os dados na forma de casos, é proposto um novo framework destinado a organizar o conhecimento do sistema.

A ferramenta desenvolvida visa facilitar a modelagem de novos casos, bem como a introdução de novos episódios no sistema. Além disso, o framework disponibiliza medidas de similaridade para a análise da similaridade entre os casos já existentes na aplicação, apresentando valores numéricos que medem o quão similar um caso é de outro.

O framework proposto abrange as seguintes funcionalidades:

- Interface amigável para a modelagem de novos casos: O framework desenvolvido disponibiliza ao desenvolvedor uma alternativa simplicada para a definição de modelos de casos, possibilitando a criação de modelos que descrevem um roteiro de episódio. Os modelos são definidos por atributos específicos, indicando os valores aceitáveis por ele, seja atributos numéricos ou descritivos;

- Interface amigável para a inserção de novos casos: O framework utiliza os dados definidos nos modelos para oferecer diferentes combinações de valores de atributos na inserção de novos casos. Os casos adicionados ao sistema terão sempre valores aceitáveis em cada atributo identificador, sendo que o mesmo pode assumir valor nulo que não será considerado no cálculo da similaridade;

- Medidas de similaridade: Após a modelagem e a inserção de novos casos no sistema, o framework é capaz de calcular a similaridade que um dado caso possui em relação

a todos os outros casos que seguem o mesmo modelo dentro do sistema;

- Exportação de dados: Como um sistema RBC pode ser desenvolvido em diferentes linguagens de programação, o framework possibilita a exportação dos dados modelagos através de um arquivo do tipo ".txt". Esse arquivo pode ser utilizado como entrada na aplicação em desenvolvimento.

- Visualização de Modelos e Casos: A ferramenta desenvolvida possibilita que o desenvolvedor observe os casos que foram modelados no sistema e também os casos já inseridos nele. Assim como a interfaces de modelagem e inserção de casos, a visualização é feita de uma forma simples e facilitada.

1.3 Estrutura da Dissertação

A dissertação está organizada em seis capítulos correlatos. O primeiro capítulo apresenta uma introdução a sistemas tutores inteligentes e a técnica de raciocínio baseado em caso. É apresentado também o problema que justifica a elaboração de um framework para a modelagem de casos e organização da base de conhecimento.

Os Capítulos 2 e 3 descrevem toda a fundamentação teórica. O Capítulo 2 apresenta a história e as características de sistemas tutores inteligentes. O Capítulo 3 apresenta a história e a lógica da técnica de raciocínio baseado em casos. Ainda no Capítulo 3, são mostradas todas as etapas de um ciclo RBC, além das diferentes medidas de similaridade encontradas na literatura.

O Capítulo 4 descreve a modelagem do framework proposto e forma de funcionamento do mesmo. Nesse capítulo, são apresentadas todas as interações que o usuário pode realizar com o sistema, desde a modelagem dos casos até a exportação desses dados em um arquivo.

No Capítulo 5, é descrito um estudo de caso utilizando dados de um sistema tutor já implementado para a validação do framework proposto. São descritos dois tipos de modelagens de dados, casos associados ao ensino teórico e prático. O Capítulo 6 traz a conclusão do trabalho realizado, levantando as funcionalidades e facilidades encontradas na utilização da ferramenta e outras funções que podem ser adicionadas a ele como trabalhos futuros.

2 Sistemas Tutores Inteligentes

Grandes empresas reconhecem que o aprendizado é fundamental para o sucesso e investem bilhões em treinamentos de funcionários, seja para uma melhor produção ou para melhor atender seus clientes. Em muitas profissões, o treinamento baseado em prática extensiva é necessário e esse treinamento geralmente é caro. Em alguns cenários é preciso oferecer treinamento personalizado ao aluno, dificultando a ação dos tutores em suprir todas as particularidades dos alunos. Essas dificuldades podem aparecer em diversos ambientes, incluindo ambientes de ensino.

Na década de 50, foram apresentados os primeiros sistemas tutores voltados para a educação chamados CAI [1] - Instruções Assistidas por Computador. Essas ferramentas eram baseadas no modelo comportamentalista, tendo o professor como figura central e o aluno como figura passiva. No entanto, essa abordagem apresentava o problema de não se adaptar ao modelo de aprendizado do aluno, visto que não era dinâmica e seguia sempre uma ordem pré-estabelecidas de passos a serem executados pelo mesmo.

A fim de permitir aprendizagem personalizada, conforme as características do aluno, notou-se a necessidade de adicionar "inteligência" nos CAIs, onde técnicas de Inteligência Artificial começaram a ser aplicadas nos softwares. Os primeiros sistemas com inteligência foram os ICAIs [2] - Inteligente CAI que, segundo a modelagem do estudante, tornavam a apresentação apropriada ao nível de conhecimento do aluno e ao seu modo de aprendizagem.

De uma maneira geral, adicionar técnicas de inteligência artificial a um sistema tutor inclui trabalhar de forma interdisciplinar com outras áreas de conhecimento da comunicação inteligente como psicologia e pedagogia. Atualmente, esta interdisciplinaridade vem sendo aplicada em Sistemas Tutores Inteligentes (STIs), que são baseados na aprendizagem interativa, que colocam o aluno como um ser ativo no processo.

[1] *Computer-Assisted Instruction*
[2] *Intelligent Computer-Assisted Instruction*

Os sistemas tutores inteligentes foram criados com o intuito de prover ensino e treinamento de forma computadorizada, diminuindo ou dispensando a necessidade da interação direta tutor-aluno e se adaptando ao modelo de aprendizado de cada aluno em particular.

Existem diversas definições para Sistemas Tutores Inteligentes. Segundo (GAMBOA; ANA, 2001), os STI são programas de Software que dão suporte às atividades de aprendizagem. (FREEMAN, 2000) define um sistema tutor inteligente como qualquer programa de computador que contenha alguma inteligência e que possa ser usado em aprendizagem. Na visão de (FOWLER, 1991), os STI são programas de computador com propósitos educacionais e que incorporam técnicas de Inteligência Artificial. Os STIs oferecem vantagens sobre os CAIs (Instrução Assistida por Computador), pois podem simular o processo do pensamento humano para auxiliar na resolução de problemas ou em tomadas de decisões.

De uma forma geral, pode-se considerar um Sistema Tutor Inteligente (STI) como qualquer sistema de computador que forneça instruções diretas personalizadas ou feedback aos alunos, sem a intervenção de seres humanos. Os STIs podem empregar diferentes tipos de tecnologias e seguem a teoria de "aprender fazendo".

2.1 História dos Sistemas Tutores Inteligentes (STI)

No início dos anos 60, diversos pesquisadores na área de computação como Alan Turing, Marvin Minsky e John McMarthy acreditavam que computadores com capacidade de pensamento logo se tornariam realidade, o que não aconteceu. Esses pesquisadores acreditavam que, a partir do momento que as máquinas pudessem pensar, elas seriam capazes de realizar qualquer tarefa associada com o pensamento humano, como por exemplo, o processo de ensino/aprendizagem.

O passo inicial na história dos STI foram os Sistemas de Instrução Assistida por Computador (CAI), que foram evoluindo ao longo dos anos e deram origem aos Sistemas Tutores Inteligentes, como mostrado na Figura 1.

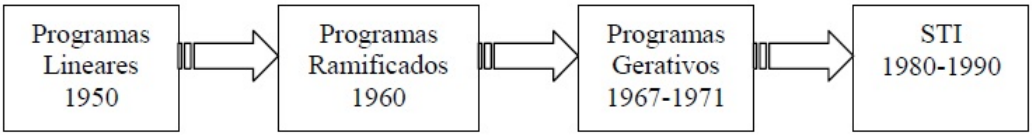


Figura 1: Evolução dos Sistemas de Ensino utilizando Computador (GAVIDIA; ANDRADE, 2003)

Os sistemas de Instrução Assistida por Computador foram chamados de programas lineares e caracterizavam-se por mostrar o conhecimento de uma forma linear, sem nenhuma mudança na ordem de ensino estabelecida pelo programador. O aluno seguia uma sequência finita e pré-determinada de passos sem estimular o raciocínio, frente a diferentes situações. Os erros dos alunos não eram considerados e a saída do programa era um frame, indicando se aluno havia aprendido até aquele ponto.

Os sucessores dos programas lineares foram os programas ramificados que eram mais adequados por ter feedback, sendo adaptado ao ensino para dar as respostas aos alunos. Diferentemente dos programas lineares, as respostas eram tratadas como aceitáveis ou parcialmente aceitáveis nos programas ramificados, ao invés de corretas ou incorretas. Os programas ramificados incorporaram "linguagem de autoria", que se caracterizavam por serem linguagens específicas e apropriadas para o desenvolvimento de materiais para softwares de instruções assistidas por computador de forma tratável pelo sistema. No entanto, apesar de algumas modificações, até então o enfoque dos programas era centrado no professor.

Os sistemas gerativos, também chamados sistemas adaptativos, surgiram com o início dos anos 70 com uma nova filosofia educacional, defendendo ensino adaptado às necessidades do aluno. Esses sistemas geram um problema, segundo o nível de conhecimento do aluno, constroem uma solução e analisam a resposta dada pelo aluno.

Em alguns domínios, como a aritmética, o próprio sistema podia gerar seu material de ensino através do computador. Para tanto, os sistemas precisam somente de uma estratégia de ensino geral para gerar uma árvore de possíveis interações com um número infinito de ramificações, incorporando alguma classificação de medida de dificuldade. No entanto, os sistemas gerativos não podiam ser aplicados a todo tipo de domínio, visto que a dificuldade aumentava consideravelmente em outras áreas.

Mesmo com esses avanços os sistemas de instruções assistidas por computador ainda possuíam o mesmo foco no professor, não se adaptando a cada tipo de aluno que o utilizasse. De uma forma geral, todos os sistemas ainda compartilhavam as características de serem cursos extensos, possuírem uma comunicação entre aluno e tutor mal definida, reagirem segundo modelos estabelecidos e independentes dos alunos, serem implementados sob medida e não evoluírem com o tempo.

Com o intuito de tratar as falhas dos sistemas gerativos, nos anos 80 surgiram os Sistemas Tutores Inteligentes (STI) que englobam técnicas de Inteligência Artificial para melhorar a forma de representação do conhecimento em um sistema inteligente. Na mesma

década, surgiram pesquisas na área de IA através da criação dos ICAI (Instruções Assistidas por Computador Inteligentes) que apresentaram uma estrutura diferenciada para trabalhar com domínios educacionais, utilizando técnicas de IA, Pedagogia e Psicologia Cognitiva no processo de ensino-aprendizagem. O termo Sistema Tutor Inteligente foi criado a fim de diferenciar sistemas ICAI de seus antecessores CAI.

2.2 Características de um Sistema Tutor Inteligente

Muitos autores definem as características de um sistema tutor inteligente de forma diferente. Para ser considerado um sistema inteligente, (JONASSEN; WANG, 1993) diz que o sistema deve ser passar em três testes:

1. O conteúdo deve ser inserido no sistema de forma que ele possa acessar as informações, realizar inferências e resolver problemas;
2. O sistema deve avaliar a aquisição do conhecimento pelo aluno;
3. As estratégias utilizadas devem reduzir a diferença entre o conhecimento do especialista e do aluno.

Segundo (URRETAVIZCAYA, 2001), um STI possui o conhecimento do domínio de forma restrita e claramente articulado, utilizam o conhecimento do aluno para adaptar o ensino, a sequência do não ensino não é predeterminado, realizam processos de diagnóstico mais adaptados ao aluno e permitem a comunicação tutor-aluno. Esses sistemas representam separadamente a matéria que ensina (modelo de domínio), as estratégias para o ensino (modelo pedagógico) e caracterizam o aluno (modelo do aluno) para prover ensino individualizado. Um STI necessita de uma interface de comunicação bem planejada e de fácil manipulação para favorecer a comunicação tutor-aluno. De uma forma geral, os Sistemas Tutores Inteligentes possuem as seguintes características listadas abaixo.

- O adjetivo “inteligente”marca o uso de técnicas de inteligência artificial nos STI, diferenciando-os dos sistemas de instruções assistidas por computador (CAI);
- A “inteligência”possibilita que esses sistemas sejam capazes de resolver os problemas apresentados aos alunos e explicar como isso foi feito;
- Permitem individualização na instrução, relacionando com o entendimento das metas e crenças do aluno;

- A apresentação do conteúdo ao aluno é organizada de forma “inteligente”pelo sistema, utilizando técnicas de IA como o planejamento, a otimização e busca;
- A interação em um STI é bastante variada; desde interações que reagem a ações realizadas pelos alunos até interações de assessoramento que observam o aluno na execução de uma tarefa apresentando conceitos importantes sem interferir em nada.

2.3 Arquitetura dos Sistemas Tutores Inteligentes

Um sistema tutor inteligente tem em sua arquitetura tradicional três componentes funcionais: o modelo de domínio, o modelo do aluno e o modelo tutor ou pedagógico, como mostrado na Figura 2. Esses modelos armazenam informações sobre o tema que será ensinado, sobre o aluno e o sobre as estratégias de ensino, que são combinadas para prover um ambiente ensino, conforme as necessidades do aluno. Segundo (AKHRAS; SELF, 2002), um STI combina dinamicamente as informações dos três componentes para tomar decisões adequadas em situações específicas em uma seção de tutoramento.

Essa arquitetura clássica é também conhecida como função tripartida, refere às funções associadas aos três modelos e trouxe grandes avanços à modelagem de ambientes educacionais pela separação do domínio e a forma de manipulação.

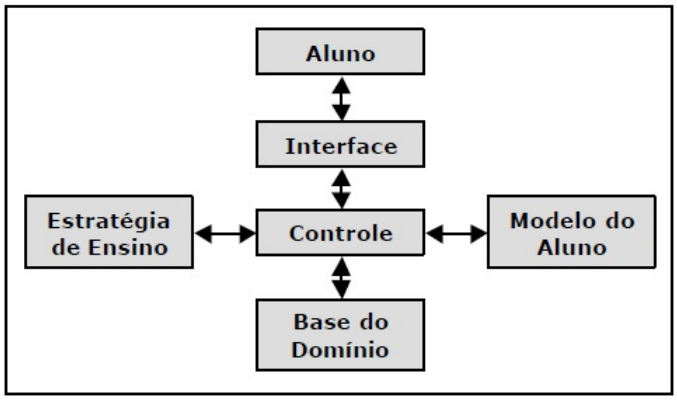


Figura 2: Arquitetura Clássica de um STI (RAABE, 2005)

Outros autores propõem uma arquitetura diferente para os sistemas tutores inteligentes, como a proposta por (MCTAGGART, 2001), mostrada na Figura 3.

Nessa arquitetura, McTaggart considera que o modelo especialista, instrucional e do aluno trabalham em conjunto para produzir um sistema instrucional, dirigido por um modelo especialista representando o conhecimento.

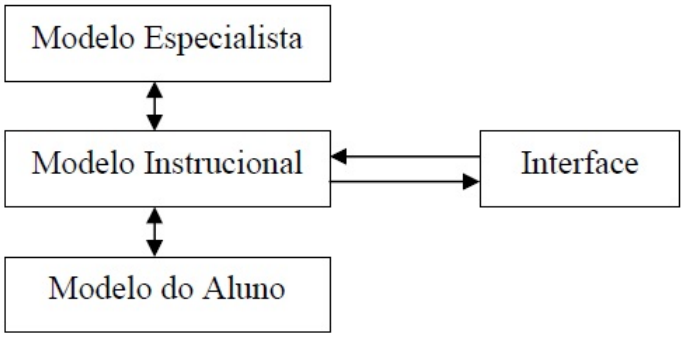


Figura 3: Arquitetura de um STI proposta por (MCTAGGART, 2001)

Uma outra variação de arquitetura de STI é a proposta por (WENGER, 1987), que identifica cinco componentes diferentes como mostrado na Figura 4.

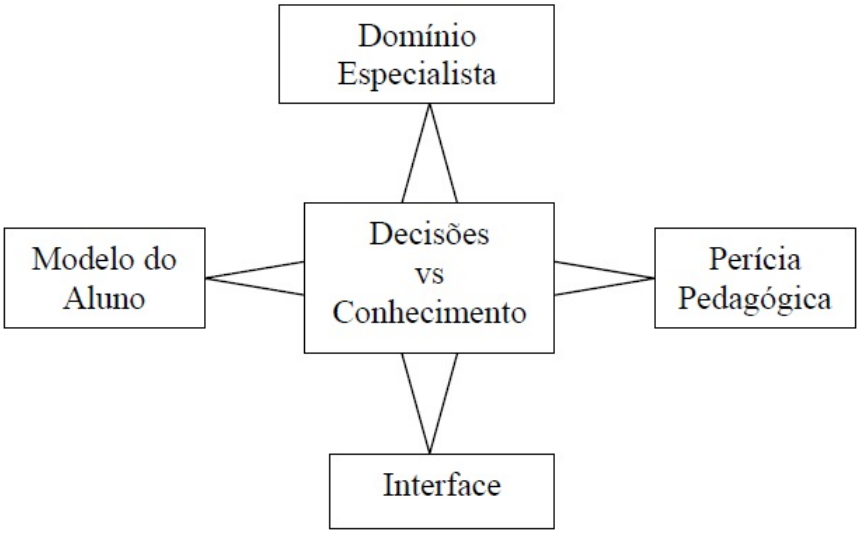


Figura 4: Arquitetura de um STI proposta por (WENGER, 1987)

Wenger abstrai os componentes acima das mais tradicionais definições de Engenharia de Software, descrevendo-o como "ferramenta de comunicação de conhecimento". (WENGER, 1987) propõe um estudo em conjunto das disciplinas de Inteligência Artificial, Ciência Cognitiva e Educação.

As diferentes arquiteturas propostas possuem basicamente quatro módulos funcionais, o modelo do aluno, modelo de domínio, modelo pedagógico e modelo de interface. É possível acrescentar outros componentes a esses módulos, como uma Base de Conhecimento, um módulo para tomara de decisão ou outros, dependendo do domínio do STI. A seguir são detalhados os quatro módulos básicos de um STI.

2.3.1 Modelo do Aluno

O modelo do aluno representa o conhecimento e as habilidades do aluno em um dado momento, representando o estado do conhecimento no momento de interação com o STI. (SELF, 1990) diz que a característica fundamental da pesquisa em STI é a atenção meticulosa ao aluno, visto que essas informações serão utilizadas para a tomada de decisão. Portanto, o objetivo fundamental do modelo do aluno é conhecer o estado atual de um aluno em uma situação de tutoria.

Segundo (GIRAFFA, 1999), o modelo do aluno é constituído de dados estáticos e dinâmicos que serão fundamentais para o tutor comprovar hipóteses sobre o aluno. O sistema deve ser capaz de inferir a melhor estratégia de ensino a ser aplicada através desse modelo em conjunto com o modelo de domínio.

Existem diversas técnicas que são utilizadas para a construção do modelo do aluno, como por exemplo, a inclusão de reconhecimento de padrões aplicados à história das respostas fornecidas pelo aluno, comparação da conduta de um aluno com a de um especialista, inclusão de preferências do aluno, indicação de objetivos particulares, entre outros. Segundo (COSTA; WERNECK, 1996), o modelo do aluno pode ser representado com base em alguns modelos de descrição:

- **Modelo Diferencial:** a resposta do aluno é comparada com a base de conhecimento. Essa modelagem divide o conhecimento em duas classes, a do conhecimento que se espera que o aluno adquira e a do conhecimento que se espera que ele possua. Nesse modelo, o conhecimento do aluno é um subconjunto do conhecimento do especialista;

- **Modelo de Overlay ou superposição:** o conhecimento do aluno é um subconjunto da base de conhecimento, implicando que a representação do conhecimento no modelo do aluno e do domínio seja a mesma. O modelo overlay assume que os erros do aluno ocorrem pela ausência de alguma informação na base de domínio, utilizando o pressuporto psicológico que comportamentos incorretos originam-se da presença de concepções incorretas na mente do aluno;

- **Modelo de Perturbação:** proposto inicialmente por (BROWN; BURTON, 1978), recebeu o nome de modelo BUGGY e assume que os erros do aluno são decorrentes da concepção errônea de algum conceito ou ausência dele;

- **Modelo de Simulação:** o ambiente possui um modelo de como o aluno pode ou

deve se comportar em uma dada situação e através desse modelo ele permite prever o comportamento futuro do aluno, baseado no seu comportamento durante a sessão de trabalho;

- **Modelo de Crenças:** consiste em um conjunto de crenças refletindo o grau que pensamos que o aluno entende sobre um conceito em particular.

Segundo Wenger (1987), o modelo do aluno possui três tarefas:

1. Colher dados sobre e do aprendiz, que podem ser explícitos (requisitando alguma resposta) ou implícitos (forma de interação);
2. Usar os dados para representar o conhecimento do aluno e processos de aprendizagem;
3. Representar os dados fazendo algum tipo de diagnótico, tanto no estado do conhecimento do aluno como em termos de seleção ótima de estratégias pedagógicas para apresentar depois a informação do domínio ao aluno.

A efetividade de um STI em apresentar ensino individualizado a cada aluno depende do tipo e da exatidão da informação sobre o aluno no modelo. Depende também do nível de sofisticação da representação do conhecimento no sistema e da eficácia dos métodos utilizados para extrair e incorporar novas informações sobre o aluno, visto que o conhecimento do aluno muda e o modelo deve incorporar essas mudanças dinamicamente.

O problema da modelagem do aluno já foi considerado intratável, no sentido de que não há uma possibilidade realista de construir modelos de aluno que atendam a todos os objetivos dos desenvolvedores de STI.

2.3.2 Modelo de Domínio

O modelo do domínio é o componente especialista do tutor e contém o conhecimento sobre o domínio que se deseja ensinar. Pode-se utilizar vários modelos de representação de conhecimento como redes semânticas, scripts, regras de produção, entre outras, escolhendo a que melhor atenda aos requisitos de representação e manipulação do raciocínio. Segundo (MCTAGGART, 2001), o modelo de domínio é um banco de dados organizado em conhecimentos declarativos e procedurais em um domínio específico.

O objetivo de um STI seria reproduzir essas estruturas de conhecimento na mente do aprendiz. O modelo de domínio está intimamente ligado ao modelo do aluno e o sistema busca no domínio do conhecimento exaustivamente comparando o modelo de aprendizagem do aluno com o domínio do conhecimento.

A forma como o modelo de domínio trabalha não é necessariamente a forma humana de resolver problemas, pois os humanos aplicam técnicas apropriadas para a resolução de problemas. Novos modelos de domínio têm surgido para simular a resolução humana de forma real, incorporando conhecimento reflexivo de fatos, procedimentos e qualidade que os seres humanos utilizam para estruturar seu conhecimento.

2.3.3 Modelo Pedagógico

O modelo pedagógico possui as estratégias e táticas de ensino. As estratégias constituem conhecimento sobre como gerar, a partir das informações de diagnóstico, monitoração e análise, uma sequência de táticas de ensino para apresentar um tópico a um determinado aluno. Uma estratégia de ensino deve definir:

1. Informações sobre quando interromper o curso de raciocínio ou aprendizagem do aluno;
2. Informações sobre quais tópicos apresentar e a ordem de apresentação;
3. Informações sobre a forma como apresentar o conteúdo. Está é a questão mais difícil, pois não há soluções gerais concretas.

Um método bastante utilizado é o chamado método socrático, onde o tutor ensina através de perguntas e diálogos, deixando o aluno tirar suas próprias conclusões. Outro modelo teórico utilizado é o modelo de treinamento (*coaching*) que utiliza atividades de entretenimento para transmitir conceitos relacionados, onde a aprendizagem é uma consequência indireta. Um terceiro modelo é o de hipertextos, onde o aluno navega em uma estrutura de hipertextos e explora o conteúdo segundo seus interesses, trabalhando de forma mais participativa e dinâmica.

O modelo instrucional ou estratégias de ensino contém conhecimento para tomar decisões sobre as táticas do tutor. Esse modelo é altamente dependente do processo de diagnóstico do modelo do aluno para a tomada de decisão sobre a forma e o conteúdo a ser apresentado ao aprendiz.

2.3.4 Modelo de Interface

Durante a interação, o sistema tutor apresenta o material de ensino e monitora o progresso do aluno através da recepção de suas respostas. Portanto, o modelo de interface é essencial para o sucesso do sistema.

Em um STI o modulo de interface deve oferecer diversos recursos na apresentação do conteúdo para evitar que o aluno se entedie; deve possibilitar facilidade na troca da iniciativa do diálogo onde o aluno possa intervir no discurso do tutor e vice-versa; o tempo de resposta deve estar dentro de limites aceitáveis e o monitoramento deve ser realizado de forma que não carregue o aluno com questionários excessivos.

O desenvolvimento de uma interface eficaz pode determinar o sucesso ou o fracasso da aplicação, visto que é através desse modelo que o aluno consegue interagir com todo o sistema. Atualmente, existem diferentes tipos de interfaces, cada qual pode ser empregada em um tipo específico de sistema, dado que as necessidades de interação podem ser alteradas conforme o conteúdo de ensino.

Sistemas de hipertexto e hipermídia têm sido utilizados no modelo de interface. Um sistema de hipertexto é um sistema de gerenciamento de bases de dados que permite conectar telas de informação através de ligações associativas definidas pelo usuário. O termo hipermídia é utilizado quando as informações conectadas empregam texto, material gráfico, recursos de vídeo, animação, som, etc. Essa variedade de recursos, aliada à possibilidade de percorrer o material de maneira vinculada à semântica do conteúdo, fazem dos sistemas de hipermídia uma ferramenta de alto potencial para apresentação do material instrucional em sistemas tutores inteligentes.

A interface é importante como meio de comunicação e deve ser desenvolvida com uma carga cognitiva adicional mínima, pois um estilo particular pode depender da habilidade do aprendiz e do conhecimento a ser aprendido.

O elevado grau de interconexão entre os quatro componentes da arquitetura de um STI fazem com que técnicas utilizadas em um modelo sejam aplicadas em outros. O uso de um modelo cognitivo para verificar erros do aprendiz (modelo do aluno) pode apresentar conhecimento a ele (modelo de domínio) e também comunicar conhecimento real usado durante a solução de problemas relevantes, melhor que em situações abstratas (modelo pedagógico). As possibilidades de comunicar conhecimento (modelo de interface) também fornecem oportunidades para implementar estratégias educacionais, fornecendo fundamentos para o aprendiz durante as fases iniciais que vão desaparecendo conforme o

aprendizado.

3 Raciocínio Baseado em Casos

Para a resolução de problemas no mundo real, é comum recorrermos a casos similares já ocorridos no passado para encontrar uma solução ou uma explicação para o problema corrente. O RBC faz uso da mesma filosofia, usando conhecimento de fatos que já ocorreram para solucionar novos problemas.

O RBC pode funcionar como um modelo cognitivo para se entender alguns aspectos do pensamento humano. Enquanto outras abordagens de Inteligência Artificial utilizam conhecimento genérico na forma de regras e roteiros, o RBC utiliza exemplos específicos concretos para representar o conhecimento, que é utilizado como base para resolução de outros problemas similares, conforme a Figura 5.

Segundo (WANGENHEIM; WANGENHEIM, 2003), o RBC é um enfoque para a solução de problemas e para o aprendizado baseado em experiência passada. RBC resolve problemas ao recuperar e adaptar experiências passadas - chamadas casos - armazenadas em uma base de casos. Um novo problema é resolvido com base na adaptação de soluções de problemas similares já conhecidas.

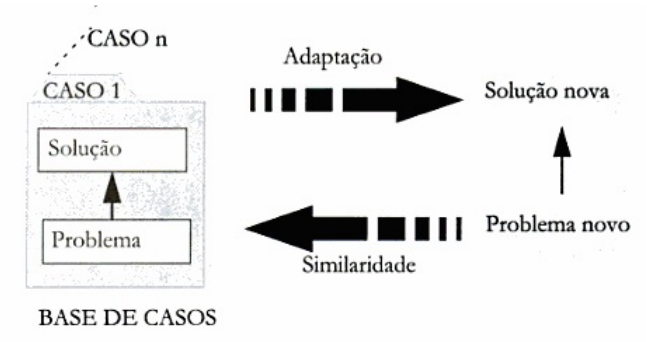


Figura 5: Modelo do Raciocínio Baseado em Casos (WANGENHEIM; WANGENHEIM, 2003)

A cada novo caso um sistema de RBC deve reconhecer e definir o problema atual a fim de encontrar a nova solução. O sistema deve ser capaz de verificar a similaridade entre o caso atual e os casos armazenados na base de dados para, a partir deles, adaptar uma

nova resolução para o problema dado.

O Ciclo de RBC, proposto por (AAMODT; PLAZA, 1994), é o modelo mais utilizado em sistemas inteligentes que utilizam a técnica de RBC. O ciclo de RBC, também chamado de *4R*, é composto basicamente por quatro tarefas, conforme pode ser observado na Figura 6.

A tarefa de "Recuperação"procura por casos similares ao novo problema na base de armazenamento de casos do sistema. A tarefa de "Reuso"utiliza os casos recuperados para encontrar uma solução adaptada ao problema proposto. Em seguida, a tarefa "Revisão"verifica se a solução dada pela tarefa anterior condiz com o escopo do problema em questão, validando ou corrigindo tal solução. Por último, é realizado a tarefa "Retenção"que armazena na base de casos o novo caso solucionado, que poderá servir como base de conhecimento para novos problemas posteriores.

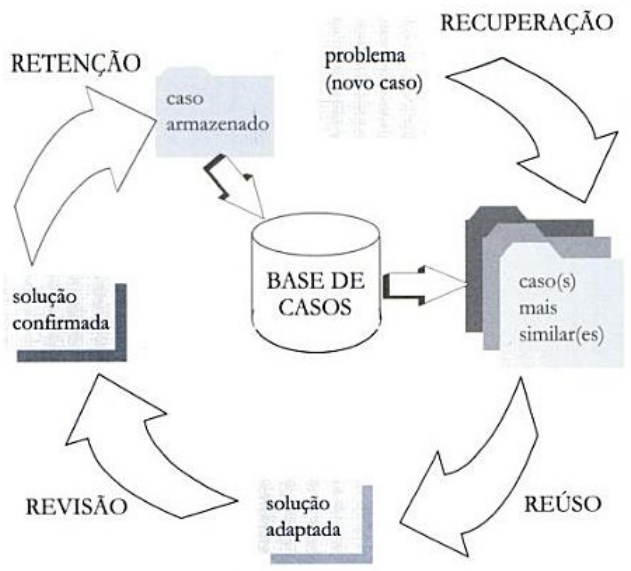


Figura 6: Ciclo de RBC (WANGENHEIM; WANGENHEIM, 2003)

Segundo o ciclo de RBC, a construção de uma aplicação inteligente deve seguir os seguintes passos:

- Representação dos Casos;
- Entrada dos Casos;
- Indexação dos Casos;
- Recuperação dos Casos;

- Comparação e Avaliação dos Casos e
- Adaptação.

Em aplicações de RBC, a qualidade da solução dada a um novo problema depende do número de casos já armazenados na base de casos, na habilidade de entendimento e comparação do novo problema com os casos já ocorridos e na capacidade de adaptação e avaliação da solução encontrada.

Assim como diversas aplicações de IA, não há um modelo de RBC universal adequado para qualquer domínio e um dos desafios é desenvolver modelos adequados a qualquer ambiente de aplicação, independente do domínio particular. Portanto, um modelo RBC é um conjunto de soluções coerentes para os problemas de representação do conhecimento através de casos juntamente com métodos de recuperação, reutilização, revisão e retenção de casos, além de métodos de medida de similaridade entre os casos propostos e os casos já armazenados.

3.1 História do Raciocínio Baseado em Casos

Os fundamentos de RBC em IA foram inspirados nos trabalhos sobre Memória Dinâmica e no modelo cognitivo de uma função central de lembrança de situações passadas e de padrões de situações estudadas por (SCHANK, 1982). Os autores propõem que o conhecimento de cada indivíduo acerca de diversas situações fica gravado na memória como roteiros que permitem a construção de expectativas sobre resultados esperados de ações planejadas através da inferência sobre relacionamentos causais entre ações.

No início dos anos 80, os roteiros foram propostos como uma estrutura de dados para a memória conceitual, que descrevem as informações sobre os eventos estereotipados. Um roteiro auxilia na análise de eventos ou atividades através da previsão das ações específicas que tipicamente serão executadas em uma determinada situação, partindo do pressuposto que determinadas coisas ocorrerão sempre, conforme esperado.

Os trabalhos de (GICK; HOLYOAK, 1980) na década de 80 sobre Raciocínio por Analogia deram origem a outra linha de pesquisa no campo de raciocínio baseado em casos, que contribuiu muito para a teoria do RBC. Pode-se considerar o RBC como uma especialização do raciocínio por analogia, diferindo um do outro principalmente nas premissas básicas de cada enfoque.

O objetivo do raciocínio analógico é a transformação e extensão do conhecimento proveniente de um domínio conhecido para dentro de um domínio com estrutura ainda não completamente compreendida. No RBC, somente problemas no âmbito de um mesmo domínio de aplicação são resolvidos, utilizando exemplos dentro do mesmo domínio.

Outros estudos de (SCHANK, 1982) exploraram o papel da memória de situações prévias e padrões de situação - ou pacotes de organização de memória (POMs) - na resolução de problemas e durante o aprendizado de situações. Suas teorias postulam que a compreensão, rememoração e o aprendizado de situações são processos mentais interligados.

Desenvolvido por (KOLODNER, 1992) na Yale University, o sistema CYRUS foi o primeiro a ser chamado de sistema de raciocínio baseado em casos. CYRUS é um sistema de perguntas e respostas que integra o conhecimento obtido da descrição de várias viagens e reuniões do ex-secretário de estado dos Estados Unidos, Cyrus Vance. Esse sistema é baseado no modelo de memória dinâmica e na teoria dos pacotes de organização de memória (POMs) para resolução e aprendizagem de novos casos. Cada POM armazena conhecimento geral a respeito das características compartilhadas pelos casos que organiza e contém uma estrutura organizacional que indexa esses casos em uma árvore.

A árvore contém nodos que podem ser um caso único, chamado eventos (EVn representado na Figura 7), ou sub-POMs (POMn representado na Figura 7) de um POM-pai que contém a lista de atributos compartilhados pela maioria dos casos agrupados nesse POM e possuem também uma estrutura de indexação. Através das respostas das perguntas realizadas, é feita uma navegação na árvore e o conjunto final de eventos é então casado com a entrada e considerado suficiente.

O modelo de memória utilizado no sistema CYRUS serviu como base de outros sistemas de RBC como o MEDIATOR que opera no domínio de solução de disputas, o PERSUADER que propõe a solução de disputas entre patrões e empregados e também para o CHEF que planeja novas receitas culinárias a partir de outras já existentes.

Outra abordagem de sistema de RBC foi desenvolvida por Porter na Universidade de Austin, que tratavam o problema de implementar o aprendizado de máquina do aprendizado conceitual para tarefas de classificação (WANGENHEIM; WANGENHEIM, 2003). Tais estudos resultaram no sistema PROTOS, que destacava a integração do conhecimento geral do domínio e conhecimento específico de casos em uma estrutura de representação unificada. A classificação de uma situação baseada em casos implementada no PROTOS é dada pelo objeto já conhecido que melhor casa com a nova entrada. O sistema então associa ao novo objeto a mesma classificação dada ao objeto já conhecido.

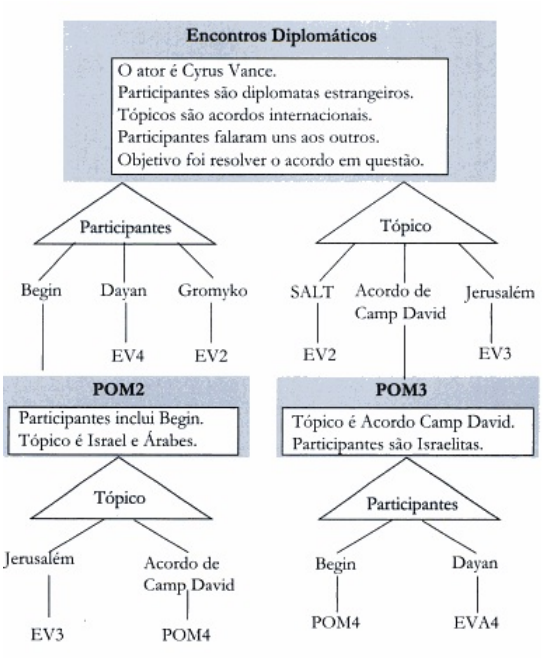


Figura 7: Representação do conhecimento no CYRUS (WANGENHEIM; WANGENHEIM, 2003)

Atualmente, o número de trabalhos comerciais e de pesquisa na área de RBC está aumentando rapidamente, o que pode ser constatado pelo número de sistemas comerciais, pelas conferências anuais realizadas sobre o assunto e também pelas ferramentas de desenvolvimento de sistemas baseados em RBC que já se encontram no mercado. O número cada vez maior de artigos e sistemas nessa área de IA devem garantir o contínuo interesse pelo RBC.

Um sistema RBC pode ser utilizado em diferentes áreas, como a de saúde, economia, previsão de tempo, sistemas caóticos e tantos outros. Temos, por exemplo, o sistema de raciocínio baseado em casos para recomendação de programa alimentar, desenvolvido por (JUNIOR et al., 2006), que utiliza informações de casos semelhantes no histórico do sistema para prever uma dieta para um dado usuário e ajudar na tomara de decisão médica. Há ainda o sistema RBC desenvolvido por (JAGANNATHAN et al., 2010) que dá suporte às decisões sobre o tratamento de radioterapia em pacientes com câncer no cérebro. Esse sistemas utiliza também outras técnicas de inteligência artificial, como a lógica fuzzy.

O sistema de predição do tempo, (RIORDAN; HANSEN, 2002), utiliza as técnicas de raciocínio baseado em casos para fazer a previsão do tempo em aeroportos e também para companhia particulares. Os casos já preditos são armazenados no sistema como base de dados de consulta. Além disso, esse sistema utiliza outras técnicas de inteligencia artificial,

como lógica fuzzy e função de esquecimento.

3.2 Representação dos Casos

O conhecimento em sistemas RBC é representado através de casos, que podem ser tratados como uma abstração de uma experiência descrita através de seu contexto e conteúdo. Os problemas devem ser representados de forma que contenham as metas a serem alcançadas, restrições e características da situação em que ocorre.

Segundo (WANGENHEIM; WANGENHEIM, 2003) um caso é uma peça de conhecimento contextualizado representando uma experiência ou episódio concretos. Contém a lição passada, que é o conteúdo do caso e o contexto em que a lição pode ser usada.

Um caso pode ser representado de diversas maneiras diferentes, dependendo da aplicação e da forma como será utilizado, assumindo representações e conteúdos diferentes. O conteúdo de cada caso está intimamente ligado ao domínio da aplicação e do objetivo do raciocínio. No entanto, para determinar uma representação de caso, duas medidas podem ser consideradas: a função de cada informação e a facilidade em adquirí-la.

Independente de domínio, um caso representa uma situação já experimentada, da qual se pode extrair informação para decidir casos futuros. Para decidir o que representar em um caso, deve-se levar em consideração a funcionalidade da informação, ou seja, garantir que somente as informações que terão utilidade para as tarefas realizadas pelo sistema serão representadas e a facilidade de aquisição da informação, ou seja, assegurar que somente informações não muito difíceis de adquirir sejam representadas.

De uma forma geral, um caso é representado através da descrição do problema ou situação do ambiente, quando o mesmo ocorre, da descrição da solução que representa os conceitos ou objetos que realizam as metas especificadas na sua descrição e a descrição do resultado, que especifica o que aconteceu como consequência da realização da solução proposta ou de como esta solução foi realizada.

- **Descrição do Problema:** a descrição do problema apresenta informações sobre um problema que deve ser resolvido ou uma situação que deve ser interpretada, classificada ou compreendida. Essa descrição deve conter dados suficientes para que outros casos similares já ocorridos sejam identificados, ou seja, devem conter dados de forma que seja possível julgar a aplicabilidade de um caso à uma nova situação. A descrição do problema envolve a identificação dos objetivos a serem atingidos

pela solução, das restrições impostas e dos atributos entre as partes do problema. Um caso armazenado na base de conhecimento deve incluir todas as informações consideradas para alcançar o objetivo específico.

- **Descrição da Solução**: a solução de um problema descreve a forma como ele foi solucionado, apresentando conceitos e objetos utilizados para atingir os objetivos específicos do caso solucionado, levando em consideração as restrições e atributos. A estrutura da solução varia conforme aplicação e pode englobar o conjunto de passos seguidos no raciocínio, justificativas para as decisões tomadas, soluções alternativas, soluções inaceitáveis, além de solução em si.

- **Descrição do Resultado:** a descrição do resultado descreve a consequência da aplicação da solução no problema inicial, incluindo um feedback do ambiente e uma interpretação do mesmo. O resultado pode englobar explicações sobre o sucesso ou fracasso da aplicação da solução, estratégias adotadas, se as expectativas foram alcançadas, formas de evitar o problema encontrado, etc.

Existem diversas formas de representar casos, dependendo do enfoque do sistema RBC e dos objetivos perseguidos, como a representação na forma de atributo-valor, na forma orientada a objetos, através de redes semânticas ou em estruturas de árvores e grafos. A escolha da maneira de representação dos casos define a forma como o conhecimento será formulado.

3.3 Indexação de Casos

Um sistema de RBC só é eficaz quando possui habilidades para recuperar e selecionar casos relevantes de forma rápida e precisa. O entendimento de quando um caso deve ser recordado em situações futuras similares constitui um problema de indexação. A indexação é vista como um problema de escolher características que possam ser usadas como índices para os casos armazenados na memória, de forma que possam ser recuperados, quando necessário, identificando casos que possuem lições a ensinar. No entanto, a indexação pode também ser vista como um problema de organizar a memória de casos, fazendo com que a recuperação seja realizada de forma precisa e eficiente.

Existem diversas formas de organização de casos na memória, entre as quais o modelo de estrutura linear, o modelo de estrutura hierárquica e também a organização pelo modelo de categoria e exemplar. Na estrutura linear, os casos são armazenados sequencialmente

em uma lista, vetor ou arquivo e as características de cada caso são indexadas independente umas das outras. Esse tipo de armazenamento apresenta a vantagem de ser fácil aprender com casos recém resolvidos, pois é simples e barato adicionar casos à memória, bastando inserí-los no final ou início da estrutura.

A estrutura hierárquica faz com que somente um pequeno subconjunto dos casos seja considerado na recuperação. A hierarquia é geralmente obtida com a ajuda de métodos de agrupamento indutivo que provêm uma forma de agrupar casos através de características que são partilhadas por um grande número de itens. Essa forma de agrupamento utiliza o conceito de árvores para separar os grupos, onde cada nó interno da rede mantém características partilhadas pelos casos abaixo dele, os nós sem aquelas características estão representados em nós irmãos ou em nós abaixo dos nós irmãos e os nós folhas são os próprios casos armazenados, como pode ser observado na Figura 8 abaixo.

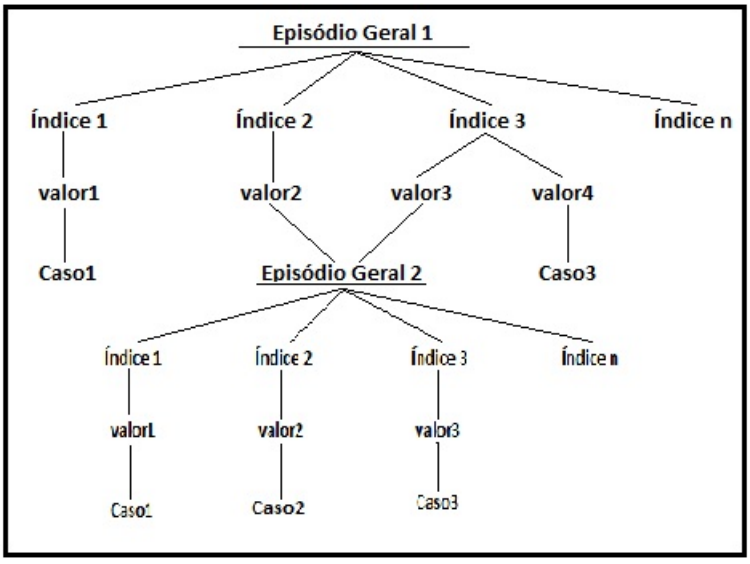


Figura 8: Exemplo de uma Árvore de Características Partilhadas

A organização da memória pode também ser feita através do modelo de categoria e exemplar, onde os casos são referenciados como exemplares. A memória do caso é incluída em uma estrutura de rede de categorias, casos e ponteiros de índices. Cada caso é associado a uma categoria e um índice pode apontar para um caso ou categoria. Os índices são de três tipos: links de características apontando de descritores de problemas (recursos) para casos ou categorias (chamados de lembretes), links de casos apontando de categorias para seus casos associados (chamados de links exemplares) e links de diferença apontando dos casos para seus casos vizinhos que diferem em uma ou em um pequeno número de características. Uma característica é descrita por um nome ou um valor e os

exemplares de uma categoria são ordenados de acordo com seu grau de prototipicidade na categoria. A Figura 9 mostra parte da estrutura da organização memória.

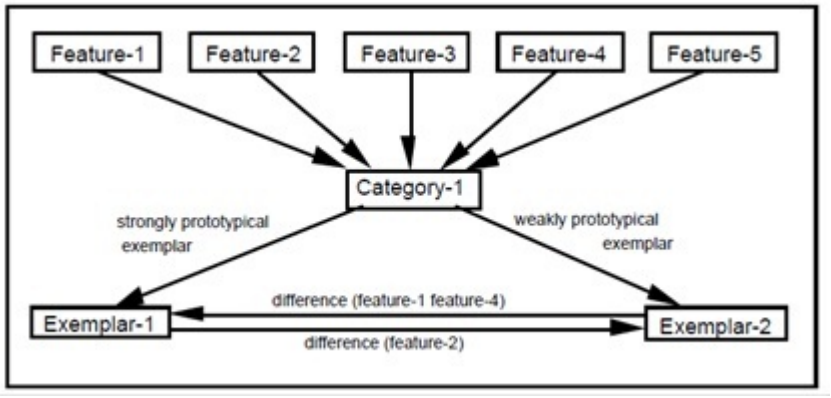


Figura 9: Estrutura de Categoria, Características e Exemplares (AAMODT; PLAZA, 1994)

Nessa organização de memória, as categorias estão inter-relacionados dentro de uma rede semântica, que contém os critérios e os estados intermediários referenciados por outros termos.

3.4 Similaridade entre Casos

As medidas de similaridade visam recuperar, da base de dados, casos que sejam úteis na resolução do novo problema apresentado. Os casos selecionados tendem a ser facilmente adaptados para o problema atual ou a possuírem a mesma solução através da semelhança entre os atributos descritores do caso analisado. Segundo (DELPIZZO, 1997), a similaridade tem extrema importância em um sistema RBC, pois é à partir dele que o processo de raciocínio se torna viável.

Em sistemas de RBC, são utilizadas diferentes métricas de similaridades. As métricas de similaridade são funções que determinam, numericamente, o grau de semelhança entre dois casos distintos, comparando um a um os atributos que descrevem as particularidades do caso. Essas funções atribuem pesos de importância diferente a cada atributo, que diferenciam a relevância de cada descritor na avaliação geral da similaridade.

Conforme (WANGENHEIM; WANGENHEIM, 2003), o grau de similaridade entre os casos serve para determinar uma ordem parcial sobre os casos da base de conhecimento em relação à utilidade dos mesmos na resolução do problema que o sistema está tentando resolver.

Para a determinação do grau de similaridade entre dois casos são utilizados, basicamente, duas funções de similaridade: as medidas de similaridade local e a global. Existem diversas funções de similaridade já definidas, tanto locais como globais. Nas seções seguintes serão apresentadas algumas dessas funções.

3.4.1 Medidas de Similaridade Local

A medida de similaridade local trata a semelhança dos casos a nível de atributos individuais. (WANGENHEIM; WANGENHEIM, 2003) define uma medida de similaridade local como a função descrita na Equação 3.1,

$$sim(x_i, y_i) : (U \times U \rightarrow [0, 1]), \tag{3.1}$$

que determina a similaridade entre duas entidades de informação, onde U é o conjunto de casos representados no sistema.

Segundo (WANGENHEIM; WANGENHEIM, 2003), as medidas de similaridade local são definidas conforme o tipo de dado do atributo, que podem ser numéricos, simbólicos, *strings*, entre outros. Além disso, essa medida é dependente do contexto da aplicação, variando conforme a aplicaçao RBC. A seguir serão apresentados algumas medidas de similaridade local que consideram diversos tipos de atributos como números, símbolos, *string*, etc.

3.4.1.1 Similaridade Simples

A forma mais simples de calcular a similaridade entre atributos correspondentes assume valores binários, sendo 1 quando os valores dos atributos são iguais e o quando não são. A similaridade simples é dada pela Equação 3.2.

$$locSim(x_i, y_i) = \begin{cases} 1 & x_i = y_i \\ 0 & x_i \neq y_i \end{cases} \tag{3.2}$$

3.4.1.2 Similaridade entre atributos numéricos

Para atributos numéricos, a medida de distância local genérica entre dois números pode ser utilizada como parâmetro para cálculo da similaridade. Sendo x_i o valor do atributo do caso C e y_i o valor do atributo do caso Q, a distância entre os valores é dado

pelo módulo da diferença, $|x_i - y_i|$.

O valor calculado através do módulo da diferença entre esses valores pode ser utilizado para diferentes tipos de funções para a determinação da similaridade local. A seguir, são apresentadas algumas dessas funções.

Função Escada

A função escada define um limiar, S, que determina quando um atributo será considerado similar ao outro. A similaridade será 1 se o módulo da diferença entre os valores for menor que o limiar definido, ou 0 se essa diferença for maior, conforme Equação 3.3.

$$locSim(x_i, y_i) = \begin{cases} 1 & |x_i - y_i| \leq S \\ 0 & |x_i - y_i| > S \end{cases} \tag{3.3}$$

Função Linear

A função linear considera que, conforme a distância entre os valores dos atributos x_i e y_i diminui, a similaridade entre eles aumenta. Essa função considera, além do módulo da diferença entre os valores, o tamanho do intervalo do domínio. A Equação 3.4 apresenta a função linear para similaridade local, com ls representando o limite superior e li o limite inferior.

$$locSim(x_i, y_i) = \begin{cases} 1 & x_i = y_i \\ 1 - \dfrac{|x_i - y_i|}{ls - li} & x_i \neq y_i \end{cases} \tag{3.4}$$

3.4.1.3 Símbolo Ordenado

Além de valores numéricos, os atributos dos casos podem assumir valores descritivos que possuem uma determinada ordem de relevância. A partir da ordenação dos símbolos, pode-se atribuir valores ordinais a cada um, utilizando valores equidistantes ou não.

Por exemplo, em uma base de casos de sintomas de pacientes o atributo *febre* pode assumir os valores simbólicos *baixa, média, alta*. Utilizando valores equidistantes unitários poderíamos atribuir os seguintes valores ordinais aos símbolos: { *baixa* $\rightarrow$ 1, *média* $\rightarrow$ 2, *alta* $\rightarrow$ 3 }. Ainda utilizando o mesmo exemplo, poderíamos associar ao atributo *febre* os seguintes valores não equidistantes: { *baixa* $\rightarrow$ 1, *média* $\rightarrow$ 3, *alta* $\rightarrow$ 7 }.

À partir da definição de valores numéricos aos símbolo as funções de similaridade numéricas podem ser aplicadas para a determinação da semelhança entre os atributos,

como a função escada ou função linear.

3.4.1.4 Símbolo não-Ordenado

Diferentemente dos símbolos ordenados, os símbolos não ordenados não apresentam nenhuma ordem definida para a associação de valores numéricos. Nesse caso, a semelhança entre os valores simbólicos pode ser definida através de uma matriz de similaridade.

A matriz de similaridade considera todos os pares possíveis de valores que um dado atributo pode assumir em um caso e associa um valor de semelhança. As medidas podem ser simétricas ou não, ou seja, a medida $sim(x_i, y_i)$ pode assumir o mesmo valor que a medida $sim(y_i, x_i)$ ou não. Utilizando medidas simétricas obtemos uma matriz de similaridade diagonal, enquanto que para medidas assimétricas obtemos uma matriz quadrada, visto que nos dois sentidos os valores podem ser diferentes.

3.4.1.5 Strings

Um dado atributo pode assumir valores textuais com um número variável de palavras. É aconselhável substituir *strings* por valores simbólicos sempre que possível, pois o cálculo da similaridade entre atributos descritos por *strings* é uma tarefa árdua, no entanto existem alguns métodos de mensuração desse valor.

Correspondência Exata

Dois *strings* são considerados similares se forem escritos da mesma forma. Uma das formas mais simples de medida de similaridade entre *strings* consiste em compará-las, atribuindo valor de similaridade 1 quando são iguais e valor 0 quando diferem.

Correção Ortográfica

O método de similaridade de correção ortográfica considera o número de caracteres idênticos, ponderando com o número total de caracteres no *string*-consulta. Esse método pode ser utilizado em atributos que recebem *strings* constituídos por apenas uma palavra.

Contagem de Palavras

O método de similaridade de contagem de palavras conta o número de palavras idênticas nos dois casos em questão, o de consulta e o caso estudado. O valor da similaridade entre os atributos dos casos é contabilizado através da divisão do número de palavras idênticas pelo número total de palavras no *string*-consulta.

3.4.1.6 Métodos Híbridos

Além dos métodos de similaridade local descritos acima existem outros que podem ser desenvolvidos através de diferentes técnicas de inteligência artificial. Uma das técnicas de IA mais utilizadas é a de lógica fuzzy que tenta discretizar atributos que possuem valores não numéricos, considerando a nebulosidade existente nessa transição.

Atualmente existem diferentes sistemas que utilizam lógica fuzzy para calcular a similaridade local entre os atributos dos casos em estudo. O sistema de predição de tempo de (RIORDAN; HANSEN, 2002) utiliza uma função fuzzy para discretizar os atributos considerados nos casos e, posteriormente, calcular a similaridade entre eles.

O trabalho de (JAGANNATHAN et al., 2010) também utiliza uma função fuzzy para discretizar os atributos que compõem os casos. Esse trabalho utiliza funções diferenciadas para cada atributo e associa pesos diferentes (negativos e positivos) para cada um deles, definindo a relevância de cada um na composição da similaridade global.

3.4.2 Medidas de Similaridade Global

A medida de similaridade global mede a similaridade geral entre dois casos, ou seja, determina o quanto um dado caso é semelhante ao outro, considerando todos os seus atributos. (WANGENHEIM; WANGENHEIM, 2003) define a medida de similaridade global como a função descrita na Equação 3.5

$$sim(x, y) : (U \times U \rightarrow [0, 1]) \tag{3.5}$$

que determina a similaridade entre a pergunta e um caso sob consideração de todos os índices, onde U é o conjunto de todos os casos representados no sistema.

3.4.2.1 Distância do Vizinho-mais-Próximo

A distância do vizinho-mais-próximo utiliza cálculos bem simples e considera os casos como pontos em um espaço multidimensional, onde a dimensão do espaço é determinada pela quantidade de atributos dos casos. Dado que cada atributo assume um índice, a similaridade pode ser determinada através da distância espacial entre os pontos, como pode ser observado na Figura 10.

A distância do vizinho-mais-próximo pode ser calculada através da Equação 3.6.

Figura 10: Distância do Vizinho-mais-Próximo

$$sim(Q, C) = \sum_{i=1}^{n} f(Q_i, C_i). \tag{3.6}$$

A função f determina uma medida de similaridade local entre os casos C e Q.

3.4.2.2 Distância do Vizinho-mais-Próximo Ponderado

A Distância do vizinho-mais-próximo ponderado é uma variação da distância do vizinho-mais-próximo, que considera a importância de cada índice do caso, determinando qual atributo possuirá peso maior no cálculo da similaridade.

A distância do vizinho-mais-próximo ponderado pode ser calculada através da Equação 3.7.

$$sim(Q, C) = \sum_{i=1}^{n} f(Q_i, C_i) \times w_i. \tag{3.7}$$

A função f determina uma medida de similaridade local entre os casos C e Q e w_i indica o peso desse atributo no cálculo da similaridade.

3.4.2.3 Distância Euclidiana

A distância euclidiana realiza o cálculo da distância real entre dois pontos quaisquer em um dado espaço, onde a dimensão também é definida pela quantidade de atributos válidos na mensuração da semelhança. Essa medida de similaridade é dada pela Equação 3.8

$$d(Q, C) = \sqrt{\sum_{i=0}^{n} (q_i, c_i)^2}. \tag{3.8}$$

3.4.2.4 Distância Euclidiana Ponderada

Assim como a distância do vizinho-mais-próximo pode ser estendida, a distância euclidiana também pode distribuir pesos diferentes entre seus atributos de maior impacto. No cálculo da distância euclidiana ponderada é utilizado um índice w_i interligado a cada

x/y	1	0
1	a	b
0	c	d

Tabela 1: Tabela de Contingencia

atributo, indicando o quão importante o mesmo é em relação à similaridade do caso em geral. A distância euclidiana ponderada é dada pela Equação 3.9.

$$d(Q, C) = \sqrt{\sum_{i=0}^{n} w_i \times (q_i, c_i)^2}. \tag{3.9}$$

3.4.2.5 Distância de Manhattan

A distância de Manhattan também utiliza um modelo geométrico para determinar a distância entre dois casos e pode ser chamada de *métrica do quarteirão* e considera todos os atributos de forma homogênea. Considerando um modelo de quarteirões urbanos com ruas perpendiculares, a distância de Manhattan analisa o menor caminho possível entre dois pontos e é dado pela Equação 3.10.

$$d(Q, C) = \sum_{i=0}^{n} |q_i, c_i|. \tag{3.10}$$

3.4.2.6 Coeficiente de Casamento Simples

A métrica de coeficiente de casamento simples utiliza tabelas de contingências para determinar a similaridade entre dois casos C e Q distintos.

Tabelas de Contingências

As tabelas de contingências são utilizadas para registrar e analisar o relacionamento entre duas ou mais variáveis de escala nominal. Essas tabelas registram a ocorrência dos atributos segundo os valores encontrados nos dois casos disjuntos.

O caso mais simples de uma tabela de contingência é dado pela relação entre dois casos com apenas dois valores possível, por exemplo 0 e 1, como mostrado na 1 abaixo.

Os valores a e d da 1 representam os valores correspondentes da tabela, enquanto que os valores b e c, os valores divergentes. Podemos definir um n como sendo o valor máximo

que um dado índice da tabela pode assumir e é dado por $n = a + b + c + d$.

Portanto, o coeficiente de casamento simples considera apenas o número de valores correspondentes entre os casos em questão e pode ser calculado à partir da Equação 3.11.

$$sim(Q, C) = 1 - \frac{b + c}{n} = \frac{a + d}{a + b + c + d}. \tag{3.11}$$

3.4.2.7 Modelo de Contraste

O modelo de contraste calcula a medida de similaridade entre dois casos distintos C e Q levando em consideração os atributos que ambos os casos possuem em comum e também os atributos divergentes entre eles.

Para tanto são criados três conjuntos diferentes, descritos a seguir.

1. $C \cap Q$: Atributos que ocorrem tanto em C quanto em Q;
2. $C - Q$: Atributos que ocorrem somente em C;
3. $Q - C$: Atributos que ocorrem somente em Q.

Os três conjuntos calculados podem possuir pesos diferentes e, portanto, são ponderados com os índices a, b e c no cálculo da similaridade total. Além disso, o peso de cada atributo wi dos casos são considerados no cálculo de cada conjunto parcial. A medida de similaridade atribuída através do modelo de contraste é dado pela Equação 3.12.

$$sim(C, Q) = \left(a \sum_{i \in C \cap Q} w_i \right) - \left(b \sum_{i \in C - Q} w_i \right) - \left(c \sum_{i \in Q - C} w_i \right) \tag{3.12}$$

3.4.2.8 Modelo de Vetor

O modelo de vetor é também uma medida de similaridade que considera um espaço geométrico. Esse método considera todos os atributos relevantes do caso representando-os em um vetor. Para tanto, os atributos descritos através de *strings* são normalizados afim de se obter um descritor numérico.

O vetor $c = [c_1, c_2, ..., c_n]$ representa o vetor do caso em estudo e o vetor $q = [q_1, q_2, ..., q_n]$ o caso a ser comparado. A medida de similaridade entre ambos os casos é dado pela distância geométrica entre os vetor, calculado através da Equação 3.13.

$$sim(c, q) = \frac{c.q}{|c|.|q|} = \frac{\sum_{i=1}^{n} (c_i, q_i)}{\sqrt{\sum_{i=1}^{n} c_i^2} \sqrt{\sum_{i=1}^{n} q_i^2}} \tag{3.13}$$

3.5 Recuperação dos Casos

Na resolução de um novo problema, o sistema RBC recupera da memória o caso que se assemelha mais à nova situação e apresenta uma solução baseado no raciocínio do caso recuperado para auxiliá-lo na resolução. A tarefa de recuperação começa com uma descrição do problema e termina quando é encontrado um caso prévio com melhor ligação. Para realizar essa tarefa, a recuperação é dividida nas subtarefas de identificar as características do problema e inicializar ligação, que consiste nas tarefas de selecionar um conjunto de casos similares e selecionar o melhor entre eles.

- **Identificar as características:** identificar um problema pode envolver simplesmente a observação de suas entradas, mas muitas vezes uma abordagem mais elaborada é utilizada para entender o contexto do problema. Entender um problema envolve retirar ruídos dos descritores para inferir características relevantes, checar com outros valores de características que fazem sentido no contexto, gerar expectativas de outras características, etc. Outros descritores diferentes das entradas podem ser inferidas usando um modelo de conhecimento geral ou recuperando uma descrição de problema similar da base de casos e usando essas características como esperadas. A verificação de expectativas pode ser feita dentro do modelo de conhecimento(casos e conhecimentos gerais) ou solicitando ao usuário.

- **Selecionar conjunto de casos similares:** a seleção de um conjunto de casos correspondentes é feita usando os descritores do problema (características de entrada) como índice dos casos em memória de forma direta ou indireta. Existem três formas básicas de recuperar um caso: seguindo diretamente os ponteiros dos índices das características, procurando em uma estrutura de índice ou procurando em um modelo de conhecimento de domínio geral.

Os casos podem ser recuperados apenas por características de entrada ou também pelas características inferidas da entrada. Casos armazenados na base que possuem ligação com todas as características são claramente bons candidatos para recuperação, mas casos que possuem uma fração de características com ligação também podem ser recuperados, desde que haja uma forma de avaliar quão similar ele é do problema.

Avaliações de similaridade podem ser intensivas no conhecimento, tentando entender o problema profundamente analisando os objetivos e restrições. Uma outra opção é acrescentar pesos aos descritores do problema de acordo com sua importância na caracterização do problema, durante a fase de aprendizagem.

- **Selecionar melhor caso:** a partir do conjunto de casos previamente selecionados, é escolhido o melhor caso. O melhor caso é determinado pela avaliação do grau de similaridade mais perto do caso apresentado. Essa avaliação é feita por uma tentativa de gerar explicações para justificar as características não idênticas, com base no conhecimento semântico da rede.

  A tarefa de seleção gera consequências e expectativas de cada caso recuperado e tenta avaliar as consequências e justificas as expectativas. Isso é feito através de um sistema de conhecimento de domínio geral ou com a ajuda do usuário. Posteriormente, os casos são classificados segundo algumas métricas, selecionando o caso que possui a explicação mais semelhante ao novo problema.

3.6 Reutilização dos Casos

A reutilização do caso recuperado como solução no contexto do novo caso foca dois aspectos: na diferença entre o caso passado e corrente e na parte do caso recuperado pode ser transferido para o novo. Para solucionar o problema corrente a tarefa de reutilização de casos utiliza as sub-tarefas de cópia e adaptação do caso anterior.

- **Cópia:** em tarefas simples de classificação as diferenças entre casos corrente e recuperado são abstraídas (relevando apenas as semelhanças) e a solução do caso recuperado é transferida para o novo caso. Esse tipo de reuso é trivial, no entanto outros sistemas devem levar em consideração as diferenças, exigindo um processo de adaptação das mesmas.

- **Adaptação:** existem duas formas principais de reutilizar casos passados: reutilização da solução do caso passado (reuso transformacional) e reutilização do método que construiu a solução do caso passado (reuso derivacional).

  No reuso transformacional a solução do caso passado não é diretamente a solução para o novo caso, mas existe algum conhecimento na forma de operadores transformacionais T que podem ser aplicados na velha solução transformando-a na solução do novo caso. Uma forma de organizar esses operadores T é indexando-os em torno das diferenças detectadas entre o caso corrente e o recuperado. O reuso transformacional não se atenta em como um problema é resolvido, mas foca na equivalência de soluções, e isso requer um forte modelo dependente de domínio na forma de operadores transformacionais T juntamente com um regime de controle para organizar as aplicações operadoras.

  O reuso derivacional se atenta em como o problema foi resolvido no caso recuperado. O caso recuperado armazena informação sobre o método utilizado para resolver o problema, incluindo uma justificativa do operador utilizado, objetivos considerados, alternativas geradas, caminhos procurados que falharam, etc. O reuso derivacional utiliza o método do caso antigo e substitui no novo contexto. Durante a substituição alternativas de sucesso, operadores e caminhos serão primeiramente explorados, enquanto caminhos que falharam serão evitados; novos objetivos são perseguidos baseados no modelo do caso antigo.

3.7 Revisão de Casos

A tarefa de reutilização de casos pode gerar soluções corretas ou não. Quando a solução gerada é incorreta, o sistema tem a oportunidade de aprender com o erro ocorrido. A fase de revisão consiste nas tarefas de avaliar e reparar as soluções quando necessário. Em ambos os casos o sistema aprende com o novo caso resolvido.

- **Avaliação de Soluções:** essa tarefa toma os resultados da aplicação da solução encontrada em um ambiente real (consultando um professor ou atuando em um ambiente real, que pode envolver passos externos ao sistema RBC). A avaliação das soluções em ambiente real pode levar algum tempo e o sistema marca esses casos como não avaliados. Outra forma de avaliação da solução é feita através da aplicação em programas de simulação habilitados a gerar uma solução correta.

- **Reparação de Falhas:** a reparação envolve a detecção de erros da solução corrente e a recuperação ou geração de explicações de soluções que não alcançam certas metas. Essas explicações podem ser armazenadas em uma memória de falha como uma forma de prever possíveis deficiências, e posteriormente evitar e manipular erros. Uma segunda fase da revisão repara a solução, utilizando as explicações para modificar a solução de modo que falhas não ocorram.

3.8 Retenção de Casos

Essa fase do processo retém o que é útil do novo problema resolvido na base existente de conhecimento. O aprendizado com o sucesso ou falha da solução proposta é disparado na avaliação ou possível reparação do caso. Essa tarefa envolve selecionar quais informações do caso armazenar, de que forma armazenar, como indexar esse caso para posterior recuperação em problemas similares e como integrar o novo caso na estrutura da memória.

- **Extração:** independente da forma como o problema foi solucionado o sistema RBC será atualizado. Se for resolvido usando um caso prévio, um novo caso será construído ou o caso anterior será generalizado englobando o presente caso. Se a solução utilizou outros métodos, um novo caso será construído. Em ambos os casos, é necessário escolher o que será utilizado como fonte de aprendizado e muita das vezes são utilizadas os descritores e a solução do caso. No entanto, uma explicação ou outra forma de justificativa do porque essa solução é apropriada ao problema pode ser armazenada juntamente com o caso. De maneira análoga, falhas podem ser extraídas e retidas no sistema afim de evitar erros futuros.

- **Índice:** o problema de indexação é bastante focado em sistemas baseados em casos e envolve a decisão de que tipo de indexação utilizar para futuramente recuperar casos e como estruturar o espaço de procura de índices.

- **Integração:** Esse é o último passo na atualização da base de conhecimento. Se não foi construído um novo caso e índice, esse passo é o principal na retenção. Modificando a indexação de casos existentes os sistemas RBC aprendem a se tornar avaliadores de similaridades, pois a seleção de índices é uma parte importante do aprendizado do sistema. Índices fortes ou importantes para um caso ou solução em particular são ajustados de acordo com o sucesso ou falha do problema de entrada. Características julgadas relevantes para a recuperação são associadas ao caso com

maior peso, enquanto que características menos importantes são ligadas fracamente a ele. Dessa forma, a estrutura de índice tem o papel de ajustar e adaptar o processo de uso da memória.

4 Framework para Estruturar a Base de Conhecimento

Devido a busca contínua por ensino personalizado surgiram os sistemas tutores inteligentes. Esses sistemas podem utilizar variadas técnicas de inteligência artificial para prover um ambiente dinâmico e adaptativo, visando oferecer aos seus usuários um aprendizado fácil e voltado ao seu perfil e deficiências.

Um sistema tutor inteligente é composto de quatro módulos distintos, cada qual representando uma função diferente dentro do sistema. Os quatro módulos de um STI - modelo de domínio, modelo do usuário, modelo de estratégia e modelo de interface - são representandos na Figura 11. Cada um desses módulos pode utilizar técnicas diferentes de inteligência artificial, tentando aprimorar a inteligência e a personalização do sistema.

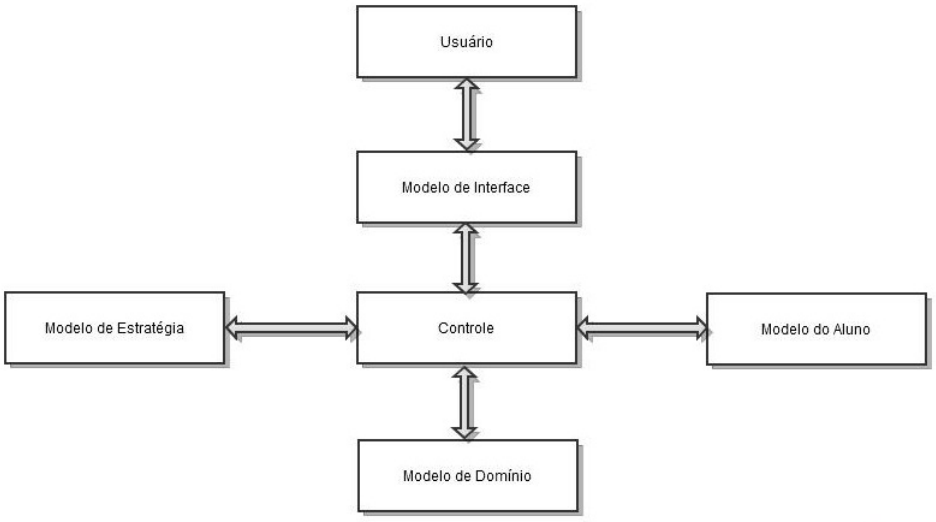


Figura 11: Módulos de um Sistema Tutor Inteligente. Adaptado de (RAABE, 2005)

O raciocínio baseado em casos pode ser utilizado para organizar qualquer um desses módulos. No modelo do aluno, o RBC pode ser usado para conter os diferentes perfis de usuário que já utilizaram o sistema e, a partir desses casos, recuperar um perfil que mais

se assemelhe ao usuário corrente.

No modelo de domínio, toda a informação a ser passada ao aluno pode ser organizada em casos, contendo itens atômicos, que são necessários para o entendimento de um assunto em particular. Há sistemas que utilizam a simulação como meio tutoramento, proporcionando ambientes bem próximos do real, para que o usuário tenha conhecimento das diferentes situações que podem ocorrer, além de terem contato com os diferentes recursos que dispõe aquele sistema específico. Nesses sistemas, o modelo de domínio pode conter diferentes casos de simulações já realizadas, que podem ser recuperadas e apresentadas para um novo aluno.

Similarmente, no modelo de estratégia, o RBC pode ser empregado para armazenar diferentes táticas de ensino. O modelo de interface também pode ser organizado de forma a conter diferentes interfaces e modos de navegação disponíveis para ser apresentado ao usuário. O modelo de controle ficaria encarregado de manipular as informações dos módulos unitários e mostrar ao usuário o conteúdo apropriado, buscando o conteúdo no módulo de domínio segundo o perfil do aluno, a melhor estratégia existente para esse perfil e levando em consideração as interfaces existentes no sistema.

Um sistema tutor inteligente que utilize técnicas de RBC nos quatro módulos pode ser representado como na Figura 12. A base de conhecimento é dividida em duas partes: uma contendo o material a ser apresentado em fase de aprendizado teórico e outra com informações sobre as simulações realizadas. O modelo do aluno é organizado em casos que representam diferentes perfis de usuários. O modelo de estratégia pode receber como entrada o perfil do usuário e também dados sobre a interação do usuário corrente com o sistema. Esse módulo é responsável por modificar tanto o conteúdo apresentado ao aluno quanto a forma como será apresentado, ou seja, esse módulo pode escolher entre as diferentes interfaces disponíveis.

Nesse trabalho, estamos interessados apenas na organização do modelo de domínio, ou seja, na base de conhecimento do sistema.

4.1 Base de Conhecimento Teórico

O processo de aprendizado geralmente começa com a apresentação teórica do conteúdo. O conteúdo programatico pode ser dividido em pequenos tópicos, que são apresentados progressivamente ao usuário. Muitos desses tópicos estão relacionados, de forma que alguns deles dependem do bom entendimento de tópicos anteriores para serem assimilados.

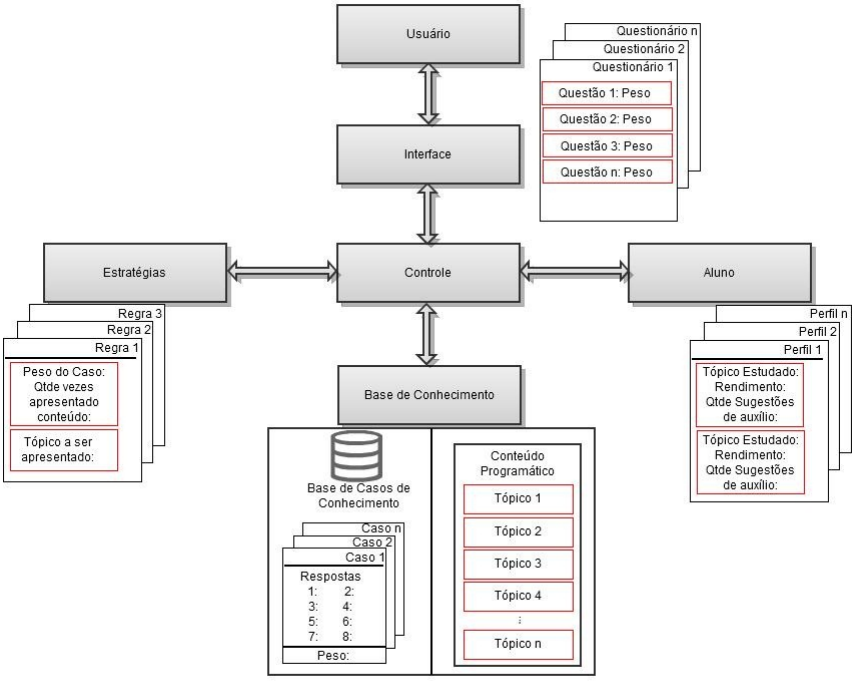


Figura 12: Sistema Tutor Inteligente empregando Raciocínio Baseado em Casos

O bom entendimento de um tópico isolado pode afetar o aprendizado como um todo, visto que a deficiência em um assunto básico pode determinar a falha no entendimento do conteúdo em geral. Por exemplo, crianças do ensino fundamental aprendem sobre operações matemáticas. As operações de soma e subtração são introduzidas antes das operações de multiplicação e divisão. O não entendimento das primeiras operações pode acarretar o fracasso no aprendizado das operações seguintes. Um sistema tutor inteligente pode tratar casos de mal aprendizado de tópicos básicos necessários para o entendimento de tópicos atuais através de técnicas de RBC.

No modelo proposto, todo o conteúdo programático do sistema é armazenado na forma de casos. Os casos representam tópicos e são interligados conforme a dependência de cada um, ou seja, eles podem ser organizados como uma árvore onde os nodos indicam os casos e as arestas mostram a relação e precedência entre eles.

Para mensurar o conhecimento adquirido pelos usuários e permitir que eles passem a outro nível de tutoria, os STI utilizam questionários. As diferentes combinações de respostas de um mesmo questionário são também armazenados na forma de casos, visto que conforme as respostas dadas pelos alunos o sistema pode identificar os tópicos nos quais ele possui dificuldade.

Questão	Resposta	Peso
1	$a \sim e$	$0 \sim 10$
2	f/v	$0 \sim 10$
3	$a \sim e$	$0 \sim 10$
4	f/v	$0 \sim 10$
...	$a \sim e$	$0 \sim 10$
n	$a \sim e$	$0 \sim 10$
Peso Total	$0 \sim 100$	

Tabela 2: Representação de Casos de Questionário

As questões do questionário são objetivas e possuem pesos associados a cada possível resposta, considerando que cada questão é formulada com respostas possíveis e inaceitáveis. A partir da realização do questionário, o sistema é capaz de reconhecer as habilidades e fraquezas do usuário. Através dessa identificação, o sistema pode apresentar novamente um tópico já visto e não entendido, ou então permitir que o aluno continue com o aprendizado.

Para que o sistema consiga interpretar as habilidades e dificuldades dos alunos é importante que os pesos associados às alternativas de cada questão sejam estabelecidos corretamente. Além disso, deve-se associar pesos às questões do questionário, determinando diferentes graus de importância a cada uma delas.

A atribuição de pesos aos questionários pode ser feita de diversas maneiras. Se o questionário todo for da mesma matéria, as questões podem possuir pesos iguais, apenas com diferentes pesos nas alternativas.

Além dos pesos associados às alternativas e questões, cada caso de questionário pode ter determinado um limiar mínimo de acertos do usuário. Se o aluno não alcançar esse limiar mínimo, todo o conteúdo associado ao nível trabalhado deve ser reapresentado a ele, como uma forma de reforçar o aprendizado deficiente.

Uma forma de representação de casos de questionário pode ser observado na Tabela 2. As questões podem ser objetivas, com um dado número de opções de alternativas, ou do tipo falso/verdadeiro. Cada uma delas possui um valor associado, como pode ser observado na última coluna da Tabela 2.

Para a boa compreensão dos casos, será utilizado o exemplo de tutoramento de alunos do ensino médio, no âmbito da matemática. O caso contêm quatro questões sobre operações matemáticas, apenas para demonstrar a representação e a pesificação do caso. A Tabela 3 mostra a representação do caso exemplo.

Questão	Resposta	Peso
1. O resultado da operação $\frac{10}{0}$ é:	$a \sim e$	$0 \sim 10$
2. O produto de 10*5 é o mesmo de 5*10.	f/v	$0 \sim 10$
3. A soma $\frac{a}{b} + \frac{c}{d}$ é calculada como:	$a \sim e$	$0 \sim 10$
4. Toda multiplicação por 0 resulta em 0.	f/v	$0 \sim 10$
Peso Total	$0 \sim 100$	

Tabela 3: Representação do Caso Exemplo de Questionário

Para associar pesos diferentes a cada alternativa das questões, um caso pode ser representado minusciosamente como mostrado na Tabela 4. Cada alternativa pode ser associada a um peso diferente, compondo o peso final da questão. Além disso, pode-se associar cada alternativa errada a um tópico diferente, determinando o conteúdo mal aprendido pelo usuário. A Tabela 5 demonstra a representação completa do caso tomado como exemplo, disponibilizando as alternativas das questões e o peso associado de cada uma delas.

Questão	Altern.	Peso Altern.	Resposta	Tópico	Peso Questão
1	a	$0 \sim 10$	x_1	Tópico x_1	$\sum_{i=1}^{5}(peso\_alternativas) = 10$
	b	$0 \sim 10$		Tópico x_2	
	c	$0 \sim 10$		Tópico x_3	
	d	$0 \sim 10$		Tópico x_4	
	e	$0 \sim 10$		Tópico x_5	
2	f	$0 \sim 10$	x_2	Tópico x_1	$\sum_{i=1}^{2}(peso\_alternativas) = 10$
	v	$0 \sim 10$		Tópico x_2	
3	a	$0 \sim 10$	x_3	Tópico x_1	$\sum_{i=1}^{5}(peso\_alternativas) = 10$
	b	$0 \sim 10$		Tópico x_1	
	c	$0 \sim 10$		Tópico x_1	
	d	$0 \sim 10$		Tópico x_1	
	e	$0 \sim 10$		Tópico x_1	
4	f	$0 \sim 10$	x_4	Tópico x_1	$\sum_{i=1}^{2}(peso\_alternativas) = 10$
	v	$0 \sim 10$		Tópico x_2	
n	$0 \sim 10$				$\sum_{i=1}^{n}(peso\_alternativas) = 10$
Peso Total	$0 \sim 100$				

Tabela 4: Representação de Casos de Questionário Completo

Questão	Altern.	Peso Altern.	Resposta	Tópico	Peso Questão
1. O resultado da operação $\frac{10}{0}$ é:	a. 10 b. 1 c. 0 d. 0.1 e. Não existe.	1 1 2 1 5	x_1	Tópico: Divisão por 0.	10
2. O produto de 10*5 é o mesmo de 5*10.	f v	0 10	x_2	Tópico: Propriedades da Multiplicação.	10
3. A soma $\frac{a}{b} + \frac{c}{d}$ é calculada como:	a. $\frac{a+b}{c+d}$ b. $\frac{a.d+b.c}{b.d}$ c. $\frac{a}{c}$ d. $\frac{b}{d}$ e. $\frac{a.d}{b.c}$	2 5 5 1 1	x_3	Tópico: Soma de Frações.	10
4. Toda multiplicação por 0 resulta em 0	f v	0 0	x_4	Tópico: Termo neutro da multiplicação.	10
Peso Total	40				

Tabela 5: Representação do Caso Exemplo de Questionário Completo

Questão	1	2	3	4	...	n
Resposta	$a \sim e$	f/v	$a \sim e$	f/v	...	$a \sim e$

Tabela 6: Vetor de Representação de Casos de Questionário

Diversas medidas de similaridade podem ser utilizadas na etapa de recuperação de casos. A recuperação permite obter um peso associado ao questionário realizado pelo aluno. Esse peso determina o ponto de onde o usuário presseguirá seu treinamento, se deverá rever conteúdos passados ou poderá continuar com novos conteúdos.

As medidas de similaridade podem considerar os casos como um vetor, contendo as respostas das questões para determinar a semelhança entre eles. A Tabela 6 apresenta o vetor de respostas utilizado na recuperação de casos na base de casos.

Os resultados dos questionários podem ser transformados em novos casos, quando aquela combinação de respostas não existe no banco de casos. Nessa situação a medida de similaridade tenta encontrar o caso que mais se assemelhe a ele e, depois de associar um peso ao resultado, esse novo caso é armazenado, servindo para posteriores consultas.

4.2 Base de Conhecimento Prático

A prática de simulações no sistema pode ser realizado em paralelo ao aprendizado teórico ou após o término de todo o conteúdo disponível para estudo. No entanto, é importante que o aluno realize a tutoria sobre o assunto, conforme a simulação que será feita.

Os casos armazenados na base de conhecimento do sistema representam simulações diversas, contendo variadas condições que o usuário poderá presenciar em um ambiente real. Os dados contido nos casos são bem variados e são totalmente dependentes do domínio da aplicação.

A representação de casos de simulação é uma tarefa árdua pois, dependendo do domínio, pode ser necessário muitos atributos para descrever uma situação. Além disso, os atributos são dos mais diversos tipos, podendo conter números, conjunto de valores, *strings*, valores discretos e tantos outros. Portanto, além da descrição dos casos, são necessárias tabelas auxiliares para a correspondência entre valores assumidos pelos atributos e os valores numéricos.

A descrição de um caso prático deve conter todas as diretrizes para a realização da

Atributo	Valor	Resposta
Atributo a_1	Conjunto Valores $\{v_{11}, v_{12}, v_{13}, ..., v_{1m}\}$	x_1
Atributo a_2	Conjunto Valores $\{v_{21}, v_{22}, v_{23}, ..., v_{2m}\}$	x_2
Atributo a_3	Conjunto Valores $\{v_{31}, v_{32}, v_{33}, ..., v_{3m}\}$	x_3
...	...	...
Atributo a_n	Conjunto Valores $\{v_{n1}, v_{n2}, v_{n3}, ..., v_{nm}\}$	x_n
Peso Total	$0 \sim 100$	

Tabela 7: Representação de Casos Práticos

Atributo	Valor	Resposta
5	Conjunto Valores {operador, operando, divisor, quociente}	x_1
2	Conjunto Valores {operador, operando, divisor, quociente}	x_2
a	Conjunto Valores {produto, quociente, soma, subtração}	x_3
b	Conjunto Valores {resto, diferença, quociente, produto}	x_4
Peso Total	$0 \sim 100$	

Tabela 8: Representação do Caso Exemplo Prático

simulação do ambiente. Um caso pode ser descrito como na Tabela 7, contendo o atributo e também o conjunto de valores possível.

Afim de exemplificar um caso prático, utilizaremos ainda o ambiente de ensino no âmbito da matemática. A simulação consiste em identificar os diferentes elementos da equação $\frac{5}{2} = a, restob$, além de algumas propriedades da equação, e chegar ao resultado final. A Tabela 8 apresenta o caso prático exemplo.

Assim como nos casos teóricos, os casos práticos podem distribuir pesos aos diferentes atributos. Os pesos associados indicam o quão importante é que o usuário considere esse atributo na realização da simulação. Similarmente, cada atributo pode ter associado um tópico do conteúdo teórico, indicando qual assunto deve ser abordado novamente ao usuário para um melhor aprendizado.

A Tabela 9 descreve um caso prático representado com pesos e tópicos ligados aos seus diferentes atributos. É importante notar que, para atributos numéricos, pode-se definir faixas de valores aceitáveis e tópicos interligados a cada uma dessas faixas.

Similarmente, a Tabela 10 representa o caso exemplo prático minusciosamente, com o peso de cada valor possível de atributo, além de apresentar os tópicos que poderiam ser novamente trabalhados com o aluno.

A atribuição de pesos e tópicos aos descritores do caso é feita para que o sistema consiga identificar as deficiências e habilidades dos usuários. Se o usuário não alcançar

Atributo	Valor	Peso	Tópico	Resposta
Atributo a_1	Valores $\{v_{11}, v_{12}, v_{13}, ..., v_{1m}\}$	p_1	Tópicos $\{t_{11}, t_{12}, t_{13}, ..., t_{1m}\}$	x_1
Atributo a_2	Valores $\{v_{21}, v_{22}, v_{23}, ..., v_{2m}\}$	p_2	Tópicos $\{t_{21}, t_{22}, t_{23}, ..., t_{2m}\}$	x_2
Atributo a_3	Valores $\{v_{31}, v_{32}, v_{33}, ..., v_{3m}\}$	p_3	Tópicos $\{t_{31}, t_{32}, t_{33}, ..., t_{3m}\}$	x_3
...	...	...	...	...
Atributo a_n	Valores $\{v_{n1}, v_{n2}, v_{n3}, ..., v_{nm}\}$	p_1	Tópicos $\{t_{n1}, t_{n2}, t_{n3}, ..., t_{nm}\}$	x_n
Peso Total	$0 \sim 100$			

Tabela 9: Representação de Casos Práticos Completo

Atributo	Valor	Peso	Tópico	Resposta
5	operador operando divisor quociente	3 5 1 1	Tópico: Identificação de elementos da divisão.	x_1
2	operador operando divisor quociente	3 5 1 1	Tópico: Identificação de elementos da divisão.	x_2
a	produto quociente soma diferença	1 7 1 1	Tópico: Identificação do quociente.	x_3
b	resto diferença quociente produto	6 1 2 1	Tópico: Identificação do resto.	x_4
Peso Total	40			

Tabela 10: Representação do Caso Exemplo Prático Completo

um limiar inferior definido no sistema em cada etapa de simulação, o sistema pode optar por reapresentar o conteúdo teórico do qual a simulação depende. Após essa nova tutoria, o sistema passa a apresentar simulações básicas e de menor dificuldade, treinando esse mesmo usuário a trabalhar seus pontos fracos. Aqui, vale ressaltar que ambas as bases de conhecimento, tanto teórico quanto prático, podem trabalhar em conjunto, visando a melhor fixação do conteúdo por parte dos usuários.

A etapa de recuperação dos casos tenta encontrar casos similares ao que o usuário acabou de realizar para atribuir um peso à simulação. A partir desse peso, o sistema decide qual será o próximo passo do usuário, se ele continuará com as simulações seguintes ou se ele deverá voltar a estudar um tópico ou realizar outra simulação de mesmo grau de dificuldade para atender às suas deficiências.

Diversas medidas de similaridade podem ser empregadas para essa etapa de recuperação, tanto para definição de similaridades locais quanto globais. As medidas de similaridade utilizadas podem também ser dependentes de domínio e, portanto, o framework traz diferentes padrões de medida.

4.3 Estruturação do Framework

O framework utiliza técnicas de RBC para estruturar a base de conhecimento de um STI totalmente independente de domínio. A finalidade da utilização do framework é conseguir organizar toda a base de conhecimento, tanto dos tópicos teóricos a serem trabalhados com o aluno quanto as simulações que serão posteriormente empregadas para fixar o conhecimento. Além disso, o framework provê diferentes medidas de similaridade, tanto locais quanto globais para a realização das diferentes etapas do ciclo RBC.

É importante nota que, o usuário do framework não é o aluno que utiliza o sistema e sim o desenvolvedor que trabalhará na implementação do aplicativo que emprega o RBC como técnica de inteligência artificial. Os casos experimentados pelo usuário final (aluno), são utilizados para o aprendizado do sistema, visto que os mesmos são anexados à base de conhecimento, podendo ser utilizado para posteriores consultas. A Figura 13 mostra a interação usuário e framework. O usuário (desenvolvedor) é capaz de executar diferentes funcionalidades disponíveis no sistema.

Para a boa definição da base de conhecimento, o desenvolvedor deve estruturar os casos com seus atributos. Por sua vez, os atributos devem ser definidos quanto a tipo e conjunto de valores aceitáveis. É necessário também a definição da medida de similaridade

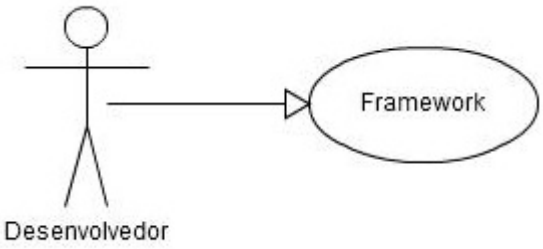


Figura 13: Interação Usuário x Framework

que será utilizada para recuperação de casos na base de dados. A Figura 14 apresenta as diferentes interações disponíveis entre framework e desenvolvedor.

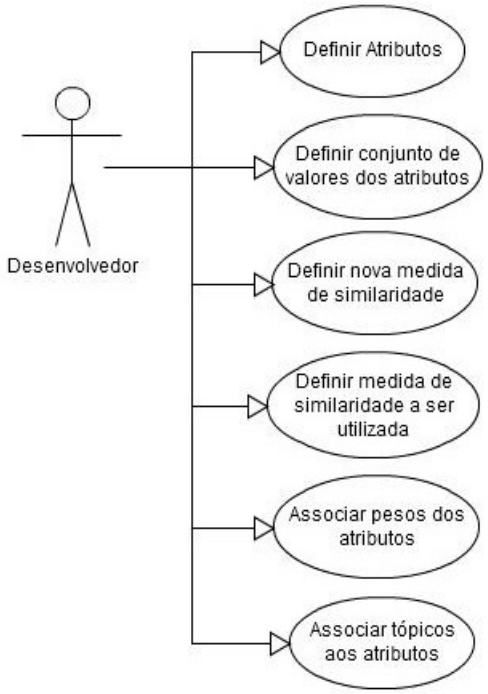


Figura 14: Tarefas executadas pelo desenvolvedor

- **Definição de Atributos:** Todo caso deve conter atributos, que descrevem os dados coletados do usuário ou conteúdos que devem ser apresentados a ele. É através desses atributos que todo o ciclo RBC é realizado. Segundo o exemplo prático dado anteriormente, os atributos consistem nos dados de análise, 5, 2, a e b.

  A definição dos atributos de um caso constitui uma das tarefas mais difíceis de um sistema RBC, pois nem todo dado deve ser considerado e dependendo do domínio da aplicação podem haver muitos atributos.

- **Definição de Conjunto de Valores dos Atributos:** Além da definição dos atributos, o desenvolvedor deve associar um conjunto de valores válidos para cada um deles. Esse conjunto de valores determina quais valores de entrada aquele dado atributo pode assumir. Por exemplo, na definição de casos de questionário, um atributo que descreve uma questão pode receber valores do conjunto a, b, c, d, e ou do conjunto f, v. Qualquer resposta fora do conjunto de valores válidos não deve ser recebido como entrada para aquele atributo.

  Trabalhando ainda com o exemplo prático dado anteriormente, para os atributos 5 e 2 o conjunto de valores possível seria $\{operador, operando, dividor, quociente\}$

- **Definição de Nova Medida de Similaridade:** Existem muitas medidas de similaridade disponíveis na literatura. No entanto, dependendo da aplicação pode ser necessário a definição de uma nova medida de similaridade local ou global. A tarefa de definição de nova medida de similaridade permite que o desenvolvedor descreva a forma como a similaridade entre os casos e/ou atributos será medida.

- **Definição da Medida de Similaridade Utilizada:** Após a definição dos casos com seus atributos e valores válidos, o desenvolvedor deve selecionar a medida de similaridade que será utilizada para recuperar casos semelhantes da base de conhecimento. Essa tarefa também depende do domínio da aplicação, pois cada padrão de medida pode ser apropriado para um tipo específico de sistema inteligente.

- **Associação de Pesos aos Atributos:** Os casos armazenados na base de conhecimento representam experiências já vividas pelos usuários e esses casos são utilizados para posteriores consultas. Cada atributo dos casos pode significar habilidade/deficiência por parte do responsável por vivenciar aquele episódio. Portanto, o desenvolvedor pode associar pesos diferentes aos atributos, determinando o grau de importância de cada um deles, visto que a deficiência em um certo atributo pode acarretar mais dificuldade no prosseguimento do aprendizado do que outros.

- **Associação de Tópicos aos Atributos:** Assim como a associação de pesos, um atributo em particular pode ter associado a ele um tópico da base de conhecimento. A associação de tópicos a atributos determina onde o usuário possui deficiência/habilidade ao errar/acertar aquele dado.

Levando em consideração todas as funcionalidades que devem ser oferecidas ao desenvolvedor, foi montado um diagrama de casos para a construção do framework. A Figura 15 apresenta as classes necessárias e as ligações entre elas.

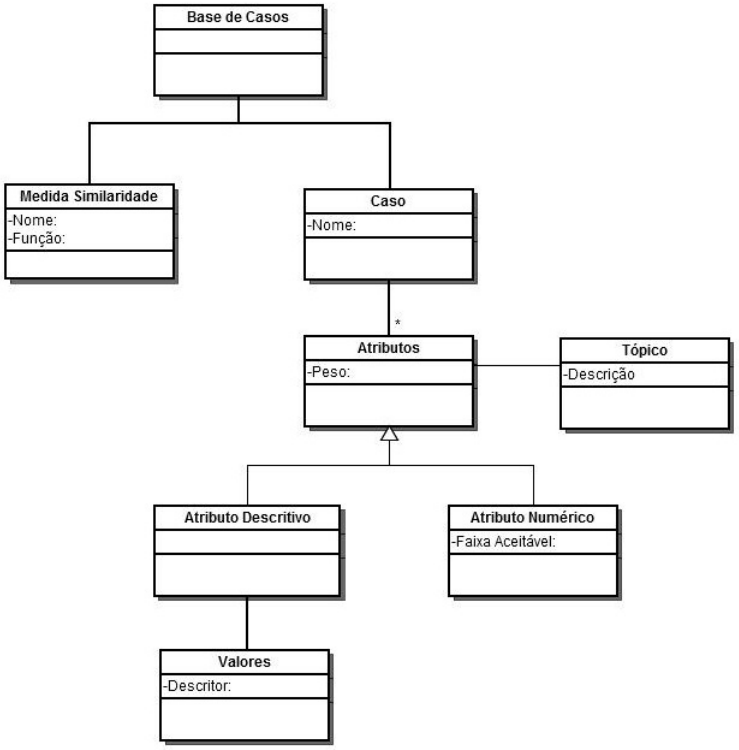


Figura 15: Diagrama de Classes do Framework

A classe ”Base de Casos”armazena todos os casos da base de conhecimento. Ela é composta por objetos das classes ”Medida de Similaridade”e ”Caso”. A classe ”Medida de Similaridade”possui descrições de objetos que representam as funções de medidas de similaridade, ou seja, ela descreve padrões de medidas. A classe ”Caso”armazena os atributos que descrevem um caso. Uma mesma base de casos pode conter diferentes tipos de casos, no entanto as etapas do ciclo RBC devem ser realizados sob casos iguais.

Um caso é constituído de diversos atributos que descrevem os dados coletados do usuário em um dado episódio. Cada atributo possui um determinado peso e pode estar associado a um certo tópico (que também são representados através de casos no sistema), indicando qual conteúdo é/foi necessário para o aprendizado da correta manipulação daquele atributo.

Por fim, as classes ”Atributo Descritivo”, ”Atributo Numérico ”e ”Valores”descrevem os valores que os atributos podem assumir. Através dessas classes, é possível determinar valores descritivos, conjunto de valores válidos, faixas numéricas de valores para cada entrada de atributo.

4.4 Implementação do Framework

A implementação do framework proposto foi realizada em Java, utilizando classes e objetos disponibilizados pela linguagem. A escolha da linguagem utilizada foi feita levando em consideração as facilidades oferecida pela mesma, sua abrangência no mercado e a afinidade do implementador. As diferentes janelas de interação com o usuário foram montadas utilizando ferramentas da própria linguagem, objetivando construir uma ferramenta simples e eficaz.

A tela inicial da ferramenta construída é apresentada na Figura 16. A barra de ferramentas disponibiliza ao usuário quatro opções para interação com o sistema, o menu arquivo, cadastro, visualização e similaridade, mostrados na Figura 17. O menu arquivo possibilita carregar dados já modelados e salvos, salvar novas alterações e cadastros, exportar dados no formato *".txt"* e sair do sistema. Através do menu cadastro, o usuário pode inserir novos modelos ou casos, que serão utilizados para mensurar a similaridade entre quaisquer casos. O menu visualização possibilita que o usuário verifique os modelos e casos já cadastrados no sistema e o menu similaridade apresenta a similaridade entre os casos.

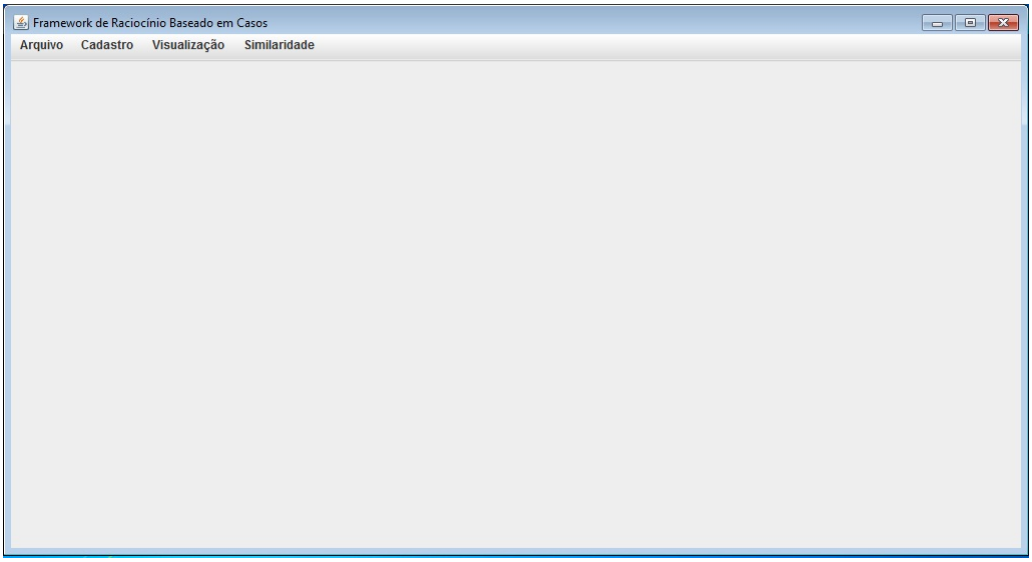


Figura 16: Tela Inicial da Ferramenta

As funcionalidades disponíveis no menu arquivo apresentam mensagens ao usuário, indicando que a opção selecionada foi realizada. A Figura 18 apresenta a mensagem apresentado ao usuário após a seleção da funcionalidade "Carregar" do menu "Arquivo".

No menu cadastro, o usuário pode criar/alterar novos modelos. Ao escolher a opção

Figura 17: Menus Arquivo, Cadastro, Visualização e Similaridade, respectivamente

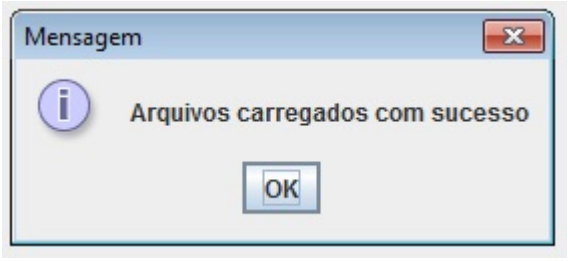


Figura 18: Mensagem do Sistema para a Função Carregar Arquivos

"Cadastro > Novo Modelo", o sistema exibirá a tela mostrada na Figura 19, possibilitando a criação de um novo modelo ou a alteração de um já existente.

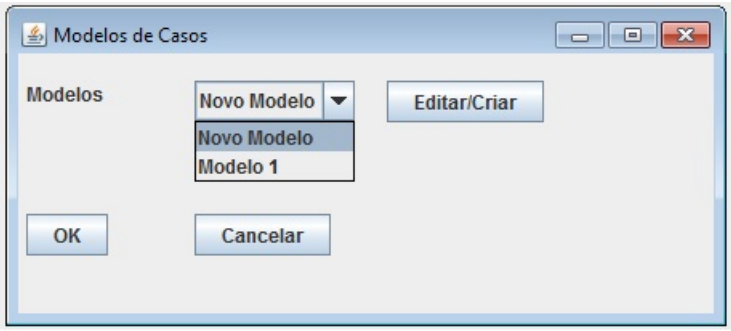


Figura 19: Cadastro/Alteração de Novo Modelo

Clicando no botão "Editar/Criar", o usuário será direcionado a uma nova tela, onde poderá adicionar/mudar o nome do modelo ou adicionar/modificar seus atributos, conforme Figura 20.

Os atributos de um modelo podem ser descritivos ou numéricos. A escolha do tipo de atributo e a nomeação do mesmo é feita no próximo passo, conforme a Figura 21. Nesse passo ainda não temos os valores que o atributo pode assumir.

A definição dos valores aceitáveis de um dado atributo é feita através das janelas mostradas na Figura 22. No caso dos atributos descritivos, o usuário deve cadastrar apenas o símbolo e o peso associado a ele. Para atributos numéricos, o usuário deve

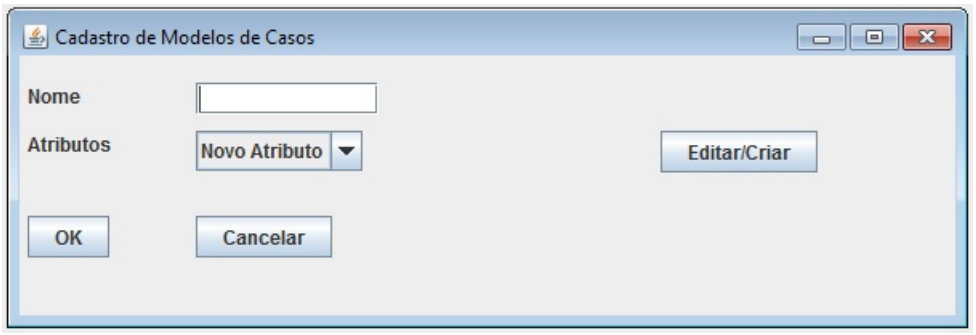


Figura 20: Cadastro do Nome e Atributos de Novo Modelo

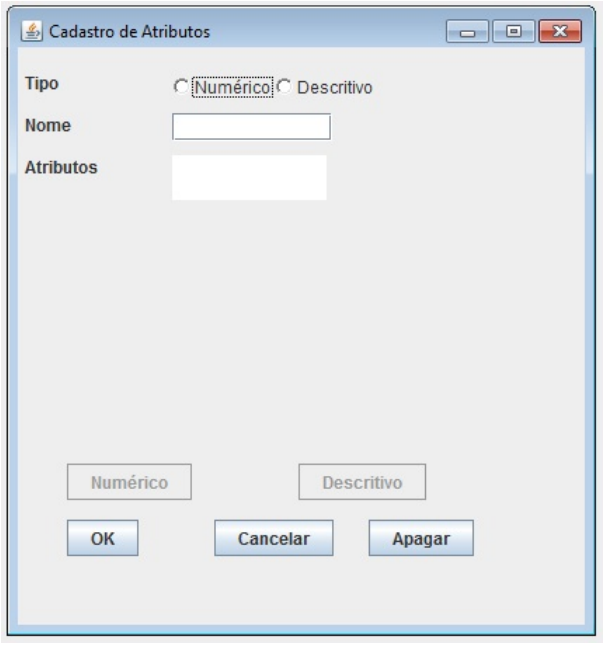


Figura 21: Cadastro do Nome e Tipo do Atributo

definir uma faixa de valores, associando um nome a peso a essa faixa.

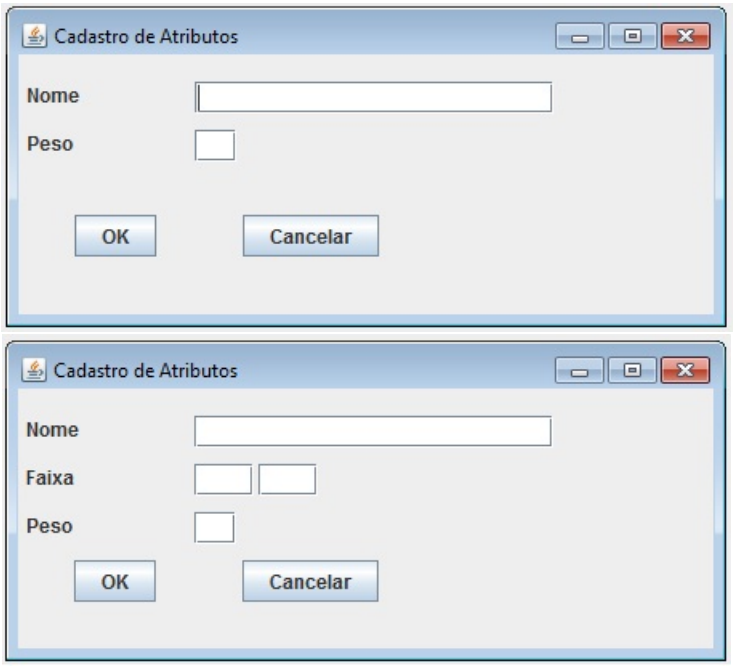


Figura 22: Definição de atributos descritivos e numéricos, respectivamente

Um modelo de caso é constituído por vários atributos e para definí-los, basta que o usuário realize, para cada novo atributo do mesmo modelo, os passos já apresentados. É importante que o cadastro dos modelos seja feito cuidadosamente, visto que a introdução de um novo caso depende de sua definição.

O cadastro de um novo caso é feito em duas etapas. Primeiramente, o usuário deve escolher o tipo de caso que será introduzido, ou seja, o modelo que o caso seguirá, conforme Figura 23.

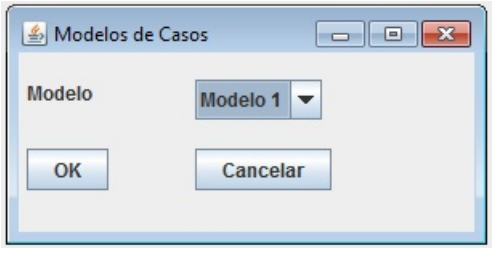


Figura 23: Escolha do Modelo do Novo Caso

Após escolher o modelo do caso, o usuário poderá definir os valores para cada atributo do caso. A tela de cadastro faz um listagem de todos os atributos e, para cada um deles, apresenta os valores que podem assumir. Observe a Figura 24.

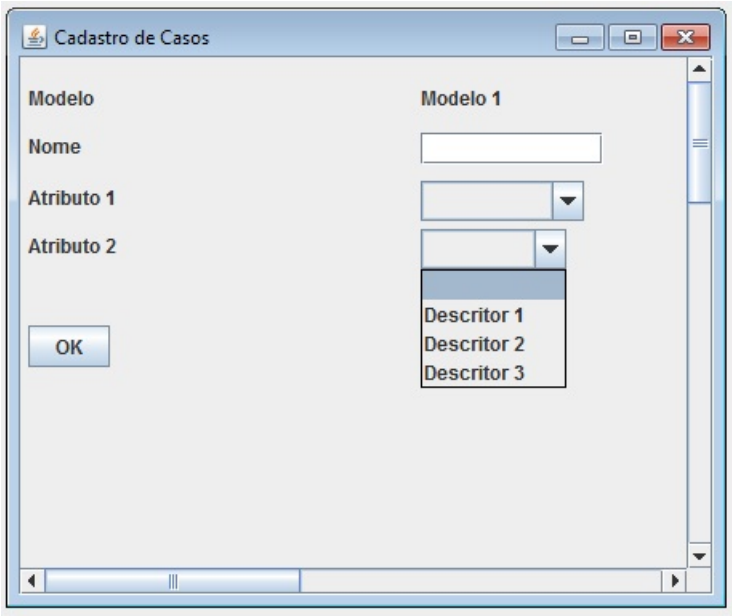


Figura 24: Cadastro dos Atributos de um Novo Caso

O cadastro de novos modelos e novos casos são as tarefas que devem ser executadas pelo usuário. Tendo cadastrado todos os modelos e casos no sistema, o usuário pode visualizar os casos armazenados e também medir a similaridade entre um caso e todos os outros existentes.

O menu “Visualização” possibilita que o usuário verifique tanto modelos quanto casos já cadastrados. Conforme a Figura 25, a visualização de um modelo disponibiliza ao usuário tanto os atributos do modelo quanto os valores que cada um pode assumir, além de apresentar o nome do modelo que está sendo apresentado.

A visualização de um caso cadastrado segue o mesmo princípio que os modelos, no entanto é apresentado apenas o valor que cada atributo assumiu nesse dado episódio, além do nome do caso e modelo segundo o qual ele é definido. A Figura 26 mostra um exemplo de visualição de um caso hipotéticamente cadastrado.

Por fim, o usuário pode analisar a similaridade entre os casos, acessando o menu “Similaridade” e a opção “Definir Similaridade”. Ao escolher essa opção, o usuário deverá selecionar o modelo e o caso com o qual deseja trabalhar, conforme Figura 27.

A similaridade entre todos os casos do mesmo modelo cadastrados no sistema é apresentado ao usuário, inclusive a similaridade entre o caso estudado e ele mesmo. O sistema disponibiliza duas métricas de similaridade, a distância euclidiana e o vizinho-mais-próximo. O valor apresentado na tela é a medida de distância entre os casos analisados,

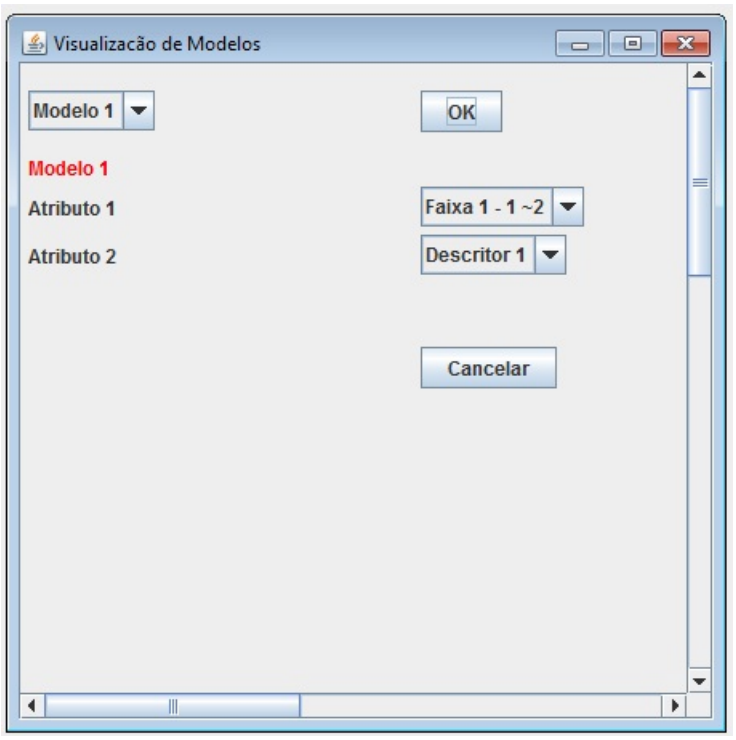


Figura 25: Visualização de um Modelo Cadastrado

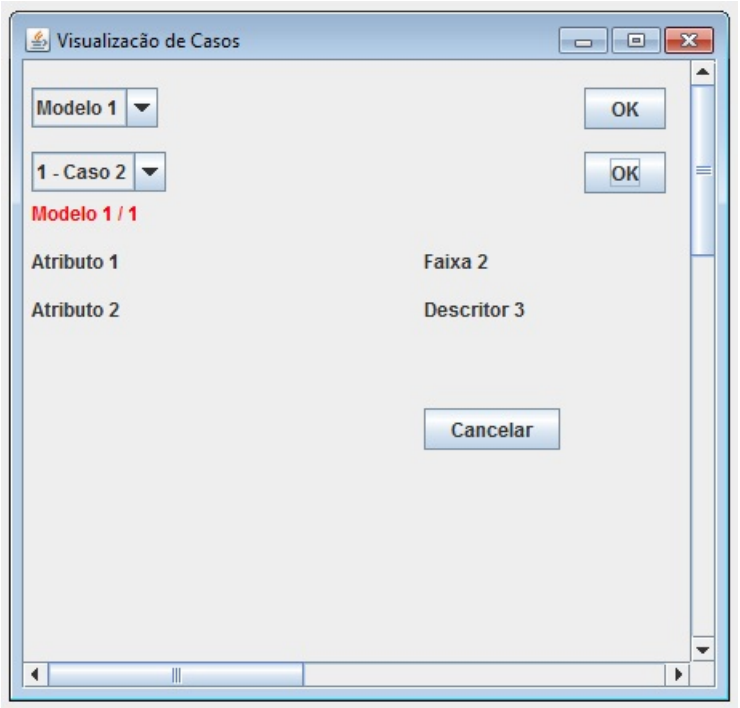


Figura 26: Visualização de um Caso Cadastrado

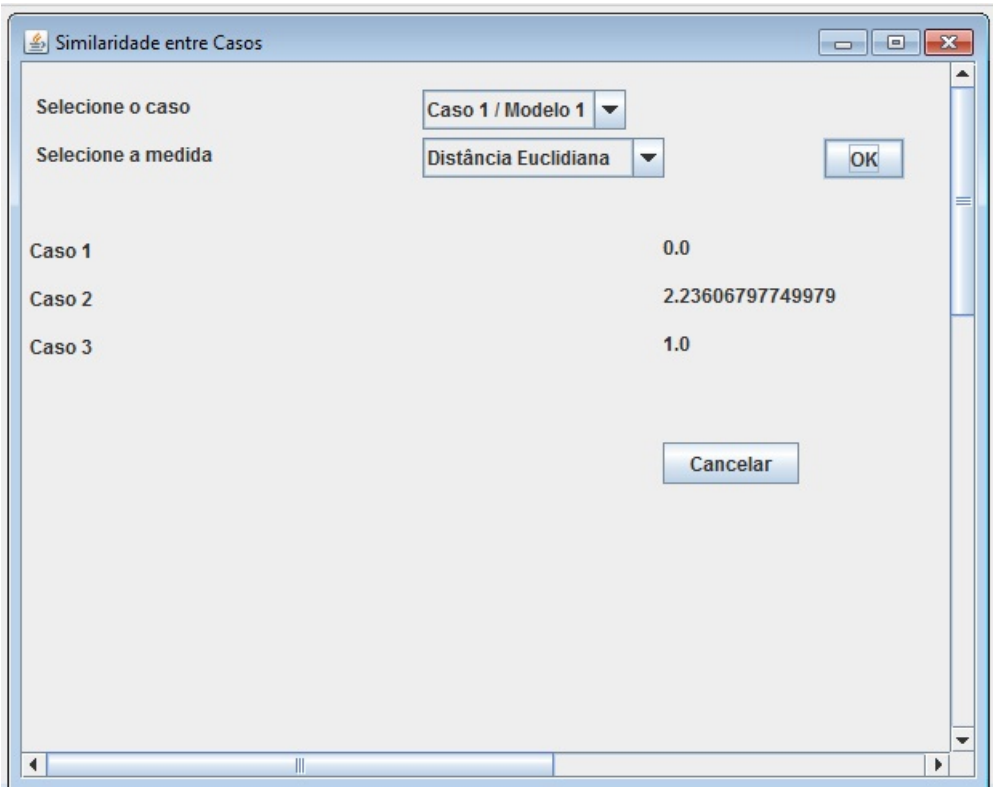


Figura 27: Visualização da Similaridade entre os Casos

ou seja, quanto menor o valor, mais semelhante um caso é de outro.

A utilização do framework para modelar casos visa facilitar o construção de sistemas tutores inteligente que utilizam o raciocínio baseado em casos como meio de adicionar inteligência ao projeto. O arquivo ".txt" gerado pode ser utilizado como entrada no sistema, independente da linguagem e do tipo de sistema em desenvolvimento.

5 Utilização do Framework

O framework proposto visa auxiliar o desenvolvedor na implementação e organização da base de conhecimento. Além disso, o framework deve ser capaz de configurar os casos segundo a necessidade do sistema tutor inteligente que o usuário deseja desenvolver, possibilitando determinar seus atributos e também as métricas de similaridade que serão utilizadas na recuperação dos casos e adaptação dos mesmos.

Para a validação do framework proposto foi utilizado um STI já implementado, visando assegurar suas funcionalidades e eficiência. Como o framework é totalmente independente do domínio da aplicação, tal análise poderia ser feita utilizando qualquer aplicação, visto que a finalidade do sistema é prover uma ferramenta para organizar o conhecimento em casos que assumem diferentes formatos e configurações.

5.1 Descrição do Sistema Tutor Inteligente

O sistema tutor inteligente utilizado para verificação do funcionamento do framework visa auxiliar novos pilotos de helicóptero a desenvolverem habilidades para pilotar essas aeronaves. Tal sistema foi desenvolvido no Laboratório de Multimídia e Interatividade (LMI), anexo ao Instituto de Ciências Exatas (ICE) da Universidade Federal de Itajubá (UNIFEI).

O sistema de treinamento de pilotos de helicóptero conta com duas formas de interação com os usuários, o módulo de treinamento teórico e o módulo de simulação de vôo. O primeiro módulo constitui um sistema de ensino teórico que visa apresentar ao aluno as diferentes partes da aeronave e suas diferentes funções. Esse módulo mostra ao usuário toda a fundamentação teórica de uma aeronave, oferecendo a ele todo o conhecimento necessário para dominar a fundo as particularidades de um helicóptero.

O módulo de ensino teórico utiliza diferentes recursos audio-visuais para aplicar o ensino, conforme apresentado na Figura 28. Cada tela de apresentação de conteúdo é

dotada de texto descritivo da matéria, animações que demonstram o funcionamento da parte analisada e também recursos de voz, que descrevem o que está sendo apresentado ao aluno. Os diferentes recursos de multimídia visam explorar as habilidades potenciais que um aluno em particular possui.

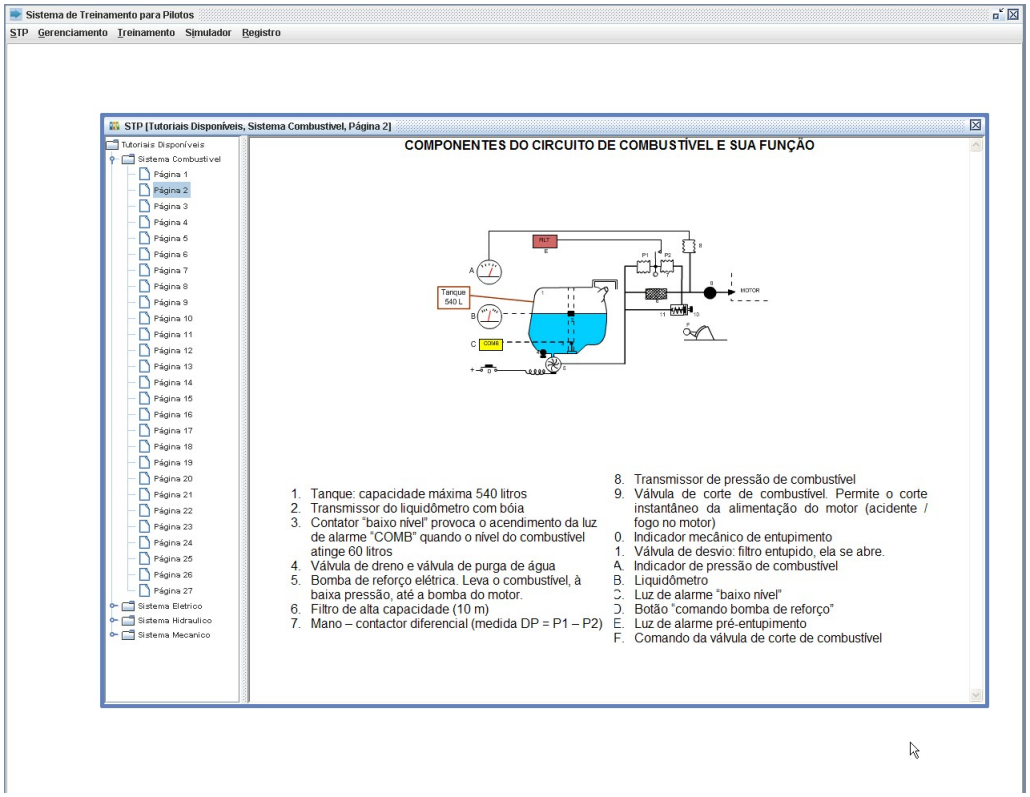


Figura 28: Tela de Apresentação do Conteúdo do STI

Além de apresentar o conteúdo programático, o módulo de ensino téorico também aplica questionários de mensuração de aprendizado do aluno, conforme apresentado na Figura 29. Esses questionários são compostos por questões objetivas de dois tipos, questões de múltipla escolha e questões falso/verdadeiro. Somente depois de ter realizado o questionário do nível estudado, o aluno pode prosseguir para o próximo nível de ensino.

O aluno pode navegar por diferentes tópicos dentro do nível corrente. Além disso, ele pode retornar e rever conceitos já estudados, desde que esses tópicos estejam abaixo do nível o qual ele está estudando no momento.

Todas essas interações entre o aluno e o sistema são armazenadas no sistema na forma de um log para cada usuário em particular. Esse histórico pode ser visto e analisado por tutores humanos, verificando o andamento do ensino dos alunos e também as notas obtidas por eles.

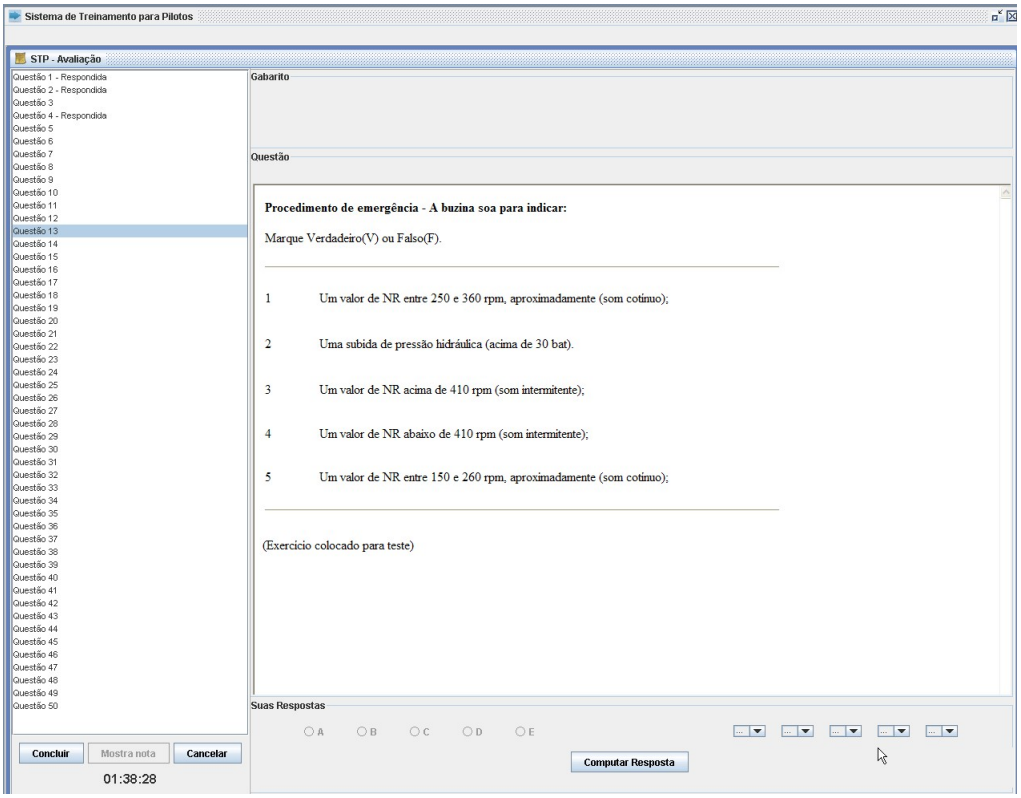


Figura 29: Tela de Aplicação de Questionário do STI

O módulo de simulação de vôo tenta representar uma cabine de helicóptero real para que os alunos se familiarizem com a aeronave. Tal sistema conta com pedais de velocidade, joysticks para representar o manche e uma estrutura em madeira que simula o interior de um helicóptero, com os painéis de controladores e os assentos. A Figura 30 mostra o simulador.

Além da representação do interior do helicóptero, o módulo de simulação de vôo utiliza um *datashow* para apresentar o ambiente ao usuário, afim de que ele visualize as reações do helicóptero advindas de suas manipulações dos controladores. O módulo de simulação utiliza também uma tela em anexo a cabine, utilizada para configurar dados de entrada, como o combustível inicial da aeronave, latitude, longitude, velocidade horizontal e velocidade do ar.

Foi utilizado o software Microsoft Flight Simulator para compor os cenários de vôo e também para controlar a aeronave. O software capta todas as interações do usuário feitas através dos controladores de vôo e através dos dados gerados as telas dos painéis são modificados.

Além disso, o software provê dados do vôo realizado, devolvendo dados como a ve-

Figura 30: Simulador de Vôo do STI

locidade de vôo, altitude, coordenadas geográficas do ponto de partida e chegada, panes ocorridas e tantos outros dados que podem ser utilizados na modelagem dos dados.

Ambos os módulos podem trabalhar em conjunto. O módulo teórico determina as habilidades já adquiridas pelos usuários, afim de prever quais as simulações que podem ser realizados no módulo de simulação. O módulo de simulação pode determinar a necessidade do aluno voltar a rever alguns tópicos já vistos durante seu treinamento teórico, melhorando suas deficiências.

5.2 Dados Utilizados na Validação

A validação do framework necessitou de dados para a modelagem e inclusão de casos na base de conhecimento. O sistema descrito na seção anterior possibilitou sua verificação através da modelagem de casos teórico e práticos. Nos casos teóricos, foram utilizadas as questões dos questionários disponíveis para aplicação nos usuários. Para a modelagem dos casos práticos de simulação foram considerados os relatórios gerados pelo software Microsoft flight simulator juntamente com algumas reportagens reais de panes em helicópteros, obtidos através do *site* da (CPU-RS, 2010), coletadas da SIPAER - Sistema de Prevenção e de Investigação de Acidentes Aeronáuticos.

A modelagem dos atributos pertinentes a um caso é dependente do domínio da aplicação que se deseja construir. A coleta dos mesmos é feita considerando os tópicos que se deseja trabalhar com o aluno. Portanto, a descrição de um mesmo episódio pode dar ori-

gem a diferentes modelagens de casos, visto que os casos devem ser enquadrados conforme a aplicabilidade do sistema.

A Figura 31 apresenta uma dessas reportagens de falha utilizada, que pode ser vista no Anexo A na página 83. Esses relatórios possuem descrição dados técnicas de diferentes panes ocorridas em acidentes envolvendo helicópteros. Os documentos trazem o histórico do voo, análise do erro ocorrido com uma descrição do que poderia ser realizado para evitar o acidente e também uma conclusão sobre os dados analisados.

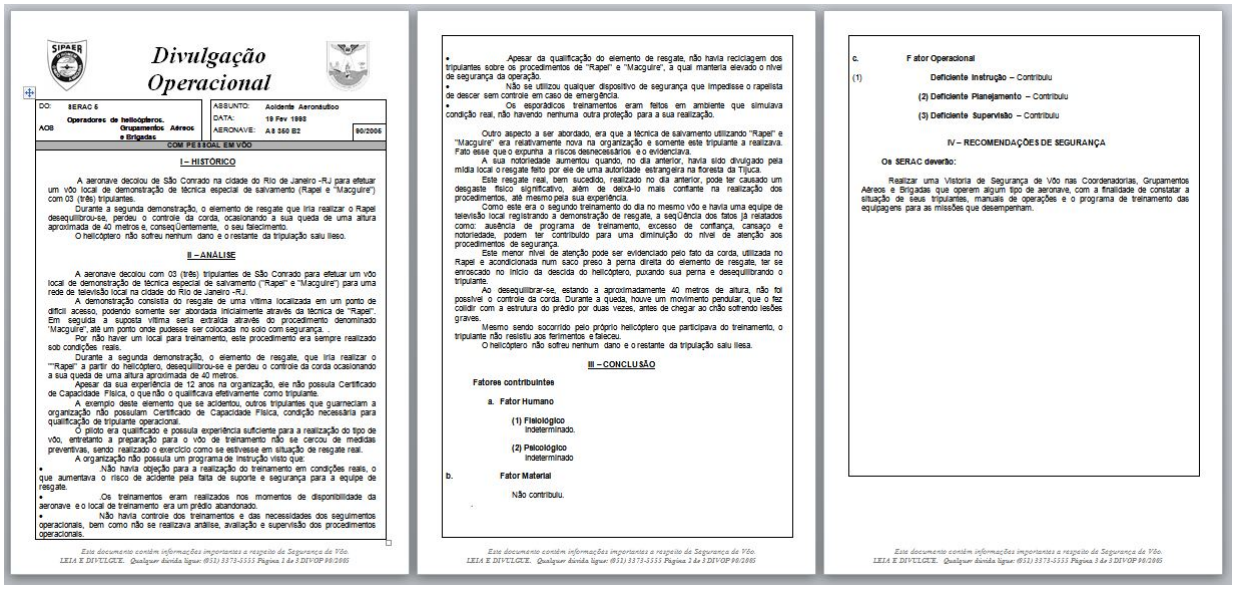


Figura 31: Exemplo de uma Reportagem de Falha

Tendo em mãos os dados do sistema em questão, pôde-se coletar atributos para modelar casos para representar o conhecimento da base de casos. A Figura 32 apresenta a modelagem de casos para questões teóricas do STI. Como as questões podem ser de múltipla escolha ou falso/verdadeiro, o caso é modelado de forma que as possibilidades de respostas estão no conjunto $\{a, b, c, d, e\}$ ou $\{f, v\}$. Cada uma das alternativas possui um peso associado. Tal valor é utilizado no cálculo da similaridade entre os casos existentes.

Para a modelagem dos casos práticos, foram utilizados cinco reportagens de falha. O Apêndice A na página 83 apresenta esses episódios observados. Através da análise dos mesmos, coletou-se alguns atributos e faixas de valores associados a cada um. A Figura 33 apresenta parte desses atributos modelados utilizando o framework proposto.

Após a modelagem dos casos teóricos e práticos, foram inseridos alguns casos exemplos para que fosse possível encontrar a similaridade entre eles. Para os casos teóricos, as respostas para as questões definidas foram definidas aleatoriamente. A inserção dos casos

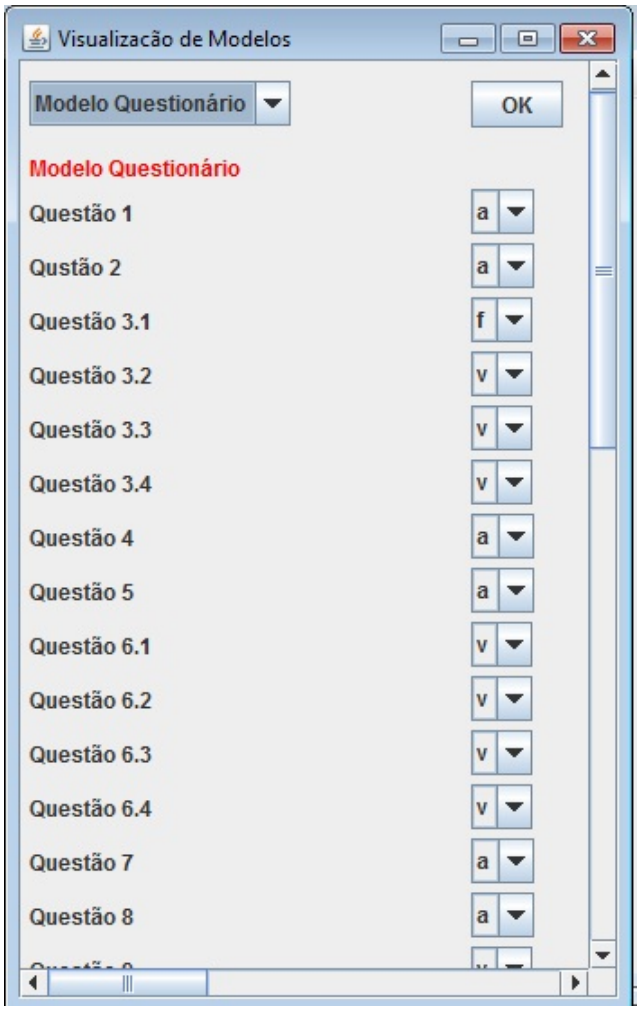


Figura 32: Modelagem dos Casos Teóricos

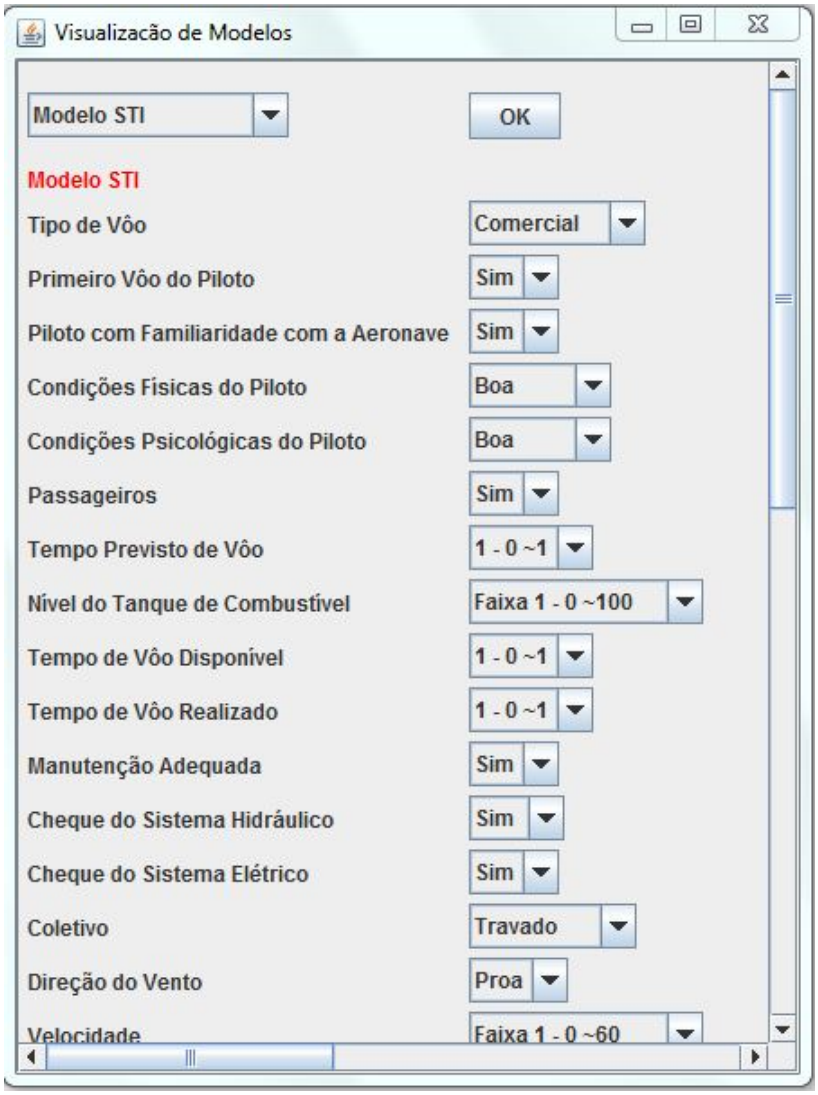


Figura 33: Modelagem dos Casos Práticos

práticos foi feita com base nas reportagens selecionadas. As Figuras 34 e 35 apresentam um caso teórico e prático, respectivamente, a nível de ilustração.

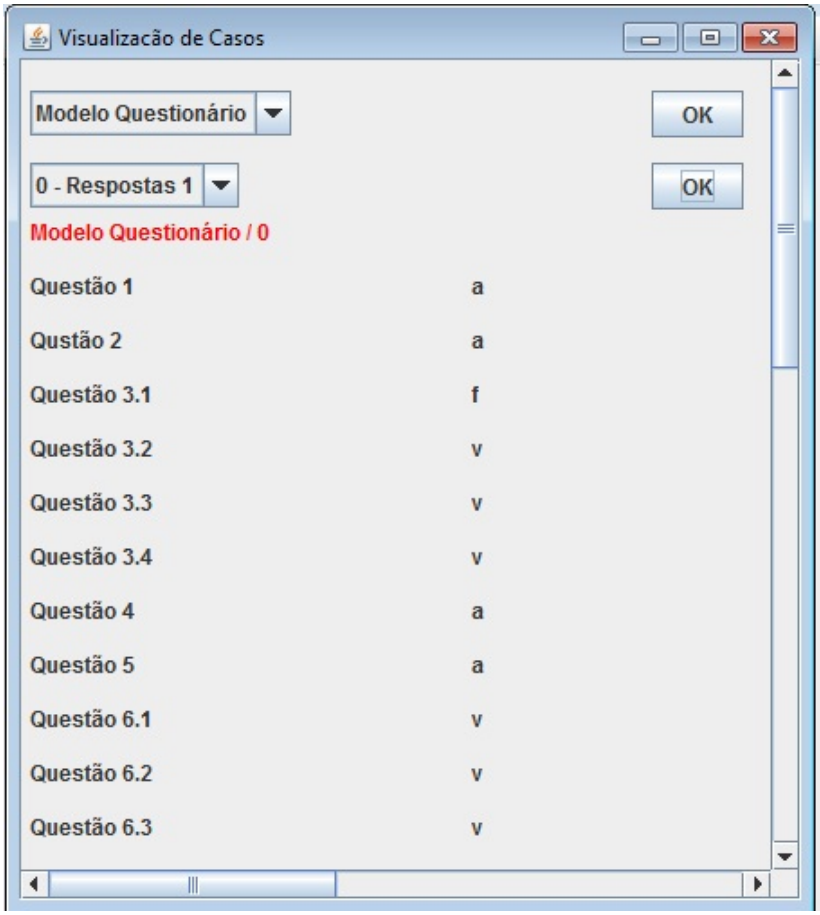


Figura 34: Exemplo de um Caso Teórico

Após modelar e inserir novos casos no sistema, foi possível verificar a similaridade entre um caso selecionado e todos os outros existentes. Para os casos teóricos foi selecionado o caso com título "Respostas 2" como exemplo - qualquer um dos casos poderia ser utilizado - para verificarmos a similaridade utilizando a métrica de distância euclidiana. A Figura 36 apresenta os valores retornados pelo sistema, indicando que o caso com título "Respostas 3" é o mais similar a ele, pois possui menor distância.

Em um sistema RBC e considerando apenas os casos existentes, o caso "Respostas 3" poderia ser utilizado na recuperação, pois é o que possui menor diferença em relação ao "Respostas 2". Como estamos tratando de questionários, poderíamos considerar que os alunos que realizaram os questionários "Respostas 2" e "Respostas 3" possuem nível de conhecimento parecido.

Similarmente, podemos medir a similaridade entre dois casos práticos. Para exempli-

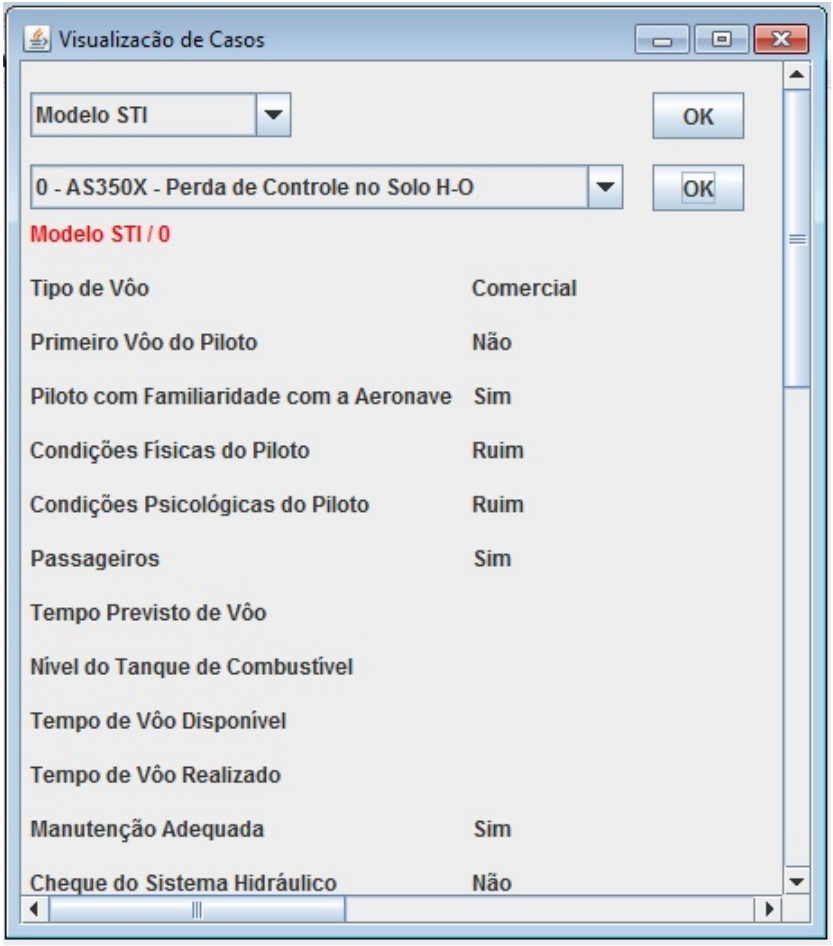


Figura 35: Exemplo de um Caso Prático

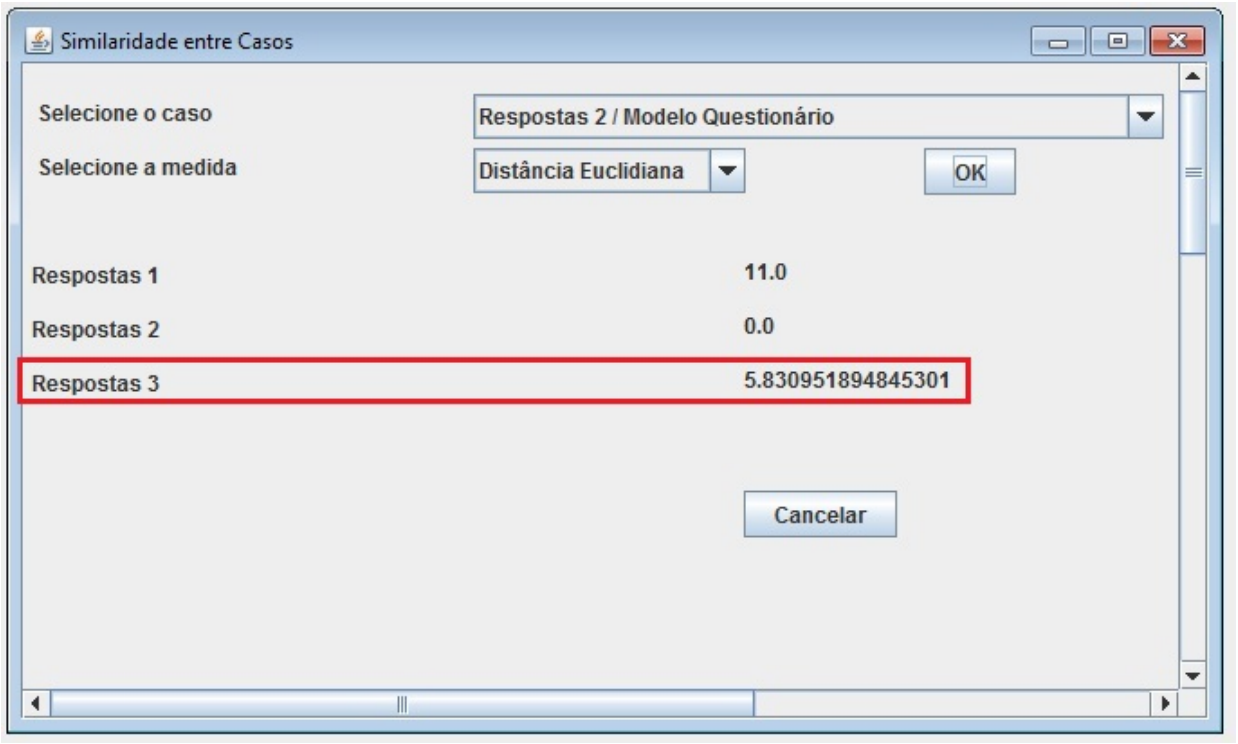


Figura 36: Similaridade entre dois casos teóricos

ficar a verificação de similaridade utilizamos o caso prático com título "BELL 1 - Colisão no Solo com Obstáculo H-O". Como resultado, Figura 37, temos que o caso com título "BELL 3 - Perda de Controle em Voo H-O" é o mais similar a ele. Como estamos tratando de casos práticos, em um sistema RBC o caso "BELL 3 - Perda de Controle em Voo H-O" poderia ser utilizado como um meio adicional de fixação de conhecimento adquirido também através do caso "BELL 1 - Colisão no Solo com Obstáculo H-O".

Portanto, o framework proposto pode ser utilizado para organizar diferentes tipos de conhecimento de um sistema tutor inteligente, independente do domínio da aplicação e do número e forma dos atributos utilizados em sua modelagem.

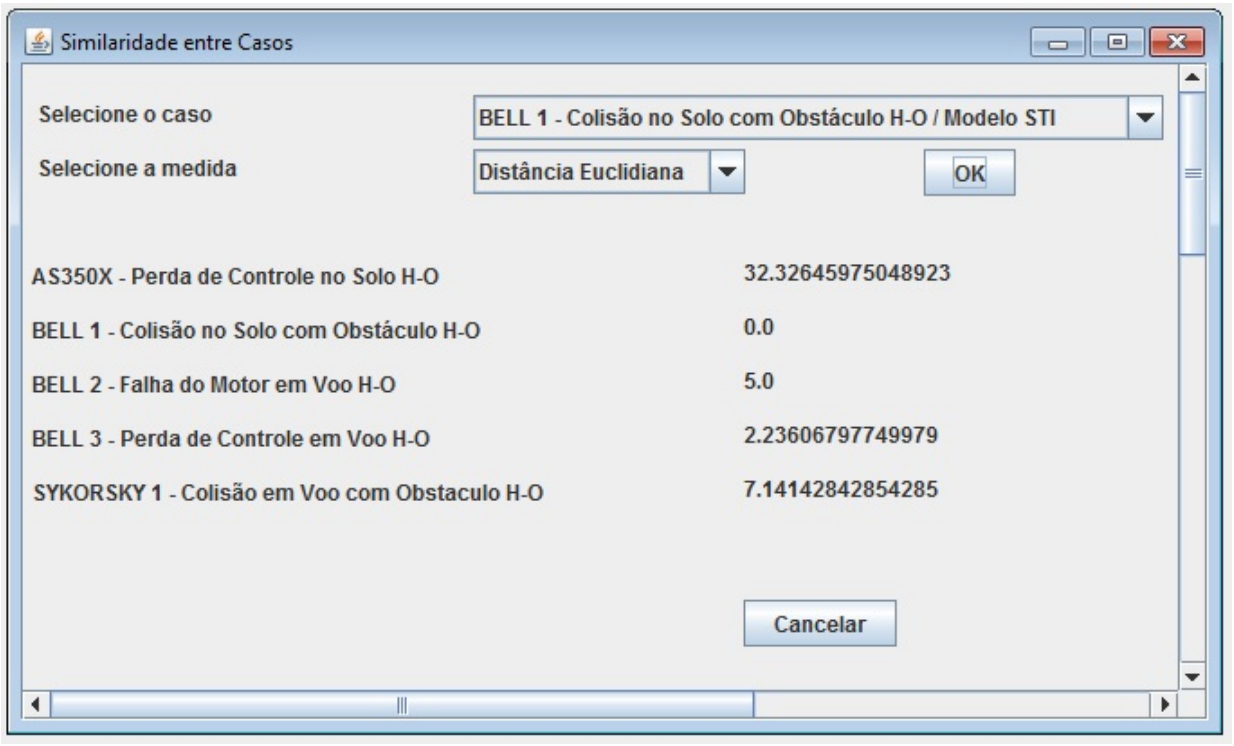


Figura 37: Similaridade entre dois casos práticos

6 Conclusão

A necessidade de ensino personalizado motivou a criação de sistemas tutores computadorizados. Os primeiros sistemas eram baseados em regras e todo o sistema seguia uma ordem pré-estabelecidas de passos e não se adaptavam às características do usuário. Atualmente, existem diversas linhas de pesquisas em inteligência artificial que buscam desenvolver STI capazes de se adaptar segundo a necessidade de cada aluno. Esses sistemas podem incorporar inteligência através de variadas técnicas de IA, como redes bayesianas, redes neurais, lógica fuzzy, entre outras.

O raciocínio baseado em caso é uma técnica de IA relativamente nova. Essa abordagem tenta simular o funcionamento do cérebro humano, armazenando episódios vivênciados no passado para solucionar novos casos. Esse método de inteligência trabalha em um ciclo de quatro etapas, recuperação, reúso, revisão e retenção. Existem medidas de similaridade que são utilizadas na etapa de recuperaçao para medir a semelhança entre dois casos distintos e, posteriormente, utilizar o caso existente para encontrar a solução desejada. É importante que os casos sejam modelados de forma consistentes para que o sistema consiga executar todas as etapas do ciclo. Sistemas RBC estão em constante modificação, sempre incorporando novos casos e, portanto, mais conhecimento na forma de casos que podem ser utilizados posteriormente para pesquisa.

O framework desenvolvido utiliza essa abordagem de inteligência artificial para estruturar o conhecimento de diferentes domínios. Tal ferramenta visa auxiliar desenvolvedores de sistemas a organizarem e definirem atributos de casos e também valores com os quais cada atributo pode trabalhar.

Como a ferramenta é totalmente independente de domínio, ela pode ser utilizada na elaboração de sistemas inteligentes que utilizam a mesma técnica de IA. A vantagem encontrada na utilização de tal ferramenta é a simplificação da organização da base de conhecimento e também na mensuração da similaridade entre os casos existentes no sistema. Além disso, como o sistema exporta os dados na forma de um arquivo ".txt", os dados

podem ser integrados a sistema feitos em qualquer tipo de linguagem de programação.

A utilização do framework possibilita que o desenvolvedor não se preocupe com a aplicação da técnica de IA, pois toda a estruturação está implementada na ferramenta. O restante do sistema pode ser desenvolvido independentemente do framework, sendo que o mesmo pode ser utilizado em três das quatro etapas do ciclo RBC, nas etapas de retenção, recuperação e reúso dos casos.

6.1 Trabalhos Futuros

Os diferentes sistemas que trabalham com inteligência artificial são dependentes do domínio no qual estão inseridos. Uma das maiores dificuldades em desenvolver um framework de uso geral é modelar os dados de forma que qualquer tipo de aplicação possa utilizá-los. A ferramenta desenvolvida pode ser utilizada independente do sistema, modelando diferentes tipos de dados e suas faixas de valores.

No entanto, no desenvolvimento do framework pudemos identificar outras funcionalidade que poderiam ser inseridas e/ou melhoradas na versão corrente, que incluem:

- Criação de novas medidas de similaridade: Como as medidas de similaridade são importantes para o bom funcionamento de um sistema RBC e por serem dependentes do domínio, o framework poderia oferecer ao desenvolvedor uma interface para definição de novas medidas de similaridade para serem utilizadas segundo o domínio específico;

- Definição de medidas de similaridade local: As medidas de similaridade implementadas no framework consideram apenas cálculos globais dos pesos dos atributos. Métricas de similaridade local podem ser utilizadas para melhorar a recuperação de casos semelhantes;

- Inclusão de peso aos casos: Além da descrição dos atributos pertencentes aos casos, poderia ser incorporado pesos aos mesmos, de forma que possa ser utilizado para mensurar o grau de conhecimento do usuário que o realizou;

- Estruturação do conteúdo programático: A organização da base de conhecimento pelo framework se prendeu apenas em questões práticas e questionários, ou seja, todo o conteúdo programático de ensino do STI não foi considerado. Seria interessante otimizar o framework tentando oferecer ao desenvolvedor uma forma de organizar esse conhecimento também, facilitando a estruturação de todo a aplicação;

- Associação de tópicos aos casos: Um caso representa um episódio já ocorrido. Através desse episódio é possível identificar habilidades e deficiências no usuário que o experimentou. Através dessa identificação, os casos poderiam ser associados a tópicos que poderiam ser sugeridos para leitura e estudo por parte do aluno.

Referências

AAMODT, A.; PLAZA, E. Case-basead reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications. IOS Press, V.7:1 pp.39-59*, 1994.

AKHRAS, F.; SELF, J. Beyond intelligent tutoring systems: situations, interactions, processes and affordances. *Instructional Science, Kluwer Academic Publishers, Netherlands*, 2002.

BROWN, J. S.; BURTON, R. R. Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science, V.2:2 pp.71-192*, 1978.

COSTA, R.; WERNECK, V. Sistemas tutoriais: Aplicação das tecnologias de hipermídia e de inteligência artificial em educação. *Relatório Técnico do Programa de Engenharia de Sistemas e Computação, ES- 427/97 - COPPE - UFRJ*, 1996.

CPU-RS, F. de C. A. *Relatório de Falhas de Helicópteros*. 2010. Página na Internet - http://www3.pucrs.br/portal/page/portal/facauni/facauniCapa/facauniDepVoo.

DELPIZZO, V. L. F. *Prescrição de atividades físicas através do uso da inteligência artificial*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, 1997.

FERNANDES, A. M. R. *Inteligência Artificial: noções gerais*. [S.l.]: Visual Books, 2003.

FOWLER, D. G. A model for designing intelligent tutoring systems. *Journal of Medical Systems, V.15:1*, 1991.

FREEMAN, R. What is an intelligent tutoring system? *Published in Intelligenge, V.11:3 pp.15-16*, 2000.

GAMBOA, H.; ANA, F. Designing intelligent tutoring systems: a bayesian approach. *3rd International Conference on Enterprise Information Systems, ICEIS*, 2001.

GAVIDIA, J. J. Z.; ANDRADE, L. C. V. *Sistemas Tutores Inteligentes*. 2003.

GICK, M. L.; HOLYOAK, K. J. Analogical problem solving. *Cognitive Psychology, V.12 pp.306-355*, 1980.

GIRAFFA, L. M. M. *Uma Arquitetura de Tutor Utilizando Estados Mentais*. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, 1999.

JAGANNATHAN, R. et al. *A Fuzzy Non-linear Similarity Measure for Case-Based Reasoning Systems for Radiotherapy Treatment Planning*. 2010.

JONASSEN, D. H.; WANG, S. The physics tutor: Integrating hypertext and expert systems. *Journal of Educational Technology Systems, V.22:1 pp.19-28*, 1993.

JUNIOR, D. T. et al. Sistema de raciocínio baseado em casos para recomendação de programa alimentar. *Revista Eletrônica de Sistemas de Informação, V.9:3*, 2006.

KOLODNER, J. L. An introduction to case-based reasoning. *Artificial Intelligence Review, V.6:3 pp.3-34*, 1992.

MCTAGGART, J. Intelligent tutoring system and education for the future. *CI 512X Literature Review*, 2001.

RAABE, A. L. A. *Uma proposta de arquitetura de sistema tutor inteligente baseada na teoria das experiências de aprendizagem mediadas*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 2005.

RIORDAN, D.; HANSEN, B. K. A fuzzy case-based system for weather prediction. *Engineering Intelligent Systems, V.3 pp.139-146*, 2002.

SCHANK, R. C. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. [S.l.]: Cambridge University Press, 1982.

SELF, J. Bypassing the intractable problem of student modeling. *In C. Frasson and G. Gauthier (eds) Intelligent Tutoring systems: at the crossroads of artificial Intelligence and education, Norwood*, 1990.

URRETAVIZCAYA, L. M. Sistemas inteligentes em el âmbito de la educación. *Revista Iberoamericana de Inteligência Artificial, V.12 pp.5-12*, 2001.

WANGENHEIM, C. G.; WANGENHEIM, A. V. *Raciocínio Baseado em Casos*. 1. ed. [S.l.]: Manole, 2003.

WENGER, E. *Artificial Intelligence and Tutoring Systems: Computational andCognitive Approaches to the Communications of Knowledge*. [S.l.]: Los Altos, 1987.

APÊNDICE A – Reportagens de Falha

Divulgação Operacional

DO:	**SERAC 5**	ASSUNTO:	**Acidente Aeronáutico**	
AOS	**Operadores da Aeronave e Empresas de Táxi Aéreo**	DATA:	**24 Nov 2000**	
		AERONAVE:	**AS-350B3**	**101/2005**

PERDA DE CONTROLE NO SOLO

I – HISTÓRICO

A aeronave realizaria um vôo local com um piloto e cinco passageiros. Seria um vôo de cortesia para apresentação do helicóptero a um grupo de executivos da empresa que o estava adquirindo.

Após o acionamento do motor, durante o cheque do sistema hidráulico, a aeronave decolou sem controle, girou e colidiu com o solo.

O helicóptero sofreu danos graves. O piloto e os passageiros saíram ilesos.

II – ANÁLISE

O helicóptero iria realizar um vôo local com um piloto e cinco passageiros. Tinha como objetivo a apresentação da aeronave para um grupo de executivos.

Durante o cheque do sistema hidráulico, enquanto o piloto movimentava o cíclico o sentido longitudinal, o helicóptero saiu do solo, evidenciando uma movimentação do comando coletivo, ligado mecanicamente à mudança de passo das pás do rotor principal.

Conforme a lista de verificações da aeronave, o coletivo deveria estar travado durante este procedimento.

Surgem duas hipóteses para o comando estar fora da posição: o destravamento inadvertido por ter apoiado a mão, e o esquecimento de checar a posição durante a execução dos procedimentos. Os dois fatos podem ter ocorrido simultaneamente.

O piloto declarou que estava com dificuldade em manter a concentração no vôo, dividindo sua concentração no vôo anterior-e no seguinte. Esta perda ou diminuição da consciência situacional pode ter sido causada por fatores intrínsecos ao piloto, como sua motivação e ansiedade combinadas a uma carga de trabalho elevada e continuada. Pode ter sido criado um quadro de fadiga crônica, construída ao longo de um período e não somente naquele dia.

O conceito de elevada carga de trabalho e as diferentes respostas de pilotos a estas situações são questões subjetivas, cabendo às instituições e aos pilotos identificar e prevenir condições inseguras. É certo, porém, que há uma grande dificuldade em qualquer pessoa reconhecer que está no seu limite.

Em aviação, no entanto, deve haver uma conscientização e estímulo para que estes limites ou sintomas de dificuldades de concentração sejam percebidos e classificados como fatores de risco.

O piloto afirmou ter deixado de observar alguns itens da lista de verificações, como resultado de uma autoconfiança excessiva e do julgamento de possuir elevado conhecimento. 0 manual do helicóptero previa o cheque da trava do coletivo.

Seu conhecimento e experiência favoreceram o comportamento auto-suficiente e confiante na prática cotidiana.

A supervisão da empresa não percebeu que um piloto que voa modelos diferentes de aeronaves em configurações diversas, pode desenvolver o hábito de não seguir a risca cada uma das listas de verificações.

Quanto ao deslocamento do coletivo para cima há duas hipóteses:

- A primeira seria de o piloto ter esbarrado na trava, soltando-a. Foi descrito que, durante o cheque do sistema, ele estaria executando outra tarefa no painel da aeronave o ajuste do VEMD. Tal fato poderia ter sido causado por contato das mangas do uniforme de vôo ou luvas, se estivesse utilizando-as. Alguns pilotos costumam utilizar as luvas dobradas por causa do calor, assim como as mangas dos macacões quando compridas. Nesta hipótese, possivelmente o piloto teria percebido e associado o movimento do braço e o contato com a saliência de encaixe na trava com o movimento do helicóptero.
- .A Segunda hipótese seria da trava já estar, solta antes da realização do cheque hidráulico. Nos testes realizados pela empresa na aeronave, apesar da falta dos componentes dinâmicos e da movimentação destes, o sistema hidráulico foi acoplado e a trava não se desprendeu quando a mão era apoiada no coletivo. Com o coletivo solto, foi feito o teste do sistema hidráulico. Foi observada uma subida do coletivo ao se esgotar a pressão hidráulica com os movimentos longitudinais de cíclico. Esta tendência é deveras conhecida entre os operadores.

Baseando-se nestes testes, esta hipótese aparece como a mais provável de provocar a decolagem inadvertida do helicóptero.

Uma vez fora do solo, com o sistema hidráulico desligado, a aeronave Esquilo apresenta condições de controlabilidade degradadas, exigindo forças maiores nos comandos. No treinamento de aproximações e pairados já são consideráveis os esforços.Ao assumir os comandos a partir de uma condição instável, com giro e deslocamentos, torna-se ainda mais difícil a situação para o piloto, exigindo mais espaço físico e tempo para a sua estabilização. :,

Enquanto o helicóptero girava, o piloto percebeu a proximidade de outra aeronave o solo. O comando executado pelo piloto para livrá-la pode ter aumentado ainda mais a instabilidade da aeronave, a qual terminou por inclinar-se à direita e colidir o rotor principal com o solo.

Os serviços de manutenção foram considerados periódicos e adequados.

Não houve evidência de contribuição do fator material para o acidente.

De acordo com os dados, estiveram presentes no acidente o aspecto operacional, pela não observância da posição da trava do comando coletivo, originada em comportamento (autoconfiança e auto-suficiência), hábitos (não utilização da lista de verificações), dificuldade de concentração e elevada carga de trabalho continuada.

III – CONCLUSÃO

Fatores contribuintes

a. Fator Humano

(1) Fisiológico
Contribuiu

(2) Psicológico
Contribuiu

b. Fator Material
Não Contribuiu

.

c. F ator Operacional

(1) Deficiente Supervisão – Contribuiu

(2) Deficiente Aplicação dos Comandos – Contribuiu

(3) Deficiente Julgamento – Contribuiu

IV – RECOMENDAÇÕES DE SEGURANÇA

Os SERAC deverão:

Divulgar este acidente para os operadores de AS 350 em suas áreas de jurisdição.

Divulgação Operacional

DO:	**SERAC 5**	ASSUNTO:	**Acidente Aeronáutico**
AOS	**Aeroclubes e Escolas de Aviação**	DATA:	**08 Ago 1998**
		AERONAVE:	**BELL-47 G2**

76/2005

COLISÃO NO SOLO COM OBSTÁCULO

I – HISTÓRICO

A aeronave do Aeroclube realizava um vôo local de instrução para formação de instrutor de helicóptero.

Durante um treinamento de auto-rotação, na vertical do campo, houve colisão da cauda com o solo e, como conseqüência, perda de controle.

A aeronave sofreu danos graves, o instrutor sofreu lesões leves e o piloto saiu ileso.

II – ANÁLISE

O instrutor e o piloto eram qualificados e possuíam experiência para o tipo de vôo.

Os serviços de manutenção foram considerados periódicos, porém inadequados, devido ao fato de terem sido realizados pelo aeroclube, que não era homologado para tal.

Não houve indícios de falha material ou mau funcionamento da aeronave.

O relacionamento de amizade e confiança entre os pilotos contribuiu para que houvesse um relaxamento dos níveis de ansiedade e do julgamento dos riscos do treinamento.

O bom desempenho do aluno nos exercícios anteriores contribuiu para uma redução do nível de atenção do instrutor.

Em entrevista ambos afirmaram considerar a formação de instrutor "uma atividade simples por se tratar de aluno com experiência na aeronave, que necessita apenas ser adaptado à mudança de posição na aeronave".

As declarações dos tripulantes referentes ao nível de atenção e grau de risco do vôo em si, denotam excesso de autoconfiança e a complacência, que foram determinantes para a ocorrência do acidente. Estes fatores vêm contribuindo para vários acidentes na aviação civil e militar ao longo dos anos e vêm sendo objetos permanentes de palestras, aulas, DIVOP e seminários. Cabe às instituições supervisionarem e orientarem seus instrutores para o risco destes comportamentos.

O brifim incompleto e a orientação insuficiente do instrutor sobre o procedimento em vento calmo podem indicar falta de experiência e de conhecimento teórico da manobra.

Ambos Informaram que nunca haviam realizado o treinamento em condições de vento "nulo". Esta informação não indica necessariamente uma falha no programa de instrução aérea da Escola. Seria necessária para cada exercício uma análise sobre sua execução nas diversas condições de direção e intensidade de vento.

O vôo de helicóptero requer constante atualização do conhecimento teórico sobre as técnicas de pilotagem, mecânica e aerodinâmica para este tipo de aeronave. Caberia à instituição programar e propiciar estes conhecimentos aos pilotos em formação.

No treinamento de auto-rotação em qualquer helicóptero monomotor, independente das suas características, os pilotos devem optar pelo pouso pontual ou com

velocidade à frente. O tipo de toque dependerá de uma série de fatores: amplitude empregada no "flare", altura de início e eficiência deste, peso da aeronave e, por fim, intensidade do vento. Quanto mais forte o vento, mais eficácia terá o "flare" para desacelerar longitudinalmente a aeronave. Em situações de vento calmo, de través ou de peso elevado o "flare" terá sua eficácia reduzida, levando a um provável pouso com velocidade à frente.

Os pilotos devem ter em mente que, apesar de serem muitas as variáveis, a constante nesta equação deve ser a altura a partir da qual, não importando a velocidade, a atitude da aeronave deve ser nivelada para evitar o afundamento com cauda baixa e colisão da mesma.

Este conhecimento teórico é parte da mecânica da auto-rotação em helicópteros que deve ser divulgada pelas escolas e operadores.

III – CONCLUSÃO

Fatores contribuintes

a. Fator Humano

(1) Fisiológico
Não contribuiu.

(2) Psicológico - Contribuiu

O excesso de autoconfiança e a complacência do instrutor permitiram que o aluno conduzisse a aeronave a uma situação irreversível.

b. Fator Material

Não contribuiu.

c. Fator Operacional

(1} Deficiente Instrução – Contribuiu

O brifim inadequado sobre o treinamento de auto-rotação contribuiu para o inadequado uso dos comandos e análise do exercício por parte do aluno.

A orientação do instrutor em vôo para "chegar mais alto" não foi suficiente para o aluno interpretar como deveria realizar o exercício.

O instrutor não se manteve alerta para atuar a tempo nos comandos em caso de erro do aluno.

(2} Deficiente Julgamento – Contribuiu

O piloto permitiu o afundamento da aeronave com cauda baixa até a colisão desta com o solo, gerando a perda de controle e pouso brusco.

IV – RECOMENDAÇÕES DE SEGURANÇA

Os SERAC deverão:

a. Alertar os operadores de helicópteros em cursos, seminários e DIVOP sobre a importância do conhecimento teórico sobre auto-rotação.

b. Alertar os operadores e escolas durante cursos, vistorias e seminários sobre a importância dos brifins de vôos de instrução e sobre o papel e responsabilidade dos instrutores sobre a condução dos mesmos.

c. Divulgar este acidente enfatizando a importância da padronização na instrução aérea, seguindo um programa de instrução eficaz onde sejam destacados os exercícios mais críticos de cada fase.

"O SIPAER solicita que seja dada ampla divulgação dessa informação a todos os órgãos e operadores que possam estar envolvidos e/ou participar na solução do problema, assim como a adoção de medidas preventivas".

Divulgação Operacional

DO:	**SERAC 5**	ASSUNTO:	**Acidente Aeronáutico**
AOS:	**Escolas de Aviação, Proprietários e Operadores da ANV.**	DATA:	**27/09/2002**
		AERONAVE:	**BELL 206B**

50/2004

Falha do motor em vôo.

I – HISTÓRICO

A aeronave decolou do Aeroporto Internacional Pinto Martins (SBFZ), com quatro passageiros e um piloto a bordo, para a realização de um vôo buscando irregularidades ambientais.

O vôo consistia em sobrevôo visual da região, decolando de SBFZ, indo até o município de Sobral -CE, onde seria feito o reabastecimento. Em seguida, seria realizado mais um sobrevôo da região e retorno para SBFZ. .

Faltando 15 min para chegar ao município de Sobral um passageiro passou mal, sendo feito um pouso para avaliar suas condições. Foi então decidido realizar o regresso direto para SBFZ.

A 4,3 NM de SBFZ, houve apagamento do motor em vôo. O piloto efetuou pouso forçado no leito do rio Siqueira.

Durante o pouso a aeronave sofreu avaria graves. O piloto e um passageiro sofreram lesões graves, os demais lesões leves.

II – ANÁLISE

A aeronave decolou do Aeroporto Internacional Pinto Martins (SBFZ), para sobrevôo visual da região, sendo planejado ir até o município de Sobral -CE, onde seria feito o reabastecimento para retorno a SBFZ.

Para o referido vôo o piloto preencheu uma notificação de vôo, dando como destino o heliponto Miguel Dias (SNMG), num total de 10 min de vôo, contrariando as normas em vigor.

Depois de 2 h 15 min de vôo, interrompidas por dois pousos, quando próximo ao município de Sobral, um dos passageiros passou mal em vôo, sendo feito um pouso para avaliar suas condições. Foi então decidido realizar o regresso direto para SBFZ, distante 120 NM, mesmo estando a 15 min do local de abastecimento, demonstrando inadequada avaliação da situação, visto que o retorno com a velocidade média da aeronave de 80 kt consumiria 01 h 30 min de vôo, extrapolando a sua autonomia em 15 min.

A 4,3 NM de SBFZ, com um total de 3 h 40 min de vôo, houve apagamento do motor em vôo, sendo realizado um pouso forçado no leito do rio Siqueira.

Segundo o manual da aeronave, sua autonomia era de 3 h 30 min, tendo, portanto, o piloto excedido em 10 min a mesma.

Na ação inicial verificou-se que não havia combustível no tanque da aeronave.

É possível que a presença de um passageiro com mal estar, elevando a carga de responsabilidade do piloto, tenha sido fator estressante para o mesmo, de forma a fixar sua atenção no problema, desviando-a dos cálculos necessários ao planejamento do vôo.

Entretanto, o piloto acreditava que havia combustível suficiente, pois informou aos passageiros. Disse que houve o apagamento do motor, sem que houvesse o acendimento da luz de baixo nível de combustível, sendo que neste modelo de aeronave não existe tal luz. A luz de baixo nível o alertaria para a situação antes de ocorrer o apagamento do motor. Tais fatos evidenciam o desconhecimento do equipamento que estava operando, demonstrando que sua adaptação não foi adequada e que ainda não tinha experiência suficiente no equipamento.

A empresa apresentava uma informalidade nos processos de recrutamento, seleção, treinamento e acompanhamento de seus pilotos, sendo o piloto acidentado contratado e colocado em operação sem uma avaliação mais formal de seu desempenho anterior e com um treinamento que não lhe proporcionou o conhecimento completo de operação da aeronave.

A falta de treinamento sistematizado e de reuniões operacionais formais apresenta contextos organizacionais que fogem aos padrões de segurança exigidos para uma empresa aérea.

O comportamento de seus funcionários dependerá diretamente de sua Cultura Organizacional.

Para o retorno foi utilizada a proa magnética 125°.

Os MET AR de SBFZ indicavam vento de 1200 com velocidade de 16 a 20 kt durante o retorno da aeronave, ou seja, vento de proa, sendo possível que a direção e a intensidade poderiam ser diferentes na região sobrevoada. Entretanto, próximo à Fortaleza, pode se afirmar que o vento era de proa, o que certamente comprometeu o alcance da aeronave.

Tal informação não era de conhecimento do piloto, visto que o mesmo não efetuou contato com os órgãos de controle, para quem a aeronave estaria pousada no heliponto Miguel Dias (SNMG), e a escuta da freqüência da TWR, que fornece o vento freqüentemente, não seria possível voando a baixa altura, 500 ft, e a longa distância. Assim o piloto deixou de considerar a influência do vento em seu planejamento de retorno.

II – CONCLUSÃO

FATOR HUMANO:
Fisiológico – Não Contribuiu
Psicológico – Contribuiu

FATOR MATERIAL:
Não Contribuiu

FATOR OPERACIONAL:
Deficiente Instrução – Contribuiu
Pouca Experiência de Vôo na Aeronave - Contribuiu
Deficiente Julgamento – Contribuiu
Deficiente Planejamento – Contribuiu
Deficiente Supervisão – Contribuiu
Deficiente Coordenação de Cabine – Contribuiu

IV – RECOMENDAÇÕES DE SEGURANÇA

Os SERAC deverão, de imediato:

RSV (A) 357/A/04 - CENIPA

Incluir nos Simpósios e Seminários Regionais e nas Jornadas Itinerantes de Segurança de Vôo, palestras específicas a respeito da operação de aeronaves de asas rotativas, alertando os pilotos e operadores para a necessidade de se levar em conta os aspectos de meteorologia e autonomia, bem como das conseqüências legais advindas de um acidente aeronáutico, ante as irregularidades constatadas durante a investigação do acidente.

"O SIPAER solicita que seja dada ampla divulgação dessa informação a todos os órgãos e operadores que possam estar envolvidos e/ou participar na solução do problema, assim como a adoção de medidas preventivas".

Divulgação Operacional

DO: **SERAC 5**	ASSUNTO: **Acidente Aeronáutico**		
AOS: **Aeroclubes.**	DATA: **18 set 1998**		
	AERONAVE: **BELL 206B**	**23/2004**	

Perda de controle em vôo.

I – HISTÓRICO

A aeronave decolou do Aeródromo de Itápolis (SOlO) às 11 h 45 min (HBV), com um piloto a bordo, para realizar um vôo de manutenção operacional de instrutor .

Após a decolagem, o piloto fez uma curva de 180º para retornar no sentido contrário ao da decolagem. Após passar por todo o comprimento da pista e ao cruzar a cabeceira que havia decolado, estando a 120 ft, simulou uma "falha do motor logo após a decolagem" e iniciou os procedimentos para pouso simulado em terreno não preparado (canavial). Em torno de 20 ft de altura, iniciou a arremetida.

Como estava baixo e com inércia descendente, não houve tempo de o motor reagir e a asa direita da aeronave veio tocar o solo. O piloto perdeu o controle direcional, levando a aeronave a colidir com o solo.

Em conseqüência, a aeronave teve danos graves e o piloto saiu ileso.

II – ANÁLISE

Tratava-se de um vôo solo com uma aeronave, pertencente ao Aeroclube de Itápolis, para treinamento de instrução e reciclagem de instrutor .

A aeronave estava em condições de operação adequadas, com suas cadernetas de hélice e motor atualizadas.

A manobra pretendida, simulação de "Falha do motor logo após a decolagem", seguida de pouso de emergência, foi executada fora dos parâmetros estabelecidos pelo Aeroclube, no que concerne à altura da sua execução.

O piloto possuía experiência e qualificação para o tipo de vôo.

Com relação ao fator humano, verificou-se que não houve contribuição do aspecto fisiológico. O piloto estava com seu Certificado de Capacidade Física válido, não possuía qualquer patologia que pudesse interferir na condução do vôo e não estava sob efeito de fadiga.

Quanto ao Aspecto Psicológico, na entrevista realizada com o piloto por ocasião do exame de saúde requerido devido ao acidente, mostrou que o mesmo possui traços que afetaram seu desempenho quando da realização do exercício proposto, com relação aos parâmetros mínimos de altura.

Na característica Personalidade, foi observada uma exacerbação de sua Auto-Imagem, com uma sobrevalorização da experiência profissional -o que explica a falta de atenção (Aspecto da Atenção) e o Excesso de Confiança para realizar o exercício. Todos esses aspectos influenciaram na Tomada de Decisão em realizar o treinamento fora dos parâmetros previstos.

No tocante ao Fator Operacional, é importante salientar que o piloto não fez um correto planejamento do vôo que iria realizar, pois deixando de consultar o altímetro, findou por não observar os limites de altura para o início do exercício e da arremetida.

O piloto também não fez uma correta avaliação quanto à altura de arremetida, achando que conseguiria fazê-Ia mais baixo, cerca de 20 ft, prejudicado duplamente pela não observação dos instrumentos, bem como pelo vento que soprava de cauda.

Por fim, o tripulante levou a manete de potência bruscamente para frente, na tentativa de ter uma

resposta mais rápida do motor. Como este teve um retardo considerado normal, o piloto reduziu e voltou a acelerá-lo novamente mas, desta vez, suavemente. Como não havia mais altura suficiente, a aeronave não conseguiu arremeter .

Verifica-se, então, que o excesso de auto confiança foi fator decisivo para a ocorrência do acidente, posto que levou o piloto a acreditar na auto-suficiência dos seus atos, os quais findaram por traí-Io. Ao descumprir os preceitos da instrução aérea, acabou por envolver-se em uma condição irreversível, que o levou ao acidente. Embotado no seu julgamento, acreditou que seria capaz de conduzir, com sucesso e sem se ater aos instrumentos da aeronave, o treinamento de "Falha do motor logo após a decolagem". Iniciando-o abaixo da altura prevista, bem como retardando o início da arremetida, possivelmente veio a se assustar com a razão de afundamento e com a pouca altura disponível, aplicando bruscamente potência do motor, não obtendo a resposta esperada e necessária para lograr sucesso na manobra.

II – CONCLUSÃO

FATOR HUMANO:
(1) Fisiológico – Não Contribuiu.
(2) Psicológico - Contribuiu.

FATOR OPERACIONAL
Deficiente Aplicação de Comando – Contribuiu
Deficiente Planejamento - Contribuiu
Deficiente Julgamento – Contribuiu
Indisciplina de Vôo – Contribuiu

IV – RECOMENDAÇÕES DE SEGURANÇA

Os SERAC deverão, no prazo de três meses:

RSV (A) 61/B/04 – CENIPA

Exigir que os aeroclubes façam reuniões (no mínimo mensais) com todos os instrutores do seu quadro de tripulantes, a fim de relembrar os procedimentos previstos e os mínimos de segurança para os exercícios a serem realizados quando em instrução, tanto em duplo comando quanto em treinamento solo dos instrutores (manutenção operacional).

Emitida em 26 / 03 / 2004 Cumprida em / /

"O SIPAER solicita que seja dada ampla divulgação dessa informação a todos os órgãos e operadores que possam estar envolvidos e/ou participar na solução do problema, assim como a adoção de medidas preventivas".

Divulgação Operacional

DO:	**SERAC 5**	ASSUNTO:	**Acidente Aeronáutico**	
AOS	**Aeroclubes, Escolas de Aviação e Empresas de Táxi Aéreo.**	DATA:	**02 AGO 2000**	
		AERONAVE:	**SK-76A**	**12/2005**

COLISÃO EM VÔO COM OBSTÁCULO

I – HISTÓRICO

O helicóptero decolou de SDCB, em Cubatão-SP, com plano de vôo visual para SBNF (Navegantes–SC), com cinco ocupantes a bordo, a fim de cumprir vôo administrativo de transporte e troca de funcionários.

O plano de vôo previa uma rota direta, sobre o mar, a 4500ft de altitude.

Próximo ao destino a tripulação reportou para o controle Navegantes estar ascendendo de 500 para 1000 ft com a intenção de se manter em condições de vôo visuais.

Havia forte nevoeiro sobre a região, restringindo a visibilidade horizontal.

A sete milhas de Navegantes, a uma altitude aproximada de 700ft. A aeronave colidiu com o Morro do Pires, município de Penha–SC.

A aeronave ficou completamente destruída e os cinco ocupantes sofreram lesões fatais.

II – ANÁLISE

De acordo com as informações disponíveis, não houve qualquer indício de falha material nos sistemas da aeronave.

Os pilotos decolaram de Cubatão para Navegantes com um plano visual, no nível 045 e rota direta, sobre o oceano.

Na ocasião da decolagem o aeródromo de destino operava fechado por instrumentos, mas a alternativa, Florianópolis, encontrava-se aberto visual.

Durante a rota a tripulação decidiu descer para 1000 ft e, nestas condições, encontravam-se com o solo encoberto por uma camada de nuvens, o chamado "visual notopo da camada".

Conforme a previsto na IMA 100-4, esta situação não se enquadraria como dentro das regras de vôo visuais, pois o piloto, neste caso, não estaria mantendo referências com o solo ou a água.

Mediante as informações obtidas através de contato, via rádio, com a coordenação da empresa, com a PT-HUD e com o controle Navegantes, a tripulação concluiu que realmente o campo passara a operar em condições visuais.

O piloto, baseado nestas informações, decidiu não realizar procedimento de descida por instrumentos em Navegantes, optando por "furar" a camada e prosseguir visual a 500 ft até o destino.

Houve uma tentativa, nestes momentos, de se estabelecer os procedimentos seguintes, caso a situação de visibilidade se deteriorasse. O piloto sugeriu ao co-piloto que subissem, neste caso, para mil pés e assim prosseguissem até SBNF, mesmo que se deparassem em condições IMC. Não foram definidos, neste instante, quais seriam as tarefas a serem executadas pelos pilotos. As características de introspecção e dificuldade de comunicação inerentes ao co-piloto dificultaram esta interação, tendo o mesmo se limitado a responder e executar as solicitações feitas pelo piloto.

Cabe ressaltar que as elevações da rota, próximas a SBNF, tinham altitude máxima de 1001 ft. Uma subida para 1000 ft deixaria uma margem muito pequena de segurança caso a aeronave viesse a adentrar inadvertidamente em condições IMC.

Além disso, a altitude mínima de setor para o vôo por instrumentos em SBNF era de 3000 ft.

Momentos antes do impacto, percebe-se uma motivação excessiva da tripulação em manter o vôo por referências visuais a todo o custo, retardando a decisão de se ascender para uma altitude mais segura o suficiente para se livrar as elevações.

A colisão com o Morro do Pires ocorreu a 700 ft de altitude, o que reflete a reação da tripulação quanto a ascender para uma altitude de segurança, ou mesmo para os 1.000 ft sugeridos pelo piloto.

III – CONCLUSÃO

FATOR HUMANO:
Fisiológico – Não Contribuiu
Psicológico – Contribuiu

FATOR MATERIAL:
Não Contribuiu

FATOR OPERACIONAL:
Deficiente Coordenação de Cabine – Contribuiu
Deficiente Julgamento – Contribuiu
Deficiente Planejamento – Contribuiu
Indisciplina de vôo – Contribuiu
Condições Meteorológicas Adversas – Contribuíram

IV – RECOMENDAÇÕES DE SEGURANÇA

Aos operadores do tipo de aeronave deverão:

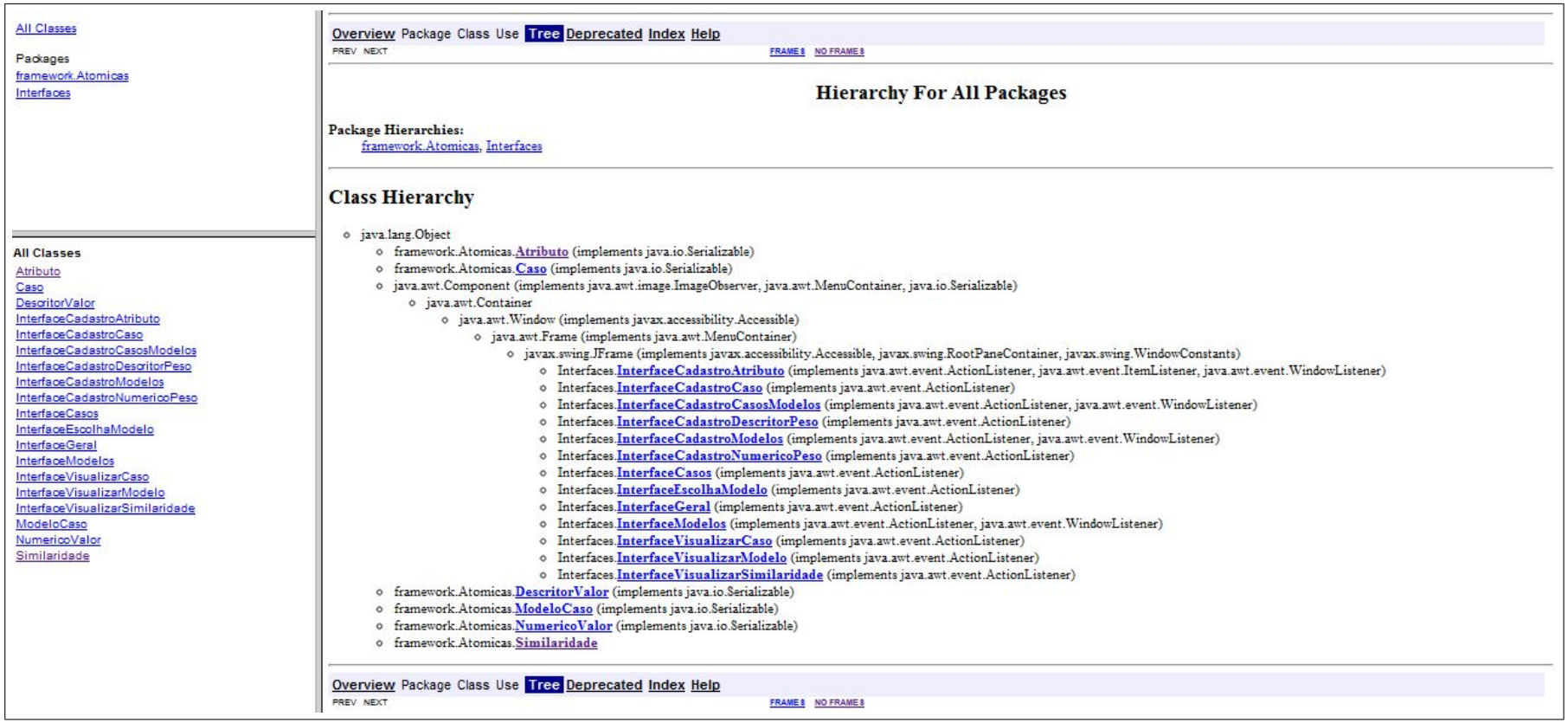
Dar ampla divulgação do conteúdo deste DIVOP em suas empresas para elevar a consciência de Segurança de Vôo, regulamentando um programa de treinamento dos tripulantes para realizar operações com helicópteros em plataformas marítimas, avaliando as peculiaridades referentes ao CFIT.

"O SIPAER solicita que seja dada ampla divulgação dessa informação a todos os órgãos e operadores que possam estar envolvidos e/ou participar na solução do problema, assim como a adoção de medidas preventivas".

APÊNDICE B – Código Fonte Gerado

O desenvolvimento da ferramenta para estruturação da base de conhecimento utilizando raciocínio baseado em casos foi realizado em linguagem Java. Para a execução das diferentes funcionalidades foi necessário trabalhar com a linguagem escolhida afim de oferecer ao usuário ferramentas eficazes. Todo esse desenvolvimento gerou códigos que serão apresentados nesse apêndice.

Primeiramente será exibido a estruturação hierárquica gerada a partir da ferramenta Javadoc, disponibilizado pela Sun Microsystems em diferentes ambientes de desenvolvimento. Posteriormente todo o código utilizado será mostrado.



All Classes

Packages
framework.Atomicas
Interfaces

All Classes
Atributo
Caso
DescritorValor
InterfaceCadastroAtributo
InterfaceCadastroCaso
InterfaceCadastroCasosModelos
InterfaceCadastroDescritorPeso
InterfaceCadastroModelos
InterfaceCadastroNumericoPeso
InterfaceCasos
InterfaceEscolhaModelo
InterfaceGeral
InterfaceModelos
InterfaceVisualizarCaso
InterfaceVisualizarModelo
InterfaceVisualizarSimilaridade
ModeloCaso
NumericoValor
Similaridade

Overview Package Class Use Tree Deprecated Index Help
PREV NEXT FRAMES NO FRAMES

Hierarchy For All Packages

Package Hierarchies:
framework.Atomicas, Interfaces

Class Hierarchy

- java.lang.Object
  - framework.Atomicas.Atributo (implements java.io.Serializable)
  - framework.Atomicas.Caso (implements java.io.Serializable)
  - java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - java.awt.Container
      - java.awt.Window (implements javax.accessibility.Accessible)
        - java.awt.Frame (implements java.awt.MenuContainer)
          - javax.swing.JFrame (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)
            - Interfaces.InterfaceCadastroAtributo (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.awt.event.WindowListener)
            - Interfaces.InterfaceCadastroCaso (implements java.awt.event.ActionListener)
            - Interfaces.InterfaceCadastroCasosModelos (implements java.awt.event.ActionListener, java.awt.event.WindowListener)
            - Interfaces.InterfaceCadastroDescritorPeso (implements java.awt.event.ActionListener)
            - Interfaces.InterfaceCadastroModelos (implements java.awt.event.ActionListener, java.awt.event.WindowListener)
            - Interfaces.InterfaceCadastroNumericoPeso (implements java.awt.event.ActionListener)
            - Interfaces.InterfaceCasos (implements java.awt.event.ActionListener)
            - Interfaces.InterfaceEscolhaModelo (implements java.awt.event.ActionListener)
            - Interfaces.InterfaceGeral (implements java.awt.event.ActionListener)
            - Interfaces.InterfaceModelos (implements java.awt.event.ActionListener, java.awt.event.WindowListener)
            - Interfaces.InterfaceVisualizarCaso (implements java.awt.event.ActionListener)
            - Interfaces.InterfaceVisualizarModelo (implements java.awt.event.ActionListener)
            - Interfaces.InterfaceVisualizarSimilaridade (implements java.awt.event.ActionListener)
  - framework.Atomicas.DescritorValor (implements java.io.Serializable)
  - framework.Atomicas.ModeloCaso (implements java.io.Serializable)
  - framework.Atomicas.NumericoValor (implements java.io.Serializable)
  - framework.Atomicas.Similaridade

Overview Package Class Use Tree Deprecated Index Help
PREV NEXT FRAMES NO FRAMES

```
package framework.Atomicas;

import java.io.Serializable;
import java.util.ArrayList;

/**
 *
 * @author Joice
 */
public class Atributo implements Serializable {

    int tipo;  // 0 - numerico   1 - descritivo
    String nome;
    ArrayList<DescritorValor> descritoresDes;
    ArrayList<NumericoValor> descritoresNum;


    public Atributo() {
        this.descritoresDes = new ArrayList<DescritorValor>();
        this.descritoresNum = new ArrayList<NumericoValor>();
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public ArrayList<DescritorValor> getDescritoresDes() {
        return descritoresDes;
    }

    public void setDescritoresDes(ArrayList<DescritorValor>
descritoresDes) {
        this.descritoresDes = descritoresDes;
    }

    public void addDescritoresDes(DescritorValor a){
        descritoresDes.add(a);
    }

    public ArrayList<NumericoValor> getDescritoresNum() {
        return descritoresNum;
    }

    public void setDescritoresNum(ArrayList<NumericoValor>
descritoresNum) {
        this.descritoresNum = descritoresNum;
    }

    public void addDescritoresNum(NumericoValor a){
        System.out.println("Passou aqui");
        descritoresNum.add(a);
    }
```

```
    public int getTipo() {
        return tipo;
    }

    public void setTipo(int tipo) {
        this.tipo = tipo;
    }

    public String toString(){
        String resultado = this.getNome() + "\t";
        if(this.getTipo() == 0) resultado += " NumÃ©rico \n";
        else resultado += " Descritivo \n";

        if(this.getTipo() == 0){ // numÃ©rico
            for(int i=0; i<this.descritoresNum.size(); i++){
                resultado +=
this.getDescritoresNum().get(i).toString() + " \n";
            }
        }else{ // descritivo
            for(int i=0; i<this.descritoresDes.size(); i++){
                resultado +=
this.getDescritoresDes().get(i).toString() + " \n";
            }
        }
        return (resultado);
    }

}
```

```
package framework.Atomicas;

import java.io.Serializable;
import java.util.ArrayList;

/**
 *
 * @author Joice
 */
public class Caso implements Serializable{

    ModeloCaso modelo;
    String nome;
    ArrayList<Integer> vetor;
    ArrayList<Integer> peso;

    public Caso(){
        vetor = new ArrayList<Integer>();
        peso = new ArrayList<Integer>();
    }

    public ModeloCaso getModelo() {
        return modelo;
```

```
    }

    public void setModelo(ModeloCaso modelo) {
        this.modelo = modelo;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public ArrayList<Integer> getVetor() {
        return vetor;
    }

    public int getVetor(int i){
        return this.vetor.get(i);
    }

    public ArrayList<Integer> getPeso() {
        return peso;
    }

    public int getPeso(int i){
        return this.peso.get(i);
    }


    public void setPosicaoVetor(int i, int v){
        this.vetor.add(i, v);
    }

    public void setPosicaoPeso(int i, int v){
        this.peso.add(i, v);
    }

    public String toString(){
        String resultado = this.getNome() + " / " +
this.getModelo().getNome() + "\n";
        for(int i=0; i<this.getModelo().getAtributos().size();
i++){
            if(this.getPeso(i) == -1){
                resultado +=
this.getModelo().getAtributo(i).getNome() + "\t" + " \n";
            }else{
                if(this.getModelo().getAtributo(i).getTipo() ==
0){// numerico
                    resultado +=
this.getModelo().getAtributo(i).getNome() + "\t";
                    resultado +=
this.getModelo().getAtributo(i).getDescritoresNum().get(this.get
Vetor(i)).getNome() + "\n";
                }else{
```

```
                    resultado +=
this.getModelo().getAtributo(i).getNome() + "\t";
                    resultado +=
this.getModelo().getAtributo(i).getDescritoresDes().get(this.get
Vetor(i)).getDescritor() + "\n";
                }
            }

        }
        return resultado;
    }


}
```

```
package framework.Atomicas;

import java.io.Serializable;

/**
 *
 * @author Joice
 */
public class DescritorValor implements Serializable{
    String descritor;
    int valor;

    public String getDescritor() {
        return descritor;
    }

    public void setDescritor(String descritor) {
        this.descritor = descritor;
    }

    public int getValor() {
        return valor;
    }

    public void setValor(int valor) {
        this.valor = valor;
    }

    public String toString(){
        return (this.getDescritor() + "\t" + this.getValor());
    }

}
```

```
package framework.Atomicas;

import java.io.Serializable;
import java.util.ArrayList;

/**
 *
 * @author Joice
 */
public class ModeloCaso implements Serializable{

    String nome;
    ArrayList<Atributo> atributos;

    public ModeloCaso(){
        atributos = new ArrayList<Atributo>();
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public ArrayList<Atributo> getAtributos() {
        return atributos;
    }

    public void setAtributos(ArrayList<Atributo> atributos) {
        this.atributos = atributos;
    }

    public void modificaAtributos(Atributo a, int idx){
        this.atributos.set(idx, a);
    }

    public Atributo getAtributo(int idx){
        return this.atributos.get(idx);
    }

    public void addAtributo(Atributo a){
        this.atributos.add(a);
    }

    public String toString(){
        String resultado = this.getNome() + "\n";
        for(int i=0; i<this.atributos.size(); i++) resultado +=
this.getAtributo(i).toString() + "\n";
        return resultado;
    }

}
```

```
package framework.Atomicas;

import java.io.Serializable;

/**
 *
 * @author Joice
 */
public class NumericoValor implements Serializable{

    String nome;
    int min, max;
    int valor;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public int getMax() {
        return max;
    }

    public void setMax(int max) {
        this.max = max;
    }

    public int getMin() {
        return min;
    }

    public void setMin(int min) {
        this.min = min;
    }

    public int getValor() {
        return valor;
    }

    public void setValor(int valor) {
        this.valor = valor;
    }

    public String toString(){
        return (this.getNome() + "\t" + this.getMin() + "\t" +
this.getMax() + "\t" + this.getValor());
    }

}
```

```
package framework.Atomicas;

/**
 *
 * @author Joice
 */
public class Similaridade {

    Caso caso1, caso2;
    int tam;

    public Similaridade(Caso c1, Caso c2){
        this.caso1 = c1;
        this.caso2 = c2;
        tam = caso1.getPeso().size();
    }

    public void setCasos(Caso caso1, Caso caso2) {
        this.caso1 = caso1;
        this.caso2 = caso2;
    }

    public Caso getCaso1() {
        return caso1;
    }

    public void setCaso1(Caso caso1) {
        this.caso1 = caso1;
    }

    public Caso getCaso2() {
        return caso2;
    }

    public void setCaso2(Caso caso2) {
        this.caso2 = caso2;
    }

    public double vizinhoProximo(){
        double resultado=0, soma=0;

        for(int i=0; i<tam; i++){
            if(caso1.getPeso(i) != -1 && caso2.getPeso(i) != -1)
                soma += Math.abs(caso1.getPeso(i) -
caso2.getPeso(i));
        }
        resultado = soma / tam;
        return resultado;
    }

    public double distanciaEuclidiana(){
        double resultado=0, soma=0;

        for(int i=0; i<tam; i++){
            if(caso1.getPeso(i) != -1 && caso2.getPeso(i) != -1)
```

```
                soma += Math.pow(caso1.getPeso(i) -
caso2.getPeso(i), 2);
        }
        resultado = Math.sqrt(soma);
        return resultado;
    }

}
```

```
package Interfaces;

import framework.Atomicas.Atributo;
import framework.Atomicas.ModeloCaso;
import java.awt.Checkbox;
import java.awt.CheckboxGroup;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.*;

/**
 *
 * @author Joice
 */
public class InterfaceCadastroAtributo extends JFrame implements
ActionListener, WindowListener, ItemListener{

    JLabel ltipo, latributos, lnome;
    JTextField nome;
    JTextArea area;
    String conteudo;
    JButton ok, cancelar, limpar;
    JButton atrNum, atrDescritivo;
    Checkbox tipo1, tipo2;
    CheckboxGroup grupo;
    JPanel painel;
    SpringLayout layout;
    Container janela;

    Atributo atributo;
    ModeloCaso modelo;
    int operacao;

    public InterfaceCadastroAtributo(ModeloCaso modelo, int op){
        this.modelo = modelo;
        this.operacao = op;
```

```
        if(operacao != 0) atributo =
this.modelo.getAtributo(operacao-1);
        else atributo = new Atributo();
        conteudo = new String();

        painel = new JPanel();
        ltipo = new JLabel("Tipo");
        lnome = new JLabel("Nome");
        latributos = new JLabel("Atributos");
        nome = new JTextField(10);
        grupo = new CheckboxGroup();
        tipo1 = new Checkbox("NumÃ©rico", grupo, false);
        tipo2 = new Checkbox("Descritivo", grupo, false);
        tipo1.addItemListener(this);
        tipo2.addItemListener(this);
        area = new JTextArea(2, 10);


        atrNum = new JButton("NumÃ©rico");
        atrNum.setEnabled(false);
        atrNum.setActionCommand("num");
        atrNum.addActionListener(this);
        atrDescritivo = new JButton("Descritivo");
        atrDescritivo.setEnabled(false);
        atrDescritivo.setActionCommand("desc");
        atrDescritivo.addActionListener(this);
        ok = new JButton("OK");
        ok.addActionListener(this);
        ok.setActionCommand("ok");
        cancelar = new JButton("Cancelar");
        cancelar.addActionListener(this);
        cancelar.setActionCommand("cancelar");
        limpar = new JButton("Apagar");
        limpar.addActionListener(this);
        limpar.setActionCommand("limpar");

        layout = new SpringLayout();
        janela = super.getContentPane();
        janela.setLayout(layout);

        this.TelaCadastro();
    }


    public void TelaCadastro(){
        atributo = new Atributo();

        janela.add(ltipo);
        janela.add(lnome);
        janela.add(latributos);

        janela.add(nome);
        painel = new JPanel(new GridLayout(0, 2));
        painel.add(tipo1);
        painel.add(tipo2);
```

```
        janela.add(painel);

        janela.add(area);

        janela.add(atrNum);
        janela.add(atrDescritivo);
        janela.add(ok);
        janela.add(cancelar);
        janela.add(limpar);

        // setando posicoes
        layout.putConstraint(SpringLayout.WEST, ltipo, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, ltipo, 17,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, lnome, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lnome, 47,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, latributos, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, latributos, 77,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, painel, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, painel, 17,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, nome, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, nome, 47,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, area, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, area, 77,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, atrNum, 35,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, atrNum, 297,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, atrDescritivo,
200, SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, atrDescritivo,
297, SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, ok, 35,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, ok, 337,
SpringLayout.NORTH, janela);
```

```
        layout.putConstraint(SpringLayout.WEST, cancelar, 140,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, cancelar, 337,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, limpar, 250,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, limpar, 337,
SpringLayout.NORTH, janela);

        setLocation(40, 40);
        setTitle("Cadastro de Atributos");
        setSize(420, 450);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);

        verificaOperacao();
        this.repaint();
    }



    public void itemStateChanged(ItemEvent e) {
        if (operacao == 0){ // novo item
            if(e.getItemSelectable().equals(tipo1)){
            // numerico
            atrNum.setEnabled(true);
            atrDescritivo.setEnabled(false);

            } else{
                //descritivo
                atrDescritivo.setEnabled(true);
                atrNum.setEnabled(false);
            }
        }

    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getActionCommand().equals("desc")){
            InterfaceCadastroDescritorPeso iDesc;
            if (operacao == 0) iDesc = new
InterfaceCadastroDescritorPeso(atributo);
            else iDesc = new
InterfaceCadastroDescritorPeso(atributo);
            iDesc.TelaCadastro();
            iDesc.addWindowListener(this);
            tipo1.setEnabled(false);
            atributo.setTipo(1);
        }else if(e.getActionCommand().equals("num")){
            InterfaceCadastroNumericoPeso iNum;
            if (operacao == 0) iNum = new
InterfaceCadastroNumericoPeso(atributo);
            else iNum = new
InterfaceCadastroNumericoPeso(atributo);
```

```
                iNum.TelaCadastro();
                iNum.addWindowListener(this);
                tipo2.setEnabled(false);
                atributo.setTipo(0);
            }else if(e.getActionCommand().equals("ok")){
                if(operacao == 0){
                    atributo.setNome(nome.getText());
                    this.modelo.addAtributo(atributo);
                }else{
                    this.modelo.modificaAtributos(atributo,
operacao-1);
                }
                this.dispose();
            }else if (e.getActionCommand().equals("cancelar")){
                this.dispose();
            }else{
                this.modelo.getAtributos().remove(operacao-1);
                this.dispose();

            }
        }

        @Override
        public void windowOpened(WindowEvent e) {

        }

        @Override
        public void windowClosing(WindowEvent e) {

        }

        @Override
        public void windowClosed(WindowEvent e) {
            preencheArea();
        }

        @Override
        public void windowIconified(WindowEvent e) {

        }

        @Override
        public void windowDeiconified(WindowEvent e) {

        }

        @Override
        public void windowActivated(WindowEvent e) {

        }

        @Override
        public void windowDeactivated(WindowEvent e) {

        }
```

```
    public void verificaOperacao(){
        if(operacao != 0){
            atributo = this.modelo.getAtributo(operacao-1);
            this.nome.setText(atributo.getNome());
            this.nome.setEditable(false);
            if(atributo.getTipo() == 0){
                atrNum.setEnabled(true);
                atrNum.setSelected(true);
                tipo2.setEnabled(false);
                tipo1.setState(true);

            }else{
                atrNum.setEnabled(false);
                atrDescritivo.setEnabled(true);
                tipo1.setEnabled(false);
                tipo2.setState(true);
            }

            preencheArea();
        }
    }

    public void preencheArea(){
        if(atributo.getTipo() == 0){//numerico
            conteudo = "Atributo NumÃ©rico \n";
            for (int i=0; i<atributo.getDescritoresNum().size();
i++){
                conteudo +=
atributo.getDescritoresNum().get(i).getNome() +
                        "\t" +
atributo.getDescritoresNum().get(i).getMin() +
                        " - " +
atributo.getDescritoresNum().get(i).getMax() + "\n";
            }
        }else{
            conteudo = "Atributo Descritivo \n";
            for (int i=0; i<atributo.getDescritoresDes().size();
i++){
                conteudo +=
atributo.getDescritoresDes().get(i).getDescritor() + "\n";
            }
        }
        area.setText(conteudo);
        this.repaint();
    }

}
```

```
package Interfaces;

import framework.Atomicas.Caso;
import framework.Atomicas.ModeloCaso;
```

```
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.*;

/**
 *
 * @author Joice
 */
public class InterfaceCadastroCaso extends JFrame implements
ActionListener{

    Caso c;
    JComboBox combos[];
    JLabel labels[];
    JLabel lmodelo, modelo, lnome;
    JTextField nome;
    JButton ok, cancelar;
    ModeloCaso modeloCaso;
    ArrayList<Caso> casos;
    SpringLayout layout;
    JPanel janela;
    JScrollPane scroll;

    public InterfaceCadastroCaso(ModeloCaso m, ArrayList<Caso>
cs){
        this.modeloCaso = m;
        this.casos = cs;

        labels = new JLabel[100];
        combos = new JComboBox[100];

        lmodelo = new JLabel("Modelo");
        modelo = new JLabel(this.modeloCaso.getNome());
        lnome = new JLabel("Nome");
        nome = new JTextField(10);
        ok = new JButton("OK");
        ok.addActionListener(this);
        ok.setActionCommand("ok");
        cancelar = new JButton("Cancelar");
        cancelar.addActionListener(this);
        cancelar.setActionCommand("cancelar");

        layout = new SpringLayout();
        janela = new JPanel();
        janela.setLayout(layout);

        janela.add(lmodelo);
        layout.putConstraint(SpringLayout.WEST, lmodelo, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lmodelo, 17,
SpringLayout.NORTH, janela);
```

```
        janela.add(modelo);
        layout.putConstraint(SpringLayout.WEST, modelo, 250,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, modelo, 17,
SpringLayout.NORTH, janela);

        janela.add(lnome);
        layout.putConstraint(SpringLayout.WEST, lnome, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lnome, 47,
SpringLayout.NORTH, janela);

        janela.add(nome);
        layout.putConstraint(SpringLayout.WEST, nome, 250,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, nome, 47,
SpringLayout.NORTH, janela);

        janela.add(ok);
        janela.add(cancelar);

        // nomeando labels[]
        for(int i=0; i<modeloCaso.getAtributos().size(); i++){
            labels[i] = new
JLabel(modeloCaso.getAtributo(i).getNome());
            janela.add(labels[i]);
        }

        // preenchendo combos
        for(int i=0; i<m.getAtributos().size(); i++){
            combos[i] = new JComboBox();
            combos[i].addItem(" ");
            String op = new String();
            int tipo = m.getAtributos().get(i).getTipo();
            if(tipo == 0){// numero
                for(int j=0;
j<modeloCaso.getAtributos().get(i).getDescritoresNum().size();
j++){
                    op =
modeloCaso.getAtributos().get(i).getDescritoresNum().get(j).getN
ome() + " - " +

modeloCaso.getAtributos().get(i).getDescritoresNum().get(j).getM
in() + " ~" +

modeloCaso.getAtributos().get(i).getDescritoresNum().get(j).getM
ax();
                    combos[i].addItem(op);
                }
            }else{//descritivo
                for(int j=0;
j<modeloCaso.getAtributos().get(i).getDescritoresDes().size();
j++){
                    op =
modeloCaso.getAtributos().get(i).getDescritoresDes().get(j).getD
escritor();
```

```
                    combos[i].addItem(op);
                }
            }
            janela.add(combos[i]);
        }


        int n = 77;
        for(int i=0; i<m.getAtributos().size(); i++){
            layout.putConstraint(SpringLayout.WEST, labels[i],
5, SpringLayout.WEST, janela);
            layout.putConstraint(SpringLayout.NORTH, labels[i],
n+2, SpringLayout.NORTH, janela);

            layout.putConstraint(SpringLayout.WEST, combos[i],
250, SpringLayout.WEST, janela);
            layout.putConstraint(SpringLayout.NORTH, combos[i],
n, SpringLayout.NORTH, janela);

            n += 30;
        }

        n+=30;
        layout.putConstraint(SpringLayout.WEST, ok, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, ok, n,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, cancelar, 250,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, cancelar, n,
SpringLayout.NORTH, janela);


        janela.setPreferredSize(new Dimension(1000, 1200));
        scroll = new JScrollPane(janela);
        scroll.setAlignmentX(LEFT_ALIGNMENT);

scroll.setVerticalScrollBarPolicy(scroll.VERTICAL_SCROLLBAR_ALWA
YS);
        add(scroll);
        setLocation(40, 40);
        setTitle("Cadastro de Casos");
        setSize(450, 600);
        setVisible(true);

    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getActionCommand().equals("ok")){
            c = new Caso();
            c.setModelo(modeloCaso);
            c.setNome(nome.getText());
            // andando em cada atributo
```

```
            for(int i=0; i<modeloCaso.getAtributos().size();
i++){
                int a = combos[i].getSelectedIndex();
                if(a == 0){
                    c.setPosicaoVetor(i, -1);
                    c.setPosicaoPeso(i, -1);
                }else{
                    c.setPosicaoVetor(i, a-1);
                    if(modeloCaso.getAtributo(i).getTipo() ==
0){ // numerico
                        c.setPosicaoPeso(i,
modeloCaso.getAtributo(i).getDescritoresNum().get(a-
1).getValor());
                    } else{// descritivo
                        c.setPosicaoPeso(i,
modeloCaso.getAtributo(i).getDescritoresDes().get(a-
1).getValor());
                    }
                }

            }
            casos.add(c);
            nome.setText("");
            for(int i=0; i<modeloCaso.getAtributos().size();
i++) combos[i].setSelectedIndex(0);
            this.repaint();
            JOptionPane.showMessageDialog(null, "Caso
cadastrado!");

        }else if(e.getActionCommand().equals("cancelar")){
            this.dispose();
        }
    }


}
```

```
package Interfaces;

import framework.Atomicas.ModeloCaso;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.util.ArrayList;
import javax.swing.*;

/**
 *
 * @author Joice
 */
```

```
public class InterfaceCadastroCasosModelos extends JFrame
implements ActionListener, WindowListener{

    ArrayList<ModeloCaso> modelos;
    ModeloCaso modelo;

    String[] labelsCombo;
    JLabel lmodelos;
    JComboBox combo;
    JButton ok, cancelar, adicionar;
    SpringLayout layout;
    Container janela;

    public InterfaceCadastroCasosModelos(ArrayList<ModeloCaso>
modelos){
        this.modelos = modelos;
        labelsCombo = new String[100];


        lmodelos = new JLabel("Modelos");
        combo = new JComboBox();
        this.preencheCombo();

        ok = new JButton("OK");
        ok.addActionListener(this);
        ok.setActionCommand("ok");
        cancelar = new JButton("Cancelar");
        cancelar.addActionListener(this);
        cancelar.setActionCommand("cancelar");
        adicionar = new JButton("Editar/Criar");
        adicionar.addActionListener(this);
        adicionar.setActionCommand("adicionar");

        layout = new SpringLayout();
        janela = super.getContentPane();
        janela.setLayout(layout);

        this.TelaCadastro();
    }

    public void TelaCadastro(){
        janela.add(lmodelos);
        janela.add(combo);

        janela.add(ok);
        janela.add(cancelar);
        janela.add(adicionar);

        // setando posicoes
        layout.putConstraint(SpringLayout.WEST, lmodelos, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lmodelos, 17,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, combo, 110,
SpringLayout.WEST, janela);
```

```
        layout.putConstraint(SpringLayout.NORTH, combo, 17,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, adicionar, 230,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, adicionar, 17,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, ok, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, ok, 100,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, cancelar, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, cancelar, 100,
SpringLayout.NORTH, janela);


        setLocation(40, 40);
        setTitle("Modelos de Casos");
        setSize(450, 200);
        setVisible(true);


    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getActionCommand().equals("ok")){
            this.dispose();
        }else if(e.getActionCommand().equals("adicionar")){
            if(combo.getSelectedIndex() == 0) this.modelo = new
ModeloCaso();
            else this.modelo =
modelos.get(combo.getSelectedIndex()-1);

            InterfaceCadastroModelos mod = new
InterfaceCadastroModelos(this.modelo);
            mod.addWindowListener(this);
        }else if(e.getActionCommand().equals("cancelar")){
            this.dispose();
        }
    }

    public void preencheCombo(){
        combo.removeAllItems();
        combo.addItem("Novo Modelo");
        for(int i=0; i<labelsCombo.length; i++)
labelsCombo[i]=null;

        for(int i=0; i<this.modelos.size(); i++){
            labelsCombo[i] = this.modelos.get(i).getNome();
            combo.addItem(labelsCombo[i]);
        }
        combo.addActionListener(this);
```

```
    }

    @Override
    public void windowOpened(WindowEvent e) {

    }

    @Override
    public void windowClosing(WindowEvent e) {

    }

    @Override
    public void windowClosed(WindowEvent e) {
        if(combo.getSelectedIndex() == 0)
this.modelos.add(modelo);
        else this.modelo = modelos.set(combo.getSelectedIndex()-
1, modelo);

        this.preencheCombo();
        this.repaint();
    }

    @Override
    public void windowIconified(WindowEvent e) {

    }

    @Override
    public void windowDeiconified(WindowEvent e) {

    }

    @Override
    public void windowActivated(WindowEvent e) {

    }

    @Override
    public void windowDeactivated(WindowEvent e) {

    }

}
```

```
package Interfaces;

import framework.Atomicas.Atributo;
import framework.Atomicas.DescritorValor;
import java.awt.Container;
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
import javax.swing.*;

/**
 *
 * @author Joice
 */
public class InterfaceCadastroDescritorPeso extends JFrame
implements ActionListener{

    DescritorValor desc;
    Atributo atributo;

    JTextField nome;
    JTextField peso;
    JLabel lnome;
    JLabel lpeso;
    JButton ok, cancelar;
    SpringLayout layout;
    Container janela;

    public InterfaceCadastroDescritorPeso(Atributo atributo){
        this.atributo = atributo;

        lnome = new JLabel("Nome");
        lpeso = new JLabel("Peso");

        nome = new JTextField(20);
        peso = new JTextField(2);

        ok = new JButton("OK");
        ok.addActionListener(this);
        ok.setActionCommand("ok");
        cancelar = new JButton("Cancelar");
        cancelar.addActionListener(this);
        cancelar.setActionCommand("cancelar");

        layout = new SpringLayout();
        janela = super.getContentPane();
        janela.setLayout(layout);

        this.TelaCadastro();

    }

    public void TelaCadastro(){
        janela.add(lnome);
        janela.add(lpeso);

        janela.add(nome);
        janela.add(peso);

        janela.add(ok);
        janela.add(cancelar);

        // setando posicoes
```

```
        layout.putConstraint(SpringLayout.WEST, lnome, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lnome, 17,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, lpeso, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lpeso, 47,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, nome, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, nome, 17,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, peso, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, peso, 47,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, ok, 35,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, ok, 100,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, cancelar, 140,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, cancelar, 100,
SpringLayout.NORTH, janela);

        setLocation(40, 40);
        setTitle("Cadastro de Atributos");
        setSize(450, 200);
        setVisible(true);

    }

    public void actionPerformed(ActionEvent e) {
        if(e.getActionCommand().equals("ok")){
            //botao ok
            desc = new DescritorValor();
            desc.setDescritor(nome.getText());
            desc.setValor(Integer.parseInt(peso.getText()));

            this.atributo.addDescritoresDes(desc);
            this.dispose();
        }
        else{
            //botao cancelar
            this.dispose();
        }
    }
```

```
}
```

```
package Interfaces;

import framework.Atomicas.ModeloCaso;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.*;

/**
 *
 * @author lmi5
 */
public class InterfaceCadastroModelos extends JFrame implements
ActionListener, WindowListener{

    ModeloCaso modelo;

    String[] labelsCombo;
    JLabel latributos, lnome;
    JTextField nome;
    JComboBox combo;
    JButton ok, cancelar, adicionar;
    SpringLayout layout;
    Container janela;

    public InterfaceCadastroModelos(ModeloCaso modelo){
        this.modelo = modelo;
        labelsCombo = new String[100];

        latributos = new JLabel("Atributos");
        lnome = new JLabel("Nome");
        if(modelo != null) nome = new
JTextField(modelo.getNome(), 10);
        else {
            nome = new JTextField(10);
        }
        combo = new JComboBox();
        this.preencheCombo();

        ok = new JButton("OK");
        ok.addActionListener(this);
        ok.setActionCommand("ok");
        cancelar = new JButton("Cancelar");
        cancelar.addActionListener(this);
        cancelar.setActionCommand("cancelar");
        adicionar = new JButton("Editar/Criar");
        adicionar.addActionListener(this);
        adicionar.setActionCommand("adicionar");
```

```
        layout = new SpringLayout();
        janela = super.getContentPane();
        janela.setLayout(layout);

        this.TelaCadastro();
    }

    public void TelaCadastro(){
        janela.add(lnome);
        janela.add(latributos);
        janela.add(nome);
        janela.add(combo);

        janela.add(ok);
        janela.add(cancelar);
        janela.add(adicionar);

        // setando posicoes
        layout.putConstraint(SpringLayout.WEST, lnome, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lnome, 17,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, nome, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, nome, 17,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, latributos, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, latributos, 47,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, combo, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, combo, 47,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, adicionar, 400,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, adicionar, 47,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, ok, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, ok, 100,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, cancelar, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, cancelar, 100,
SpringLayout.NORTH, janela);


        setLocation(40, 40);
        setTitle("Cadastro de Modelos de Casos");
```

```
        setSize(600, 200);
        setVisible(true);


    }

    @Override
    public void actionPerformed(ActionEvent e) {
        this.modelo.setNome(nome.getText());
        if(e.getActionCommand().equals("ok")){
            this.dispose();
        }else if(e.getActionCommand().equals("adicionar")){
            InterfaceCadastroAtributo atr = new
InterfaceCadastroAtributo(this.modelo,
combo.getSelectedIndex());
            atr.addWindowListener(this);
        }else if(e.getActionCommand().equals("cancelar")){
            this.dispose();
        }
    }

    public void preencheCombo(){
        combo.removeAllItems();
        combo.addItem("Novo Atributo");
        for(int i=0; i<labelsCombo.length; i++)
labelsCombo[i]=null;

        for(int i=0; i<this.modelo.getAtributos().size(); i++){
            labelsCombo[i] =
this.modelo.getAtributos().get(i).getNome();
            combo.addItem(labelsCombo[i]);
        }
        combo.addActionListener(this);
    }



    @Override
    public void windowOpened(WindowEvent e) {

    }

    @Override
    public void windowClosing(WindowEvent e) {

    }

    @Override
    public void windowClosed(WindowEvent e) {
        preencheCombo();
    }

    @Override
    public void windowIconified(WindowEvent e) {

    }
```

```
    @Override
    public void windowDeiconified(WindowEvent e) {

    }

    @Override
    public void windowActivated(WindowEvent e) {

    }

    @Override
    public void windowDeactivated(WindowEvent e) {

    }

}
```

```
package Interfaces;

import framework.Atomicas.Atributo;
import framework.Atomicas.NumericoValor;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

/**
 *
 * @author Joice
 */
public class InterfaceCadastroNumericoPeso extends JFrame
implements ActionListener{

    NumericoValor num;
    Atributo atributo;

    JTextField nome;
    JTextField min, max;
    JTextField peso;
    JLabel lnome;
    JLabel lfaixa;
    JLabel lpeso;
    JButton ok, cancelar;
    SpringLayout layout;
    Container janela;

    public InterfaceCadastroNumericoPeso(Atributo atributo){
        this.atributo = atributo;

        lnome = new JLabel("Nome");
        lfaixa = new JLabel("Faixa");
        lpeso = new JLabel("Peso");
```

```
        nome = new JTextField(20);
        min = new JTextField(3);
        max = new JTextField(3);
        peso = new JTextField(2);

        ok = new JButton("OK");
        ok.addActionListener(this);
        ok.setActionCommand("ok");
        cancelar = new JButton("Cancelar");
        cancelar.addActionListener(this);
        cancelar.setActionCommand("cancelar");

        layout = new SpringLayout();
        janela = super.getContentPane();
        janela.setLayout(layout);

        this.TelaCadastro();

    }


    public void TelaCadastro(){
        janela.add(lnome);
        janela.add(lfaixa);
        janela.add(lpeso);

        janela.add(nome);
        janela.add(min);
        janela.add(max);
        janela.add(peso);

        janela.add(ok);
        janela.add(cancelar);

        // setando posicoes
        layout.putConstraint(SpringLayout.WEST, lnome, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lnome, 17,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, lfaixa, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lfaixa, 47,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, lpeso, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lpeso, 77,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, nome, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, nome, 17,
SpringLayout.NORTH, janela);
```

```
        layout.putConstraint(SpringLayout.WEST, min, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, min, 47,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, max, 150,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, max, 47,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, peso, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, peso, 77,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, ok, 35,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, ok, 107,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, cancelar, 140,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, cancelar, 107,
SpringLayout.NORTH, janela);

        setLocation(40, 40);
        setTitle("Cadastro de Atributos");
        setSize(450, 200);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        if(e.getActionCommand().equals("ok")){
            //botao ok
            num = new NumericoValor();
            num.setNome(nome.getText());
            num.setMin(Integer.parseInt(min.getText()));
            num.setMax(Integer.parseInt(max.getText()));
            num.setValor(Integer.parseInt(peso.getText()));
            this.atributo.addDescritoresNum(num);

            this.dispose();
        }
        else{
            //botao cancelar
            this.dispose();
        }
    }


}
```

```
package Interfaces;

import framework.Atomicas.Caso;
import framework.Atomicas.ModeloCaso;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

/**
 *
 * @author Joice
 */
public class InterfaceCasos extends JFrame implements
ActionListener{

    Caso c;
    JLabel combos[];
    JLabel labels[];
    JLabel lmodelo, modelo, lnome;
    JTextField nome;
    JButton ok;
    ModeloCaso modeloCaso;
    SpringLayout layout;
    Container janela;

    public InterfaceCasos(ModeloCaso m, Caso cs){
        this.modeloCaso = m;
        this.c = cs;

        int n = modeloCaso.getAtributos().size();
        labels = new JLabel[n];
        combos = new JLabel[n];

        lmodelo = new JLabel("Modelo");
        modelo = new JLabel(this.modeloCaso.getNome());
        lnome = new JLabel("Nome");
        nome = new JTextField(10);
        ok = new JButton("OK");
        ok.addActionListener(this);
        ok.setActionCommand("ok");

        layout = new SpringLayout();
        janela = super.getContentPane();
        janela.setLayout(layout);

        janela.add(lmodelo);
        layout.putConstraint(SpringLayout.WEST, lmodelo, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lmodelo, 17,
SpringLayout.NORTH, janela);

        janela.add(modelo);
        layout.putConstraint(SpringLayout.WEST, modelo, 110,
SpringLayout.WEST, janela);
```

```
        layout.putConstraint(SpringLayout.NORTH, modelo, 17,
SpringLayout.NORTH, janela);

        janela.add(lnome);
        layout.putConstraint(SpringLayout.WEST, lnome, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lnome, 47,
SpringLayout.NORTH, janela);

        janela.add(nome);
        layout.putConstraint(SpringLayout.WEST, nome, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, nome, 47,
SpringLayout.NORTH, janela);

        janela.add(ok);

        // nomeando labels[]
        for(int i=0; i<modeloCaso.getAtributos().size(); i++){
            labels[i] = new
JLabel(modeloCaso.getAtributo(i).getNome());
            janela.add(labels[i]);
        }

        // preenchendo combos
        for(int i=0; i<m.getAtributos().size(); i++){
            combos[i] = new JLabel();
            String op = new String();
            int tipo = m.getAtributos().get(i).getTipo();
            if(tipo == 0){// numero

combos[i].setText(m.getAtributo(i).getDescritoresNum().get(c.get
Vetor(i)).getNome());
            }else{//descritivo

combos[i].setText(m.getAtributo(i).getDescritoresDes().get(c.get
Vetor(i)).getDescritor());
            }
            janela.add(combos[i]);
        }


        int a = 77;
        for(int i=0; i<m.getAtributos().size(); i++){
            layout.putConstraint(SpringLayout.WEST, labels[i],
5, SpringLayout.WEST, janela);
            layout.putConstraint(SpringLayout.NORTH, labels[i],
a, SpringLayout.NORTH, janela);

            layout.putConstraint(SpringLayout.WEST, combos[i],
110, SpringLayout.WEST, janela);
            layout.putConstraint(SpringLayout.NORTH, combos[i],
a, SpringLayout.NORTH, janela);

            a += 30;
        }
```

```
        a+=30;
        layout.putConstraint(SpringLayout.WEST, ok, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, ok, a,
SpringLayout.NORTH, janela);


        setLocation(40, 40);
        setTitle("Casos");
        setSize(450, 600);
        setVisible(true);

    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getActionCommand().equals("ok")){
            this.dispose();
        }
    }


}
```

```
package Interfaces;

import framework.Atomicas.Caso;
import framework.Atomicas.ModeloCaso;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.*;

/**
 *
 * @author Joice
 */
public class InterfaceEscolhaModelo extends JFrame implements
ActionListener{

    JLabel lmodelo;
    JComboBox combo;
    ArrayList<ModeloCaso> modelos;
    ArrayList<Caso> casos;
    ModeloCaso mod;
    String labelsCombo[];

    JButton ok, cancelar;
    SpringLayout layout;
    Container janela;
```

```
    public InterfaceEscolhaModelo(ArrayList<ModeloCaso> modelos,
ArrayList<Caso> casos){
        this.modelos = modelos;
        this.casos = casos;
        labelsCombo = new String[100];

        lmodelo = new JLabel("Modelo");
        combo = new JComboBox();
        this.preencheCombo();

        ok = new JButton("OK");
        ok.addActionListener(this);
        ok.setActionCommand("ok");
        cancelar = new JButton("Cancelar");
        cancelar.addActionListener(this);
        cancelar.setActionCommand("cancelar");

        layout = new SpringLayout();
        janela = super.getContentPane();
        janela.setLayout(layout);

        this.TelaCadastro();
    }

    public void TelaCadastro(){
        janela.add(lmodelo);
        janela.add(combo);

        janela.add(ok);
        janela.add(cancelar);

        // setando posicoes
        layout.putConstraint(SpringLayout.WEST, lmodelo, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lmodelo, 17,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, combo, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, combo, 17,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, ok, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, ok, 60,
SpringLayout.NORTH, janela);

        layout.putConstraint(SpringLayout.WEST, cancelar, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, cancelar, 60,
SpringLayout.NORTH, janela);


        setLocation(40, 40);
        setTitle("Modelos de Casos");
        setSize(300, 150);
```

```
        setVisible(true);


    }

    public void actionPerformed(ActionEvent e) {
        if(e.getActionCommand().equals("ok")){
            if(combo.getSelectedIndex() == -1){
                JOptionPane.showMessageDialog(null, "Nenhum
modelo disponÃvel!\n Cadastre um novo modelo!");
            }else{
                mod = modelos.get(combo.getSelectedIndex());
                InterfaceCadastroCaso caso = new
InterfaceCadastroCaso(mod, casos);
            }

        }else if(e.getActionCommand().equals("cancelar")){
            this.dispose();
        }
    }

    public void preencheCombo(){
        combo.removeAllItems();

        for(int i=0; i<this.modelos.size(); i++){
            labelsCombo[i] = this.modelos.get(i).getNome();
            combo.addItem(labelsCombo[i]);
        }
        combo.addActionListener(this);
    }

}
```

```
package Interfaces;

import framework.Atomicas.Caso;
import framework.Atomicas.ModeloCaso;
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;

/**
 *
 * @author Joice
 */
```

```
public class InterfaceGeral extends JFrame implements
ActionListener{

    JToolBar toolbar;
    JMenuBar menubar;
    JMenu arquivo, visualizacao, cadastro, similaridade;
    JMenuItem novoModelo, novoCaso, sair, salvar, exportar,
carregar, visModelo, visCaso;
    JMenuItem medirSimilaridade;
    Container interno;
    ArrayList<ModeloCaso> modelos;
    ArrayList<Caso> casos;

    public InterfaceGeral(){
        modelos = new ArrayList<ModeloCaso>();
        casos = new ArrayList<Caso>();

        toolbar = new JToolBar(" Ferramentas ");
        menubar = new JMenuBar();
        arquivo = new JMenu(" Arquivo ");
        visualizacao = new JMenu(" VisualizaÃ§Ã£o ");
        cadastro = new JMenu(" Cadastro ");
        similaridade = new JMenu(" Similaridade ");

        novoModelo = new JMenuItem(" Novo Modelo");
        novoModelo.addActionListener(this);
        novoModelo.setActionCommand("novoModelo");
        novoCaso = new JMenuItem(" Novo Caso ");
        novoCaso.addActionListener(this);
        novoCaso.setActionCommand("novoCaso");
        cadastro.add(novoModelo);
        cadastro.add(novoCaso);
        sair = new JMenuItem(" Sair ");
        sair.addActionListener(this);
        sair.setActionCommand("sair");
        salvar = new JMenuItem(" Salvar ");
        salvar.addActionListener(this);
        salvar.setActionCommand("salvar");
        exportar = new JMenuItem(" Exportar ");
        exportar.addActionListener(this);
        exportar.setActionCommand("exportar");
        carregar = new JMenuItem(" Carregar ");
        carregar.addActionListener(this);
        carregar.setActionCommand("carregar");
        arquivo.add(carregar);
        arquivo.add(salvar);
        arquivo.add(exportar);
        arquivo.add(sair);
        visModelo = new JMenuItem(" Visualizar Modelo ");
        visModelo.addActionListener(this);
        visModelo.setActionCommand("visModelo");
        visCaso = new JMenuItem(" Visualizar Caso ");
        visCaso.addActionListener(this);
        visCaso.setActionCommand("visCaso");
        visualizacao.add(visModelo);
        visualizacao.add(visCaso);
```

```
        medirSimilaridade = new JMenuItem(" Definir Similaridade
");
        medirSimilaridade.addActionListener(this);
        medirSimilaridade.setActionCommand("medirSimilaridade");
        similaridade.add(medirSimilaridade);


        menubar.add(arquivo);
        menubar.add(cadastro);
        menubar.add(visualizacao);
        menubar.add(similaridade);

        this.setJMenuBar(menubar);

        interno = this.getContentPane ();
        interno.setLayout (new BorderLayout ());
        interno.add (this.toolbar, BorderLayout.NORTH);

        this.setTitle("Framework de RaciocÃnio Baseado em
Casos");
        this.setVisible(true);
        Dimension screenSize =
this.getToolkit().getScreenSize();
        this.setSize((int)screenSize.getWidth(),
(int)screenSize.getHeight()-50);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getActionCommand().equals("novoModelo")){
            InterfaceCadastroCasosModelos novo = new
InterfaceCadastroCasosModelos(modelos);
        }else if(e.getActionCommand().equals("novoCaso")){
            InterfaceEscolhaModelo m = new
InterfaceEscolhaModelo(modelos, casos);
        }else if(e.getActionCommand().equals("sair")){
            this.dispose();

        }else if(e.getActionCommand().equals("salvar")){
            try{
                    // salvar objetos
                    ObjectOutputStream objOut1 = new
ObjectOutputStream(new FileOutputStream( new File( "modelo.txt"
) ) );
                    objOut1.writeObject( modelos );
                    objOut1.flush();
                    objOut1.close();

                    ObjectOutputStream objOut2 = new
ObjectOutputStream(new FileOutputStream( new File( "caso.txt" )
) );
                    objOut2.writeObject( casos );
                    objOut2.flush();
                    objOut2.close();
```

```
                    JOptionPane.showMessageDialog(null, "Arquivo
salvo com sucesso.");
            }catch ( IOException exc ) {

                JOptionPane.showMessageDialog(null, "Erro ao
salvar arquivo.");
                exc.printStackTrace();

            }
        }else if(e.getActionCommand().equals("exportar")){
            this.gravaModelos();
            this.gravaCasos();

            JOptionPane.showMessageDialog(null, "Arquivos
gerados com sucesso na raiz do projeto");

        }else if(e.getActionCommand().equals("carregar")){
            // carregar dados do arquivo
            try{
                ObjectInputStream objInp = new
ObjectInputStream( new FileInputStream( new File( "modelo.txt" )
) );
                try {
                    modelos = ((ArrayList<ModeloCaso>)
objInp.readObject());
                } catch (ClassNotFoundException ex) {

Logger.getLogger(InterfaceGeral.class.getName()).log(Level.SEVER
E, null, ex);
                }
                objInp.close();


                ObjectInputStream objInp2 = new
ObjectInputStream( new FileInputStream( new File( "caso.txt" ) )
);
                try {
                    casos = ((ArrayList<Caso>)
objInp2.readObject());
                } catch (ClassNotFoundException ex) {

Logger.getLogger(InterfaceGeral.class.getName()).log(Level.SEVER
E, null, ex);
                }
                objInp2.close();


                JOptionPane.showMessageDialog(null, "Arquivos
carregados com sucesso");
            }catch ( IOException exc ) {
                JOptionPane.showMessageDialog(null, "Arquivo
inexistente!");
                exc.printStackTrace();

            }
```

```
        }else if(e.getActionCommand().equals("visModelo")){
            InterfaceVisualizarModelo inter = new
InterfaceVisualizarModelo(modelos);
        }else if(e.getActionCommand().equals("visCaso")){
            InterfaceVisualizarCaso inter = new
InterfaceVisualizarCaso(modelos, casos);
        }else
if(e.getActionCommand().equals("medirSimilaridade")){
            InterfaceVisualizarSimilaridade inter = new
InterfaceVisualizarSimilaridade(casos);
        }

    }


    public void gravaModelos() {
       try {
          // Gravando no arquivo
          File arquivo = new File("saidaModelo.txt");
          FileOutputStream fos = new FileOutputStream(arquivo);
          String texto = "";

          for(int i=0; i<this.modelos.size(); i++) texto +=
this.modelos.get(i).toString() + "\n";
          fos.write(texto.getBytes());
          fos.close();

       }
       catch (Exception ee) {
          ee.printStackTrace();
       }
    }

    public void gravaCasos() {
       try {
          // Gravando no arquivo
          File arquivo = new File("saidaCaso.txt");
          FileOutputStream fos = new FileOutputStream(arquivo);
          String texto = "";

          for(int i=0; i<this.casos.size(); i++) texto +=
this.casos.get(i).toString() + "\n";
          fos.write(texto.getBytes());
          fos.close();

       }
       catch (Exception ee) {
          ee.printStackTrace();
       }
    }

}
```

```
package Interfaces;

import framework.Atomicas.ModeloCaso;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.util.ArrayList;
import javax.swing.*;
import java.awt.Container;
import java.awt.event.WindowListener;

/**
 *
 * @author Joice
 */
public class InterfaceModelos extends JFrame implements
ActionListener, WindowListener{

    JLabel[] dados;
    JButton[] botoes;
    JButton ok, cancelar;
    SpringLayout layout;
    Container janela;
    ArrayList<ModeloCaso> modelos;

    public InterfaceModelos(ArrayList<ModeloCaso> modelos){
        this.modelos = modelos;

        int n = modelos.size();
        dados = new JLabel[n];
        botoes = new JButton[n];

        ok = new JButton("OK");
        ok.addActionListener(this);
        ok.setActionCommand("ok");
        cancelar = new JButton("Cancelar");
        cancelar.addActionListener(this);
        cancelar.setActionCommand("cancelar");

        layout = new SpringLayout();
        janela = super.getContentPane();
        janela.setLayout(layout);

        int a = 47;
        for(int i=0; i<n; i++){
            dados[i] = new JLabel(modelos.get(i).getNome());
            botoes[i] = new JButton("Editar/Atributos");
            botoes[i].addActionListener(this);
            botoes[i].setActionCommand(""+i);


            janela.add(dados[i]);
            janela.add(botoes[i]);

            layout.putConstraint(SpringLayout.WEST, dados[i], 5,
SpringLayout.WEST, janela);
```

```
            layout.putConstraint(SpringLayout.NORTH, dados[i],
a, SpringLayout.NORTH, janela);

            layout.putConstraint(SpringLayout.WEST, botoes[i],
110, SpringLayout.WEST, janela);
            layout.putConstraint(SpringLayout.NORTH, botoes[i],
a-5, SpringLayout.NORTH, janela);

            a += 30;
        }

        a+=60;
        janela.add(ok);
        layout.putConstraint(SpringLayout.WEST, ok, 100,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, ok, a,
SpringLayout.NORTH, janela);

        janela.add(cancelar);
        layout.putConstraint(SpringLayout.WEST, cancelar, 160,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, cancelar, a,
SpringLayout.NORTH, janela);


        setLocation(40, 40);
        setTitle("Modelos Cadastrados");
        setSize(350, 450);
        setVisible(true);

    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getActionCommand().equals("ok")){
            this.dispose();
        }else if(e.getActionCommand().equals("cancelar")){
            this.dispose();
        }else{
            int numMod = Integer.parseInt(e.getActionCommand());

            InterfaceCadastroModelos mod = new
InterfaceCadastroModelos(modelos.get(numMod));
            mod.addWindowListener(this);
        }
    }

    @Override
    public void windowOpened(WindowEvent e) {

    }

    @Override
    public void windowClosing(WindowEvent e) {

    }
```

```
    @Override
    public void windowClosed(WindowEvent e) {
        carregar();
        this.repaint();
    }

    @Override
    public void windowIconified(WindowEvent e) {

    }

    @Override
    public void windowDeiconified(WindowEvent e) {

    }

    @Override
    public void windowActivated(WindowEvent e) {

    }

    @Override
    public void windowDeactivated(WindowEvent e) {

    }

    public void carregar(){
        int n = modelos.size();
        for(int i=0; i<n; i++){
            dados[i].setText(modelos.get(i).getNome());
        }
    }
}
```

```
package Interfaces;

import framework.Atomicas.Caso;
import framework.Atomicas.ModeloCaso;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.*;

/**
 *
 * @author Joice
 */
public class InterfaceVisualizarCaso extends JFrame implements
ActionListener{
```

```
    Caso c;
    JComboBox comboModelo, comboCaso;
    JLabel labels[], opcoes[];
    JLabel lcaso;
    JButton ok1, ok2, cancelar;
    ArrayList<ModeloCaso> modeloCaso;
    ArrayList<Caso> casos, casosSelecionados;
    ModeloCaso modelo;
    Caso caso;
    SpringLayout layout;
    JPanel janela;
    JScrollPane scroll;
    int n;

    public InterfaceVisualizarCaso(ArrayList<ModeloCaso> m,
ArrayList<Caso> c){
        this.modeloCaso = m;
        this.casos = c;


        comboModelo = new JComboBox();
        comboCaso = new JComboBox();
        labels = new JLabel[100];
        opcoes = new JLabel[100];
        preencheComboModelo();

        lcaso = new JLabel();
        lcaso.setForeground(Color.red);
        ok1 = new JButton("OK");
        ok1.addActionListener(this);
        ok1.setActionCommand("ok1");
        ok2 = new JButton("OK");
        ok2.addActionListener(this);
        ok2.setActionCommand("ok2");
        cancelar = new JButton("Cancelar");
        cancelar.addActionListener(this);
        cancelar.setActionCommand("cancelar");

        layout = new SpringLayout();
        janela = new JPanel();
        janela.setLayout(layout);

        janela.add(comboModelo);
        layout.putConstraint(SpringLayout.WEST, comboModelo, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, comboModelo,
17, SpringLayout.NORTH, janela);

        janela.add(ok1);
        layout.putConstraint(SpringLayout.WEST, ok1, 350,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, ok1, 17,
SpringLayout.NORTH, janela);

        janela.add(comboCaso);
```

```
        layout.putConstraint(SpringLayout.WEST, comboCaso, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, comboCaso, 57,
SpringLayout.NORTH, janela);

        janela.add(ok2);
        layout.putConstraint(SpringLayout.WEST, ok2, 350,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, ok2, 57,
SpringLayout.NORTH, janela);

        janela.add(lcaso);
        layout.putConstraint(SpringLayout.WEST, lcaso, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lcaso, 87,
SpringLayout.NORTH, janela);

        janela.add(cancelar);
        layout.putConstraint(SpringLayout.WEST, cancelar, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, cancelar, 150,
SpringLayout.NORTH, janela);

        janela.setPreferredSize(new Dimension(1000, 1200));
        scroll = new JScrollPane(janela);
        scroll.setAlignmentX(LEFT_ALIGNMENT);

scroll.setVerticalScrollBarPolicy(scroll.VERTICAL_SCROLLBAR_ALWA
YS);
        add(scroll);
        setLocation(40, 40);
        setTitle("VisualizacÃ£o de Casos");
        setSize(450, 600);
        setVisible(true);

    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getActionCommand().equals("ok1")){
            int a = comboModelo.getSelectedIndex();
            if(a == -1) JOptionPane.showMessageDialog(null,
"Nenhum modelo foi escolhido");
            else modelo = modeloCaso.get(a);
            casosSelecionados = new ArrayList<Caso>();
            lcaso.setText(modelo.getNome());
            preencheComboCaso();
        }else if(e.getActionCommand().equals("ok2")){
            int a = comboCaso.getSelectedIndex();
            if(a == -1) JOptionPane.showMessageDialog(null,
"Nenhum caso foi escolhido");
            else caso = casosSelecionados.get(a);

            preencheTela();

        }else if(e.getActionCommand().equals("cancelar")){
```

```
            this.dispose();
        }
    }

    public void preencheComboModelo(){
        for(int i=0; i<modeloCaso.size(); i++){
            comboModelo.addItem(modeloCaso.get(i).getNome());
        }
        this.repaint();
    }

    public void preencheComboCaso(){
        comboCaso.removeAllItems();
        getCasos();
        System.out.println(casos.size());
        for(int i=0; i<this.casosSelecionados.size(); i++){
            comboCaso.addItem("" + i + " - " +
casosSelecionados.get(i).getNome());
        }
        this.repaint();
    }

    public void getCasos(){

        for(int i=0; i<casos.size(); i++){

if(casos.get(i).getModelo().getNome().equals(this.modelo.getNome
())) {
                casosSelecionados.add(casos.get(i));
            }
        }
    }

    public void preencheTela(){
        for(int i=0; i<100; i++){
            labels[i] = new JLabel();
            opcoes[i] = new JLabel();
            janela.remove(labels[i]);
            janela.remove(opcoes[i]);
        }

        lcaso.setText(modelo.getNome() + " / " +
comboCaso.getSelectedIndex());
        // nomeando labels[]
        for(int i=0; i<modelo.getAtributos().size(); i++){
            labels[i].setText(modelo.getAtributo(i).getNome());
            janela.add(labels[i]);
        }

        // preenchendo opcoes[]
        for(int i=0; i<modelo.getAtributos().size(); i++){
            if(caso.getVetor(i) == -1) opcoes[i].setText(" ");
            else{
                int tipo =
modelo.getAtributos().get(i).getTipo();
                if(tipo == 0){// numero
```

```
opcoes[i].setText(modelo.getAtributos().get(i).getDescritoresNum
().get(caso.getVetor(i)).getNome());
                }else{//descritivo

opcoes[i].setText(modelo.getAtributos().get(i).getDescritoresDes
().get(caso.getVetor(i)).getDescritor());
                }
            }

            janela.add(opcoes[i]);
        }


        int n = 117;
        for(int i=0; i<modelo.getAtributos().size(); i++){
            layout.putConstraint(SpringLayout.WEST, labels[i],
5, SpringLayout.WEST, janela);
            layout.putConstraint(SpringLayout.NORTH, labels[i],
n, SpringLayout.NORTH, janela);

            layout.putConstraint(SpringLayout.WEST, opcoes[i],
250, SpringLayout.WEST, janela);
            layout.putConstraint(SpringLayout.NORTH, opcoes[i],
n, SpringLayout.NORTH, janela);

            n += 30;
        }

        layout.putConstraint(SpringLayout.WEST, cancelar, 250,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, cancelar, n+40,
SpringLayout.NORTH, janela);

        this.repaint();
    }


}
```

```
package Interfaces;

import framework.Atomicas.Caso;
import framework.Atomicas.ModeloCaso;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.*;
```

```
/**
 *
 * @author Joice
 */
public class InterfaceVisualizarModelo extends JFrame implements
ActionListener{

    Caso c;
    JComboBox combos[], comboModelo;
    JLabel labels[];
    JLabel lmodelo;
    JButton ok, cancelar;
    ArrayList<ModeloCaso> modeloCaso;
    ModeloCaso modelo;
    SpringLayout layout;
    JPanel janela;
    JScrollPane scroll;
    int n;

    public InterfaceVisualizarModelo(ArrayList<ModeloCaso> m){
        this.modeloCaso = m;

        comboModelo = new JComboBox();
        labels = new JLabel[100];
        combos = new JComboBox[100];
        for(int i=0; i<100; i++){
            labels[i] = new JLabel();
            combos[i] = new JComboBox();
        }
        preencheComboModelo();

        lmodelo = new JLabel();
        lmodelo.setForeground(Color.red);
        ok = new JButton("OK");
        ok.addActionListener(this);
        ok.setActionCommand("ok");
        cancelar = new JButton("Cancelar");
        cancelar.addActionListener(this);
        cancelar.setActionCommand("cancelar");

        layout = new SpringLayout();
        janela = new JPanel();
        janela.setLayout(layout);

        janela.add(comboModelo);
        layout.putConstraint(SpringLayout.WEST, comboModelo, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, comboModelo,
17, SpringLayout.NORTH, janela);

        janela.add(ok);
        layout.putConstraint(SpringLayout.WEST, ok, 250,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, ok, 17,
SpringLayout.NORTH, janela);
```

```
        janela.add(lmodelo);
        layout.putConstraint(SpringLayout.WEST, lmodelo, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lmodelo, 57,
SpringLayout.NORTH, janela);

        janela.add(cancelar);
        layout.putConstraint(SpringLayout.WEST, cancelar, 110,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, cancelar, 200,
SpringLayout.NORTH, janela);

        janela.setPreferredSize(new Dimension(1000, 1200));
        scroll = new JScrollPane(janela);
        scroll.setAlignmentX(LEFT_ALIGNMENT);

scroll.setVerticalScrollBarPolicy(scroll.VERTICAL_SCROLLBAR_ALWA
YS);
        add(scroll);
        setLocation(40, 40);
        setTitle("VisualizacÃ£o de Modelos");
        setSize(450, 600);
        setVisible(true);

    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getActionCommand().equals("ok")){
            int a = comboModelo.getSelectedIndex();
            if(a == -1) JOptionPane.showMessageDialog(null,
"Nenhum modelo foi escolhido");
            else modelo = modeloCaso.get(a);

            preencheTela();
        }else if(e.getActionCommand().equals("cancelar")){
            this.dispose();
        }
    }

    public void preencheComboModelo(){
        for(int i=0; i<modeloCaso.size(); i++){
            comboModelo.addItem(modeloCaso.get(i).getNome());
        }
    }

    public void preencheTela(){
        for(int i=0; i<100; i++){
            labels[i].setText(" ");
            combos[i].removeAllItems();
            janela.remove(labels[i]);
            janela.remove(combos[i]);
        }


        lmodelo.setText(modelo.getNome());
```

```
        // nomeando labels[]
        for(int i=0; i<modelo.getAtributos().size(); i++){
            labels[i].setText(modelo.getAtributo(i).getNome());
            janela.add(labels[i]);
        }

        // preenchendo combos
        for(int i=0; i<modelo.getAtributos().size(); i++){
            String op = new String();
            int tipo = modelo.getAtributos().get(i).getTipo();
            if(tipo == 0){// numero
                for(int j=0;
j<modelo.getAtributos().get(i).getDescritoresNum().size(); j++){
                    op =
modelo.getAtributos().get(i).getDescritoresNum().get(j).getNome(
) + " - " +

modelo.getAtributos().get(i).getDescritoresNum().get(j).getMin()
+ " ~" +

modelo.getAtributos().get(i).getDescritoresNum().get(j).getMax()
;
                    combos[i].addItem(op);
                }
            }else{//descritivo
                for(int j=0;
j<modelo.getAtributos().get(i).getDescritoresDes().size(); j++){
                    op =
modelo.getAtributos().get(i).getDescritoresDes().get(j).getDescr
itor();
                    combos[i].addItem(op);
                }
            }
            janela.add(combos[i]);
        }


        int n = 77;
        for(int i=0; i<modelo.getAtributos().size(); i++){
            layout.putConstraint(SpringLayout.WEST, labels[i],
5, SpringLayout.WEST, janela);
            layout.putConstraint(SpringLayout.NORTH, labels[i],
n+5, SpringLayout.NORTH, janela);

            layout.putConstraint(SpringLayout.WEST, combos[i],
250, SpringLayout.WEST, janela);
            layout.putConstraint(SpringLayout.NORTH, combos[i],
n, SpringLayout.NORTH, janela);

            n += 30;
        }

        layout.putConstraint(SpringLayout.WEST, cancelar, 250,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, cancelar, n+40,
SpringLayout.NORTH, janela);
```

```
        this.repaint();
    }


}
```

```
package Interfaces;

import framework.Atomicas.*;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.*;

/**
 *
 * @author Joice
 */
public class InterfaceVisualizarSimilaridade extends JFrame
implements ActionListener{
    //variaveis
    Caso c;
    JComboBox comboCaso, comboSim;
    JLabel labels[], pesos[];
    JLabel lcaso, lsimilaridade;
    JButton ok, cancelar;
    ArrayList<Caso> casos, casosSelecionados;
    ArrayList<Double> similaridades;
    Caso caso;
    SpringLayout layout;
    JPanel janela;
    JScrollPane scroll;

    public InterfaceVisualizarSimilaridade(ArrayList<Caso> c){
        //inializando variaveis
        this.casos = c;

        labels = new JLabel[100];
        pesos = new JLabel[100];
        comboCaso = new JComboBox();
        comboSim = new JComboBox();
        this.preencheCombos();
        for(int i=0; i<100; i++){
            labels[i] = new JLabel();
            pesos[i] = new JLabel();
        }

        lcaso = new JLabel(" Selecione o caso ");
        lsimilaridade = new JLabel(" Selecione a medida ");
        ok = new JButton("OK");
        ok.addActionListener(this);
```

```
        ok.setActionCommand("ok");
        cancelar = new JButton("Cancelar");
        cancelar.addActionListener(this);
        cancelar.setActionCommand("cancelar");

        layout = new SpringLayout();
        janela = new JPanel();
        janela.setLayout(layout);

        janela.add(lcaso);
        layout.putConstraint(SpringLayout.WEST, lcaso, 5,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lcaso, 19,
SpringLayout.NORTH, janela);

        janela.add(comboCaso);
        layout.putConstraint(SpringLayout.WEST, comboCaso, 250,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, comboCaso, 17,
SpringLayout.NORTH, janela);

        janela.add(lsimilaridade);
        layout.putConstraint(SpringLayout.WEST, lsimilaridade,
5, SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, lsimilaridade,
49, SpringLayout.NORTH, janela);

        janela.add(comboSim);
        layout.putConstraint(SpringLayout.WEST, comboSim, 250,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, comboSim, 47,
SpringLayout.NORTH, janela);

        janela.add(ok);
        layout.putConstraint(SpringLayout.WEST, ok, 500,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, ok, 47,
SpringLayout.NORTH, janela);

        janela.add(cancelar);
        layout.putConstraint(SpringLayout.WEST, cancelar, 400,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, cancelar, 150,
SpringLayout.NORTH, janela);

        janela.setPreferredSize(new Dimension(1000, 1200));
        scroll = new JScrollPane(janela);
        scroll.setAlignmentX(LEFT_ALIGNMENT);

scroll.setVerticalScrollBarPolicy(scroll.VERTICAL_SCROLLBAR_ALWA
YS);
        add(scroll);
        setLocation(40, 40);
        setTitle("Similaridade entre Casos");
        setSize(700, 600);
        setVisible(true);
```

```
        ok.requestFocus();

    }

    public void preencheCombos(){
        for(int i=0; i<casos.size(); i++){
            comboCaso.addItem(casos.get(i).getNome() + " / " +
casos.get(i).getModelo().getNome());
        }

        comboSim.addItem("Vizinho mais PrÃ³ximo");
        comboSim.addItem("DistÃ¢ncia Euclidiana");

    }

    public void getCasos(){
        casosSelecionados = new ArrayList<Caso>();
        for(int i=0; i<casos.size(); i++){

if(casos.get(i).getModelo().getNome().equals(this.caso.getModelo
().getNome())){
                casosSelecionados.add(casos.get(i));
            }
        }

    }

    public void carregaSim(){
        similaridades = new ArrayList<Double>();
        Similaridade sim;
        for(int i=0; i<casosSelecionados.size(); i++){
            sim = new Similaridade(casosSelecionados.get(i),
this.caso);
            if(comboSim.getSelectedIndex() == 0)
similaridades.add(sim.vizinhoProximo());
            if(comboSim.getSelectedIndex() == 1)
similaridades.add(sim.distanciaEuclidiana());
        }
    }

    public void preencheTela(){
        for(int i=0; i<100; i++){
            labels[i].setText(" ");
            pesos[i].setText(" ");
            janela.remove(labels[i]);
            janela.remove(pesos[i]);
        }

        for(int i=0; i<casosSelecionados.size(); i++){

labels[i].setText(casosSelecionados.get(i).getNome());
            janela.add(labels[i]);
        }

        for(int i=0; i<casosSelecionados.size(); i++){
            pesos[i].setText(""+similaridades.get(i));
```

```
            janela.add(pesos[i]);
        }


        int n=107;
        for(int i=0; i<casosSelecionados.size(); i++){
            layout.putConstraint(SpringLayout.WEST, labels[i],
5, SpringLayout.WEST, janela);
            layout.putConstraint(SpringLayout.NORTH, labels[i],
n+2, SpringLayout.NORTH, janela);

            layout.putConstraint(SpringLayout.WEST, pesos[i],
400, SpringLayout.WEST, janela);
            layout.putConstraint(SpringLayout.NORTH, pesos[i],
n, SpringLayout.NORTH, janela);

            n += 30;
        }

        layout.putConstraint(SpringLayout.WEST, cancelar, 400,
SpringLayout.WEST, janela);
        layout.putConstraint(SpringLayout.NORTH, cancelar, n+40,
SpringLayout.NORTH, janela);

        this.repaint();

    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getActionCommand().equals("ok")){
            this.caso = casos.get(comboCaso.getSelectedIndex());
            getCasos();
            carregaSim();
            preencheTela();
            this.repaint();
        }else if(e.getActionCommand().equals("cancelar")){
            this.dispose();
        }
    }


}
```