Dissertação de Mestrado

Plataforma para Construção de Ambientes Interativos de Aprendizagem baseados em Agentes

Ig Ibert Bittencourt Santana Pinto
ibert@tci.ufal.br

Orientador:

Prof. Dr. Evandro de Barros Costa

Maceió, Outubro de 2006

Ig Ibert Bittencourt Santana Pinto

Plataforma para Construção de Ambientes Interativos de Aprendizagem baseados em Agentes

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Curso de Mestrado em Modelagem Computacional de Conhecimento da Universidade Federal de Alagoas.

Orientador:

Prof. Dr. Evandro de Barros Costa

Maceió, Outubro de 2006

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária : Renata Barros Domingos

P659p Pinto, Ig Ibert Bittencourt Santana.
Plataforma para construção de ambientes interativos de aprendizagem baseados em agentes / Ig Ibert Bittencourt Santana Pinto. – Maceió, 2006.
xii, 131 f. : il.

Orientador: Evandro de Barros Costa.
Dissertação (mestrado em Modelagem Computacional de Conhecimento) – Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2006.

Bibliografia: f. 100-112.

1. Ambientes interativos de aprendizagem. 2. Agentes de software. 3.Ontologia. 4. Framework. I. Título.

CDU: 004.89

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Curso de Mestrado em Modelagem Computacional de Conhecimento da Universidade Federal de Alagoas, aprovada pela comissão examinadora que abaixo assina.

Prof. Dr. Evandro de Barros Costa - Orientador
Mestrado em Modelagem Computacional do Conhecimento
Universidade Federal de Alagoas

Prof. Dr. Angelo Perkusich - Examinador
Departamento de Engenharia Elétrica
Universidade Federal de Campina Grande

Profa. Dr. María del Rosario Girardi - Examinador
Departamento de Informática
Universidade Federal do Maranhão

Prof. Dr. Guilherme Bittencourt - Examinador
Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina

Maceió, Outubro de 2006

Dedicatória

Dedico esta dissertação ao meu pai e a meu orientador, através da adaptação de uma frase de Isaac Newton, dizendo: “Busco avistar mais longe ao ficar de pé em ombros de gigantes”.

Agradecimentos

Agradeço primeiramente a DEUS, por proporcionar-me tantas alegrias e desafios em minha vida.

Aos meus companheiros do mestrado (Ulisses, Danielle, Elba e Alencar, Alexandre, ...) pelas noites perdidas com os estudos das disciplinas e, conseqüentemente, ganhas pela obtenção do conhecimento.

Aos “mowrals” Alan, Paulista, as Camilas e Rominho, que estiveram presentes em praticamente todo o andamento do trabalho. Agradecimento especial ao Alan e Paulista que ajudaram bastante na fase final da dissertação, juntamente com Badu e Guilherme.

Ao meu Orientador por acreditar em mim no início do trabalho e pela sua imensa capacidade de orientação, transferência de conhecimento e a liberdade dada durante a pesquisa.

A minha namorada, pela imensurável compreensão nos momentos em que estive ausente e apoio/motivação nos momentos difíceis.

Aos meus amados pais e irmãos por sempre me apoiarem e acompanharem em todos os momentos de minha vida.

A todos que, de alguma forma, contribuíram para o andamento do meu trabalho e peço desculpas àqueles que não foram citados.

Por fim, agradeço a Capes pelo apoio financeiro.

Resumo

O presente trabalho aborda a concepção e desenvolvimento de ambientes interativos de aprendizagem (AIAs) baseados em agentes, mais especificamente, seguindo o modelo de arquitetura multiagente do Mathema. Tal modelo é definido com base em uma estratégia de aprendizagem baseada em problemas, disposta em um cenário onde agentes artificiais e humanos (alunos e professores) interagem com vistas a ajudar aprendizes humanos a resolver problemas em um determinado domínio de conhecimento. Nesse sentido, esse modelo propõe-se a ajudar os aprendizes durante as várias fases para a construção de uma solução, incluindo a análise do problema. Para tanto, modelou-se três categorias de interações: o agente aprendiz humano, o professor e o agente tutor artificial. Neste trabalho, propõe-se uma plataforma para construção de ambientes interativos de aprendizagem baseados em Agentes, dando suporte tanto de um framework, para os engenheiros de software/desenvolvedores, quanto de um sistema de autoria para os usuários não programadores poderem configurar o domínio de ensino. Além disso, tal plataforma possui ferramentas de colaboração, infraestrutura baseada em ontologia e uma sociedade de agentes que auxiliam na modelagem do domínio e resolução de problemas (através de técnicas de Inteligência Artificial). A fim de experimentar e validar a plataforma proposta foram desenvolvidos dois estudos de casos, um na área de direito e o outro em medicina, além de ter sido iniciado um investimento em matemática.

Abstract

This dissertation presents the design and development of an agent-based interactive learning environment based on the Mathema Model. Such model adopts a problem-based learning approach as a pedagogical method that is achieved in an interaction environment that is populated by software and human agents (students and teachers), working together in favour of the human learners. This interaction occurs aiming at helping human learners to solve problems into given knowledge domain. This model aims to help learners during several phases involved in the build solution process, including the problem analysis. With this sense, the interaction between three agent categories was modeled: human learner agent, teacher and artificial tutoring agent. This work proposes a platform for building agent-based interactive learning environments, supporting as framework to developers/software engineers as authoring to non developers users like teachers and knowledge engineers. In addition, the platform has collaborative tools, an ontology-based infrastructure and an agent society that help at the domain modeling and problem solving process (through artificial intelligence techniques). It was conducted two case studies (at legal domain and health domain) and an initial investment at mathematical domain that demonstrate that the proposed platform is feasible.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Códigos Anotados

Capítulo 1

Introdução

O domínio de estudo abordado na presente dissertação é o da concepção e desenvolvimento de ambientes interativos de aprendizagem mediados por computador. Trata-se de um domínio complexo e inerentemente interdisciplinar. A concepção de tais ambientes envolve principalmente aspectos das áreas de Educação, Computação (particularmente a Inteligência Artificial) e Psicologia Cognitiva, isso, pelo menos, foi o que se revelou nos primeiros sistemas, com propósito de ser adaptados aos seus usuários, os quais surgiram nas décadas de 70 e 80 do século XX, os denominados sistemas tutores inteligentes (STIs) (WU, 1993). Já no que se refere ao desenvolvimento de tais ambientes, a área de Computação, através da Engenharia de software, ocupa papel de destaque no provimento de técnicas, métodos, metodologias e ferramentas para apoiar a construção, manutenção e a evolução de ambientes interativos de aprendizagem (MURRAY, 2003).

Uma perspectiva histórica dos ambientes de aprendizagem em questão remonta às décadas de 50 e 60 do século XX, com a chegada dos sistemas CAIs (Computer-Assisted Instruction, idealização baseada na instrução programada) (MOLAR, 1997). No final da década de 60 do século XX, opondo-se à proposta dos CAIs, houve o surgimento dos Micromundos, valorizando e abordando uma proposta de aprendizagem através da ação(PAPERT, 1985, 1987). Outras abordagens relacionadas à aprendizagem, que surgiram, aproximadamente, na mesma época, referem-se a Simuladores e Jogos Educacionais. Com a inserção da Inteligência Artificial, deu-se origem aos *ICAI* (*Intelligent CAI*) ou STI (Sistemas Tutores Inteligentes), os quais foram concebidos para prover os estudantes com informações, tutoria personalizada (CENTINIA et al., 1995), além de mecanismos inteligentes para resolução au-

tomática de problemas. O foco no processo de aprendizagem leva em consideração 3 (três) perguntas básicas: i) o que ensinar?, ii) como ensinar? e iii) para quem ensinar? As respostas destas perguntas levaram à arquitetura clássica de um sistema tutor inteligente, onde o que ensinar está relacionado com o modelo do domínio (aqui se inclui um módulo especialista capaz de resolver problemas no domínio), *como ensinar* com o modelo pedagógico (representam as estratégias de tutoria que conduzem às interações com os estudantes) e *para quem ensinar* com o modelo do estudante (assumindo como um recurso necessário para que o sistema possa adaptar suas ações à demanda de um determinado aprendiz). Além disso, ao longo dos anos, além de aspectos cognitivos, STI começaram a levar em consideração aspectos afetivos (RODRIGUES; CARVALHO, 2004) e motivacionais (BEAL; LEE, 2005; JOHNSON et al., 2004; MATSUDA; VANLEHN, 2000), no processo de ensino-aprendizagem.

Na década de 90 do século XX, e atualmente constatou-se um movimento em favor de modelos computacionais de apoio à aprendizagem calcados na noção de cooperação. Nesse sentido, propõem-se que os STIs tradicionais evoluam para os Ambientes Interativos/Inteligentes de Aprendizagem [1] ou ainda Sistemas Tutores Cooperativos (COSTA, 1997). Além disso, essa década foi marcada pelo advento da Web e agentes de software, provendo mais recursos aos ambientes de ensino à distância (EaDs)(NASSEH, 1997; MORABITO, 1997).

Por outro lado, a construção de ambientes, nos moldes anteriormente destacados, tem, normalmente, se mostrado uma tarefa árdua, exigindo bastante esforço tanto dos engenheiros de software quanto dos autores (não programadores). Tais esforços vão desde desenhar toda a arquitetura do sistema até a preocupação com aspectos de manutenção (conteúdo, funcionalidades, etc) do ambiente (RODRIGUES; SANTOS, 2005). Tais esforços fazem com que a complexidade e os custos sejam bastante altos.

A constatação da complexidade e os diversos custos envolvidos no desenvolvimento e manutenção destes ambientes fizeram com que surgissem os sistemas educacionais de autoria, buscando garantir facilidades ao autor no processo de construção do sistema educacional, a exemplo do proposto em (GEROSA et al., 2004). Passou-se, também, mais recentemente, a investir em ferramentas para automatizar alguns aspectos envolvidos na construção dos ambientes em questão, surgindo na engenharia de software a noção de *framework*, para garantir extensibilidade e flexibilidade destes ambientes.

[1] do inglês: Interactive/Intelligent Learning Environment - ILE

Além disso, tais ambientes devem proporcionar recursos educacionais capazes de se adaptar às mudanças cognitivas dos estudantes. Assim, os *software* educacionais devem ter um projeto de *software* adequado, bem como uma metodologia de projeto educacional bem definida (TURANI; CALVO; GOODYEAR, 2005).

Diante disto, torna-se necessário o investimento em ferramentas de autoria, proporcionando ao usuário que não possui experiência em programação (autor) poder construir aplicações de *software* instrucional de forma ágil.

Atualmente, observa-se a chegada de uma nova geração de ambientes computacionais de suporte à aprendizagem colaborativa dotada de recursos ainda mais sofisticados e, conseqüentemente, trazendo mais complexidade no seu desenvolvimento. Estes ambientes, em geral, adotam, para lidar com tal complexidade, o paradigma de agentes inteligentes (PLEKHANOVA, 2002). A construção de sistemas baseados em agentes inteligentes/conhecimento ainda se constitui numa tarefa complexa, devido, principalmente, à falta de recursos, oferecendo facilidades aos seus desenvolvedores, como o suporte a técnicas de Inteligência Artificial.

1.1 Problemática

Ao longo das gerações de sistemas educacionais mediados por computador, diversos problemas foram identificados, levando determinados pesquisadores a focalizarem aspectos relevantes na construção e manutenção destes sistemas.

O desenvolvimento de um ambiente interativo de aprendizagem, normalmente, envolve problemas fundamentais localizados principalmente nas áreas de educação, inteligência artificial em educação (IA-ED) e engenharia de software. Em IA-ED, incluem-se questões que vão da definição de teoria de aprendizagem até a elaboração de componentes para tornar efetiva as interações pedagógicas entre o sistema e o aprendiz. Nesse sentido, destaca-se a proposição de resolvedores de problemas mais efetivos, modelo e representação do conhecimento, ferramentas de gerenciamento do desempenho do aluno, entre outras.

Uma vez dispondo de um modelo, surgem questões relacionadas à construção de sistemas, tratando, principalmente, de como efetivamente construir e manter tais ambientes. Aqui surgem problemas ligados tanto ao trabalho do autor (para garantir a rápida construção de

ambientes interativos de aprendizagem) quanto do desenvolvedor (provendo funcionalidades de extensibilidade para poder adicionar novas funcionalidades).

1.2 Objetivos

O objetivo deste trabalho é desenvolver uma plataforma para construção de ambientes interativos de aprendizagem baseados em agentes. Esta plataforma busca suprir necessidades de engenharia de software e inteligência artificial voltadas a ambientes interativos de aprendizagem, inspirados no modelo Mathema. Os requisitos são:

- suporte à interação entre os três papéis fundamentais, estudante/professor/agente tutor.
- suporte a mecanismos de inferências, através do projeto e implementação de algoritmos de Inteligência Artificial em componentes de software;
- suporte ao compartilhamento e padronização do conhecimento, através da utilização de ontologias;
- suporte ao usuário engenheiro de software/programador, através da noção de *framework*, garantindo a extensibilidade e a flexibilidade da plataforma;
- suporte ao usuário que não possui experiência em programação (autor/não programador), através da noção de autoria, permitindo a construção de *software* instrucionais de forma ágil;
- suporte ao usuário que possui experiência com conceitos de Inteligência Artificial, através da noção de *shell*, podendo configurar algoritmos de IA de forma ágil;
- Ambiente Educacional baseado na WEB, podendo evoluir o ambiente aos paradigmas da Web 2.0 (ALEXANDER, 2006) e Web 3.0 (DACONTA; OBRST; SMITH, 2003);

Com isso, a plataforma utilizará de agentes inteligentes objetivando a tutoria do estudante, tendo na implementação da plataforma aspectos de autonomia, inteligência, mobilidade, pró-atividade, reatividade, resolução de problemas fazendo uso de algoritmos de IA, utilização de diversos modelos mentais do estudante, como cognitivo, afetivo e motivacional, entre outros.

1.3 Contribuições

Desta forma, as contribuições deste trabalho encontram-se na construção de uma plataforma que auxilie *i)* os engenheiros de software/desenvolvedores através de um framework e *ii)* os usuários não programadores permitindo a configuração do domínio de ensino, através de uma ferramenta de autoria.

1.4 Organização

O trabalho está organizado da seguinte forma. No capítulo 2, encontra-se o embasamento teórico sobre os três aspectos basilares. Os trabalhos relacionados são apresentados no Capítulo 3. No Capítulo 4 é apresentada a proposta do trabalho, abordando a plataforma de software, seguido de detalhes nas seções associadas. Objetivando validar a Plataforma, estudos de casos são abordados no capítulo 5. Finalmente, discussão sobre as contribuições e as limitações da plataforma proposta, além de sugestões de trabalhos futuros, são apresentados no Capítulo 6.

Capítulo 2

Embasamento Teórico

Neste capítulo são apresentados os principais fundamentos relacionados ao uso das abordagens utilizadas na plataforma proposta. Isso inclui também as tecnologias utilizadas ao longo do desenvolvimento da plataforma que foram: ontologias, agentes inteligentes e mecanismos de inferência.

2.1 Agentes Inteligentes

O uso de agentes tem sido bastante explorado pela comunidade de Inteligência Artificial. Segundo Russel & Norvig (2004), um agente é uma entidade que possui a capacidade de perceber um ambiente, através de sensores, e agir no ambiente, através de atuadores. Uma arquitetura abstrata de um agente é mostrada na Figura 2.1

A abordagem de agentes é normalmente motivada pelas seguintes características:

- Racionalidade: os dados de entrada do agente equivalem a informações que são tratadas de forma correta, não necessitando de interferências de terceiros. Além disso, a racionalidade pode ser medida através de fatores:

    1. Medida de desempenho do agente;
    2. Conhecimento do agente sobre o ambiente;
    3. Capacidade de atuação dos agentes, ou seja, pertinência das ações de cada agente;
    4. Percepções dos agentes com relação ao ambiente que está sendo observado.

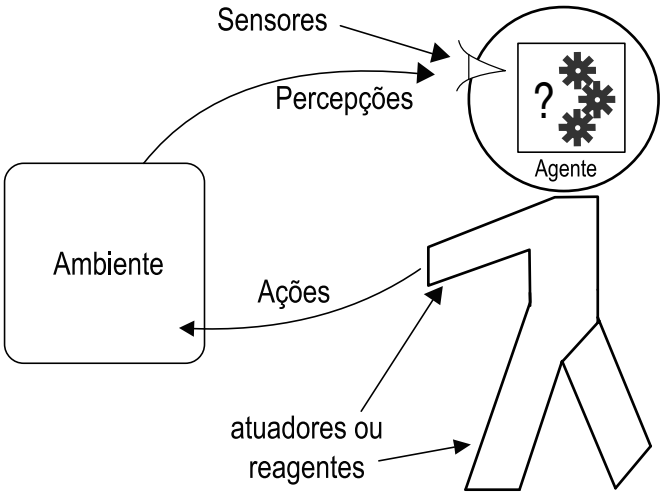

Figura 2.1: Arquitetura Abstrata de Agentes
Fonte: Extraída e adaptada de (RUSSEL; NORVIG, 2004).

- Aprendizado: capacidade do agente de aprender em suas interações;
- Autonomia: característica essencial para garantir o ciclo de vida de um agente, onde as percepções e ações dos agentes garantem sua capacidade de autonomia.

Os agentes podem ser caracterizados de diversas formas (agente reativo, baseado em objetivos, aprendizagem, etc), porém, a abordagem de agentes inteligentes tem sido também, explorada pela comunidade de inteligência artificial em educação. Tais agentes fazem uso de sistemas baseados em conhecimento para poder garantir sua inteligência no processo de resolução de problemas.

Porém, a complexidade envolvida na resolução de problemas motiva uma abordagem multiagente, onde, além das características anteriormente citadas, os agentes geralmente possuem capacidade de:

- Sociabilidade: um agente tem que ter a capacidade de interação social com outros agentes;
- Cooperação[1]: um agente pode ajudar no processo de resolução de um problema;
- Colaboração: o agente pode colaborar em alguma atividade, coordenando tal interação inter-agentes.

[1] Agentes cooperativos trabalham buscando alcançar objetivos comuns (LESSER, 1999).

2.2 Mecanismos de Inferência

O mecanismo de inferência é responsável por raciocinar baseado em um determinado conhecimento representado. Aqui são discutidos os mecanismos de inferências baseados em regras e casos.

Raciocínio Baseado em Regras

Os sistemas baseados em regras, como o próprio nome sugere, possuem o conhecimento descrito na forma de regras. Estas regras são utilizadas pelos sistemas especialistas, onde, através deste conhecimento, objetiva-se a resolução de problemas e a tomada de decisão, pretensamente com a qualidade comparável a um especialista humano. Sendo assim, o sistema precisa ter uma base de regras consistente armazenada em uma base de conhecimento, a qual inclui também um conjunto de fatos adquiridos para se chegar à conclusão do problema e tomada de decisão.

Na Figura 2.2 apresenta-se uma arquitetura detalhada de um sistema baseado em regras.

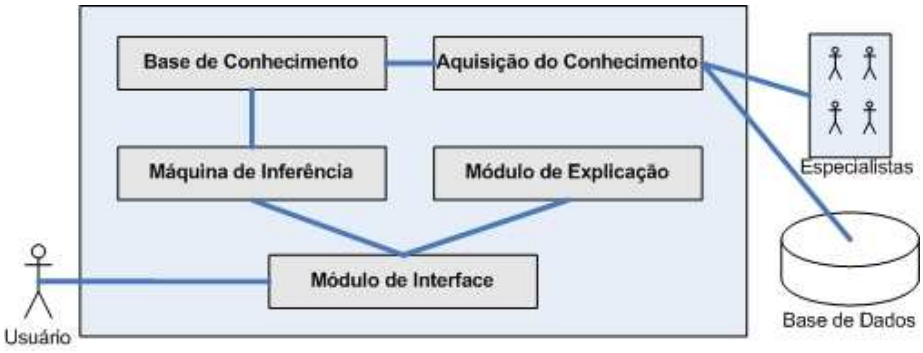

Figura 2.2: Arquitetura simplificada de RBR

Segue-se uma explicação sobre os aspectos presentes na Figura 2.2

- Engenheiro do Conhecimento: profissional que desempenha o papel de capturar o conhecimento especializado em regras e inserir na base de conhecimento;
- Base de Regras: conhecimento armazenado sobre um domínio particular, representado por uma coleção de regras. Essas regras são formadas pela relação “se-então” (if-then), comparável ao que se verifica em uma típica linguagem de programação tradicional;

- Máquina de Inferência: parte do raciocínio baseado em regras que seleciona e executa regras, resultando em uma resposta ao usuário, de acordo com os resultados que foram inferidos ao longo da execução das regras;

- Explanação: depois da conclusão da inferência ser mostrada ao usuário, pode-se haver necessidade de uma explanação acerca de "por que" (uma questão posta pelo sistema) e "como" (how) se chegou àquele resultado, e isso é responsabilidade do subsistema de explanação;

- Memória de Trabalho: local onde se encontram os fatos utilizados pelo mecanismo de inferência do sistema;

- Aquisição de Conhecimento: este subsistema permite o fluxo do novo conhecimento de um especialista humano para a base de conhecimento;

Raciocínio Baseado em Casos

O Raciocínio Baseado em Casos (RBC) teve início com os estudos de Schank 1982 sobre a compreensão da linguagem. Segundo sua teoria, a linguagem é um processo baseado em memória. Seus pressupostos teóricos afirmam que em cada experiência do ser humano, sua memória passa a se ajustar em resposta a essas experiências, implicando dessa forma, que o aprendizado depende dessas alterações na memória. Através desse estudo, conclui-se que: o processo de compreensão de uma linguagem depende de informações armazenadas previamente na memória, ou seja, pessoas não compreendem eventos sem deixar de fazer referências a fatos que já vivenciaram e que já conhecem (SILVA, 2005).

O RBC é justamente uma abordagem capaz de possibilitar que algum novo problema enfrentado por certo usuário seja confrontado com uma base de conhecimentos, formada por um conjunto de situações prévias - especialmente organizadas denominada base de casos. Qualquer sistema cuja modelagem esteja fundamentado nessa abordagem pode ser imaginado como sendo um sofisticado sistema de casamento de padrões (BITTENCOURT et al., 2006a).

A representação esquemática das operações que a comunidade de RBC tem utilizado denomina-se de algoritmo dos 4-R. O algoritmo possui este nome devido às fases presentes

no ciclo, sendo elas: (1ºR) recuperar caso(s); (2ºR) reusar o(s) caso(s); (3ºR) revisar a(s) solução(ões); (4ºR) reter solução(ões). Este ciclo utilizando RBC é ilustrado na Figura 2.3.

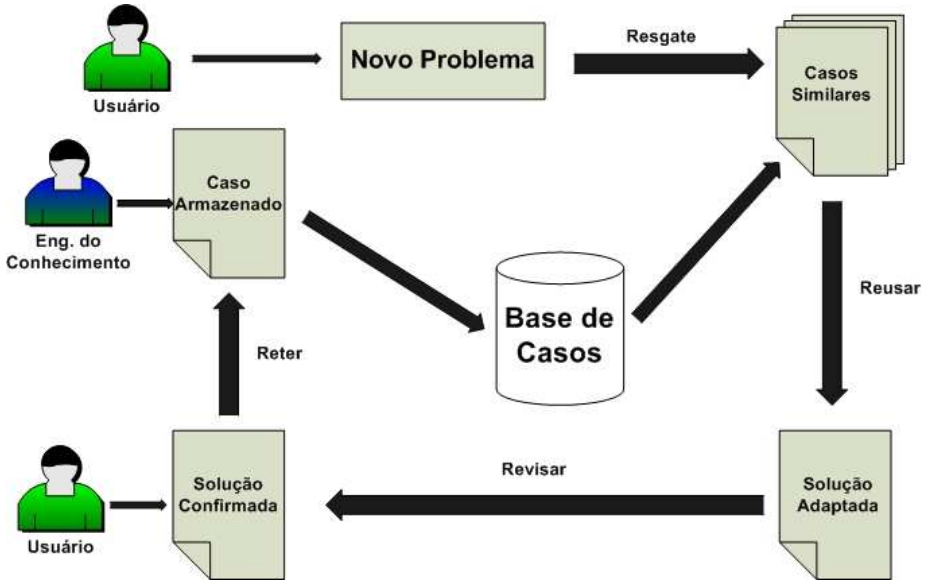

Figura 2.3: Representação esquemática do algoritmo dos 4-R

Com isso, existem quatro fases no processo de inferência em RBC, sendo elas:

1R - Resgate: objetiva achar um caso ou um conjunto de casos similares, provendo solução do problema atual. Algumas formas de recuperação de casos podem ser seqüenciais, em dois níveis, entre outras. Outro aspecto importante nesta fase é a similaridade, da qual existem dois tipos. Primeiro, similaridade local, sendo feita para descobrir a similaridade entre atributos segundo, similaridade global, para a similaridade entre casos;

2R - Reuso: objetiva fazer o reuso do caso da base de casos para ser utilizado para a solução de um problema. Nesta fase, ocorre a adaptação do caso, quando há a necessidade. Algumas formas de adaptação que podemos citar são: composicional, hierárquica, geracional, entre outras. Esta fase é bastante complicada de prever e generalizar, pois é muito específica de cada domínio;

3R - Revisão: objetiva confirmar uma solução reusada. O processo de revisão pode ser feito de forma automática ou automatizada. Esta fase é bastante complicada e, em alguns casos, pode até haver outra forma de inferência dentro desta fase, apenas para validar;

4R - Retenção: é a fase onde ocorre a aprendizagem do RBC, pois depois dos 3Rs, o novo caso é armazenado na base de casos e depois é retornada à solução para o usuário.

2.2.1 Ontologia

Do ponto de vista computacional, uma ontologia é uma especificação formal e explícita de uma conceitualização compartilhada (GRUBER, 1993). Para melhor entendimento, abaixo segue detalhes sobre os termos citados:

- Explícita: definições de conceitos, relações, restrições e axiomas;
- Formal: compreensível para agentes e sistemas;
- Conceitualização: modelo abstrato de uma área de conhecimento;
- Compartilhada: conhecimento consensual;

As ontologias fornecem um vocabulário para a representação do conhecimento. Sendo assim, caso exista uma ontologia que modele adequadamente certo domínio de conhecimento, essa pode ser compartilhada e usada por pessoas que desenvolvam aplicações dentro desse domínio. Além disso, algumas vantagens em se utilizar ontologias são:

- Entendimento comum de uma estrutura de informação; entre pessoas e agentes de software;
- Reúso do conhecimento do domínio;
- Tornar o domínio explícito;
- Separar o conhecimento do domínio do conhecimento operacional;
- Analisar o conhecimento do domínio.

Além disso, a construção de uma ontologia pode ser dividida em diversas categorias, dentre as quais, citam-se:

- Ontologias de Representação: definem as primitivas de representação (frames, atributos, facetas,...);

- Ontologias Gerais: trazem definições abstratas necessárias para a compreensão de aspectos do mundo (tempo, espaço, seres, coisas,...);
- Ontologias Centrais ou genéricas de domínio: definem os ramos de estudo de uma área e/ou conceitos mais genéricos e abstratos desta área;
- Ontologias de Domínio/Tarefas: tratam de um domínio mais específico de uma área genérica de conhecimento, como direito penal;
- Ontologias de Aplicação/Domínio: procuram solucionar um problema específico de um domínio;

Nos atuais sistemas educacionais, a idéia de ontologia tem sido aplicada para a representação do conhecimento, além de outras características como maior expressividade do conhecimento e capacidade de inferência.

O uso de ontologias é motivado pelos diversos recursos providos, possibilitando solucionar os problemas atualmente enfrentados por pesquisadores da área em questão (MIZOGUCHI; BOURDEAU, 2000), como:

- Semântica Computacional: uma ontologia provê uma coleção estruturada de termos, definições formais para prevenir interpretações errôneas de conceitos, restrições formalmente definidas através de axiomas, entre outros;
- Ontologias garantem o compartilhamento, padronização e reusabilidade do conhecimento;
- Ajudam a especificar um meta-nível do conhecimento.

2.3 Framework

Esta seção descreve conceitos referentes à construção de *Frameworks* de software. Várias definições podem ser encontradas na literatura, não existindo assim uma definição universal. Uma definição bastante referenciada de *framework* é fornecida em Johnson (JOHNSON; FOOTE, 1988).

> "Um *framework* é um conjunto de classes que incluem um projeto abstrato para soluções de uma família de problemas relacionados, e suporta reúso em maior granularidade do que classes".

Além disso, no meio empresarial, Taligent (TALIGENT, 1994) foi a principal empresa a investir nesse conceito no inicio da década de 90, definindo framework como:

> "Um *framework* define um comportamento de uma coleção de objetos, fornecendo um modo inovador para reusar código e projeto de *software*".

Alguns aspectos comuns aos *frameworks*, como foram destacados nas definições acima, são:

- É formado por um conjunto de classes;
- Reúsa o projeto de software e não apenas o código;
- Não é uma aplicação, e sim, um arcabouço para construir aplicações num domínio específico.

Numa outra definição, Johnson (JOHNSON, 1997) afirma que um *framework* é um esqueleto de uma aplicação que pode ser personalizado pelo desenvolvedor.

Esta definição considera um *framework* como uma aplicação semi-completa que pode ser especializada para produzir aplicações personalizadas. A Figura 2.4 ilustra uma aplicação desenvolvida utilizando-se um *framework* existente. A parte sombreada corresponde ao *framework* - uma estrutura de classes que é reutilizada - e a parte tracejada, corresponde as classes que são inseridas pelo usuário do *framework* para o desenvolvido de uma aplicação específica.

Os *frameworks* possuem algumas características importantes (FAYAD; SCHMIDT; JOHNSON, 1999) que os diferenciam de outras formas de reúso. Estas características são descritas a seguir:

- **Modularidade**: é obtida encapsulando detalhes de implementação de alguns trechos de código em interfaces bem definidas. Isto melhora a qualidade, minimizando o impacto das mudanças no projeto e implementação do *software*;

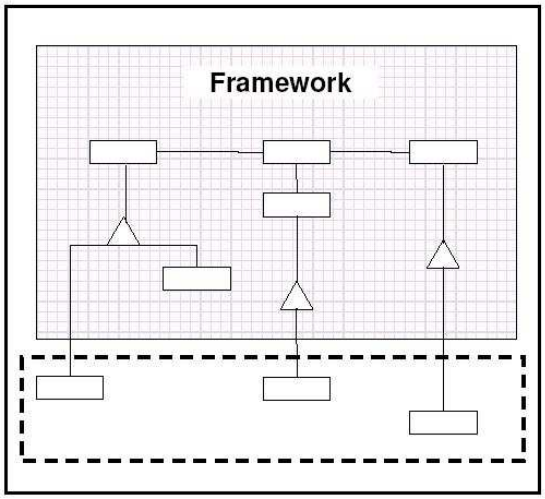

Figura 2.4: Aplicação desenvolvida utilizando um *framework*
Fonte: Adaptada de (SIVA; PRICE, 1997).

- **Reusabilidade**: esta característica é garantida definindo componentes genéricos que podem ser utilizados na criação de novas aplicações. O desenvolvedor que utiliza o *framework* pode aproveitar a análise do conhecimento do domínio e esforços anteriores de desenvolvedores experientes. Adicionalmente, algumas melhorias como o aumento da produtividade dos programadores, da qualidade e da confiabilidade dos programas gerados são também atribuído ao reúso do *framework*;

- **Inversão de controle**: equivale a uma característica fundamental na arquitetura de um *framework*. Ela permite determinar que conjunto de métodos de uma aplicação específica deve ser chamado pelo *framework*. Esta característica é baseada no princípio de *Hollywood*[2];

- **Extensibilidade**: a extensibilidade do *framework* é obtida através de métodos *hooks* (ganchos ou adaptações) que permitem que aplicações estendam ou implementem as interfaces do *framework*. Desta forma, inúmeras aplicações podem ser instanciadas com base no mesmo *framework*.

[2] *Don't call us, we'll call you.*

2.3.1 Vantagens e Desvantagens de um *Framework*

Os benefícios obtidos do desenvolvimento não são necessariamente imediatos (TALIGENT, 1994), porém, as principais vantagens em se utilizar *frameworks* podem ser resumidas como sendo (COTTER; TALIGENT, 1995):

- **Menos código para projetar e escrever**: a maior parte do projeto, estrutura e código já estão prontos para serem reutilizados no *framework*. O desenvolvedor da aplicação necessita apenas sobrepor o comportamento padrão do *framework*;
- **Código mais confiável e robusto**: o Código herdado do *framework* já tem sido testado e integrado com o restante do *framework*. Uma boa documentação e um conjunto de testes devem ser fornecidos junto com o *framework*;
- **Código mais consistente e modular**: o reúso do código fornece consistência e classes extensas são divididas em pequenos pedaços funcionais, tornado assim o código mais modular e reduzindo o esforço de manutenção;
- **Maior foco na área de perícia**: os desenvolvedores do *framework* geralmente são especialistas no domínio e esta habilidade pode ser passada para o desenvolvedor da aplicação, bastando para isto, o mesmo utilizar o *framework*. Portanto, o desenvolvedor da aplicação pode se concentrar especificamente nas características específicas de sua aplicação, melhorando assim a sua produtividade;
- **Menor tempo de desenvolvimento**: como o domínio do problema não precisa ser analisado novamente, o *framework* fornece componentes que podem ser utilizados na aplicação. Os usuários familiares com o *framework* desenvolvem novas aplicações em menor tempo quando comparado ao desenvolvimento sem o *framework*. Embora, um tempo gasto no aprendizado seja necessário antes de utilizar o *framework*;
- **Manutenção reduzida e evolução facilitada**: modificações feitas no *framework*, tais como, adicionar uma nova característica ou corrigir um erro, são automaticamente atualizadas nas aplicações construídas com ele. Levando-se em consideração que aplicações são desenvolvidas utilizando o *framework* como ponto de partida, e estes utilizam interfaces estáveis na sua construção. Pouco código é escrito para modificar ou estender o comportamento do *framework*(geralmente através de métodos *hooks*).

Embora *frameworks* aumentem a qualidade e diminuam o tempo de desenvolvimento de uma aplicação, eles devem ser projetados com reúso em mente. Algumas desvantagens e dificuldades são encontradas tais como:

- **Requer mais esforço para construir e aprender** (TALIGENT, 1994): uma vez que o desenvolvimento de sistemas complexos é difícil, o desenvolvimento destes sistemas, de forma abstrata, tendo em mente reutilização, é mais difícil ainda (FAYAD; SCHMIDT; JOHNSON, 1999). Além disto, a experiência no domínio de aplicação leva tempo para ser obtida. Na visão do desenvolvedor da aplicação, leva tempo para aprender o *framework* e então obter as vantagens fornecidas pelo mesmo;

- **Integrabilidade** (FAYAD; SCHMIDT; JOHNSON, 1999): a maior parte dos *frameworks*(GUIs, comunicação e banco de dados) são desenvolvidos exclusivamente para o propósito de extensão e não integração com outros *frameworks* externos. Portanto, problemas de fluxo de controle, interoperabilidade e integração de linguagens entre *frameworks* são difíceis de resolver durante o processo de integração;

- **Requer documentação, manutenção e suporte** (TALIGENT, 1994) : como *frameworks* são mais abstratos do que a maioria dos softwares, documentá-los torna-se mais difícil (JOHNSON, 1992). Porém, uma boa documentação é necessária para melhorar o entendimento e aprendizagem de novos *frameworks*. Além da documentação, uma forma de suporte e manutenção é necessária para que o mesmo se torne adaptável à um número cada vez maior de aplicações.

Mesmo com as desvantagens citadas, *frameworks* são importantes em vários domínios, principalmente em grandes aplicações orientadas a objetos (MATTSSON, 1996). Além disso, diversos tipos de frameworks comerciais (**??**) abrangendo os mais variados domínios, utilizam os benefícios vistos e tendem a minimizar as desvantagens explicadas nesta seção.

2.3.2 Classificação dos *Frameworks*

Várias classificações de *frameworks* têm sido propostas na literatura. Apresentamos aqui as duas principais visões de *frameworks*. Um *framework* pode ser classificado por seu escopo (domínio da aplicação) e também pela técnica utilizada para estendê-lo.

Quanto ao escopo, um *framework* pode ser classificado em relação as funções de aplicação, domínio e suporte (TALIGENT, 1994):

- ***frameworks* de aplicação**; são também conhecidos como *frameworks* horizontais. Esses fornecem funcionalidades que são aplicadas a uma variedade de problemas. São utilizados em mais de um domínio. Exemplos clássicos são os *frameworks* utilizados para construção de interfaces gráficas;

- ***frameworks* de domínio**: são também conhecidos como *frameworks* verticais. Esses fornecem funcionalidades que são aplicadas num domínio específico. Esses *frameworks* são direcionados geralmente para amplos domínios de aplicação como telecomunicações, multimídia, saúde e finanças;

- ***frameworks* de suporte**: esses *frameworks* fornecem serviços de suporte básicos ao nível de sistema. Exemplos são *frameworks* de acesso a arquivos e de comunicação para computação distribuída.

Quanto a técnica de extensão fornecida, os frameworks podem ser classificados em *whitebox*, *blackbox* e *graybox* (FAYAD; SCHMIDT; JOHNSON, 1999), como mostrado na Figura 2.5.

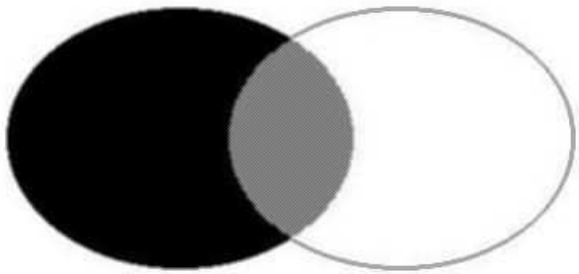

Figura 2.5: Classificação de *frameworks* de acordo com a técnica de extensão
Fonte: Adaptada de (AGUIAR, 2003).

- ***whitebox frameworks***: são também chamados de *frameworks* dirigidos a arquiteturas (*architecture driven frameworks*) (TALIGENT, 1994). São chamados de *whitebox*

porque sua implementação ou estrutura interna precisa ser entendida antes de ser utilizada. Esses *frameworks* confiam fortemente em herança para realizar customização (extensibilidade). Os clientes podem customizar o *framework*:

1. derivando novas classes do *framework*, herdando de suas classes base e
2. sobrepondo suas funções membros.

Nesses *frameworks*, devido a herança, as aplicações são fortemente acopladas aos detalhes da hierarquia do *framework*;

- ***blackbox frameworks***: são também chamados de *frameworks* dirigidos a dados (*data-driven frameworks*) (TALIGENT, 1994). A sua estrutura interna não precisa ser entendida para ser utilizada. Esses *frameworks* confiam principalmente em composição (classes ou componentes) e delegação para realizar customização. Os clientes podem customizar o *framework*:

  1. Definindo as interfaces dos componentes que serão ”plugados“ no *framework* e
  2. Integrando esses componentes no *framework* (geralmente utilizando padrões de projeto), mas o *framework* define como os componentes podem ser combinados.

  Esses *frameworks* são geralmente mais fáceis de utilizar e estender do que *whitebox frameworks* (FAYAD; SCHMIDT; JOHNSON, 1999). Porém, são mais difíceis de construir, devido a definição das interfaces e métodos *hooks* que devem se antecipar a extensa variedade dos requisitos das aplicações (AGUIAR, 2003);

- ***graybox frameworks***: são *frameworks* projetados para evitar as desvantagens apresentadas por *whitebox* e *blackbox frameworks*, permitindo um certo grau de flexibilidade e extensibilidade sem a necessidade de expor informações internas desnecessárias. Esses *frameworks* são customizados utilizando-se herança e composição.

Independente da taxonomia utilizada para classificar um *framework*, quando utilizado em conjunto com bibliotecas de classes, padrões de projeto, componentes e arquitetura de *software*, *frameworks* podem aumentar significativamente a qualidade do *software* e reduzir o esforço e custo de desenvolvimento.

2.4 Ambientes Interativos de Aprendizagem

A concepção dos Ambientes Interativos de Aprendizagem refere-se à evolução dos sistemas tutores inteligentes para os ambientes de aprendizagem cooperativos[3]. O processo de aprendizagem nestes ambientes ocorre através da interação com o ambiente. Além disso, tais ambientes podem maximizar o aprendizado do aluno através das interações entre os estudantes, isto é, compartilhando conhecimento, através de ferramentas colaborativas[4].

Ambientes Interativos de Aprendizagem mediados por computador envolvem, principalmente, aspectos das áreas de Educação, Computação (particularmente a Inteligência Artificial) e Psicologia Cognitiva (COSTA, 1997). Este domínio trata a complexidade concernente a pluridisciplinaridade referentes a sistemas educacionais.

O aprendizado nestes ambientes, facilitado por aspectos de cooperação e colaboração, remete a uma triangulação entre os três papéis fundamentais para tais ambientes, como mostrado na Figura 2.6.

Figura 2.6: Triangulação abordando a interação entre os papéis

A Figura 2.6 aborda que através de ambientes interativos de aprendizagem, é possível ter interação entre os papéis do estudante, tutor e professor.

[3] Área também conhecida na literatura como *Interactive Learning Environment (ILE).*

[4] Através da área denominada de *Computer supported collaborative learning (CSCL)* ou aprendizagem colaborativa apoiada por computador.

2.4.1 Modelo MATHEMA: Conceitos, Adaptações e Integrações

Esta subseção aborda características do modelo MATHEMA para AIA baseado em Agentes, proposto por Evandro Costa em sua tese de Doutorado (COSTA, 1997). Algumas adaptações foram feitas no modelo, principalmente levando-o para a Web e dando subsídios as especificações do *framework*. A seguir, são abordados os aspectos conceituais do modelo MATHEMA, seguidos das adaptações e integrações necessárias.

Modelo Mathema

O Modelo MATHEMA foi utilizado como uma das bases conceituais deste trabalho, principalmente por apresentar um modelo de ambiente interativo de aprendizagem baseado numa arquitetura multiagentes, sendo um representante adequado e em conformidade com as concepções de tais ambientes de suporte ao aprendizado mediado por computador.

A especificação do modelo mínimo de aprendizagem do MATHEMA equivale à idéia de um jogo. O jogo ocorre quando os dois jogadores (Estudante e Sistema Tutor) iniciam uma interação através da troca de mensagens, como mostrado na Figura 2.7, visando alcançar os critérios de satisfação de ambas as partes.

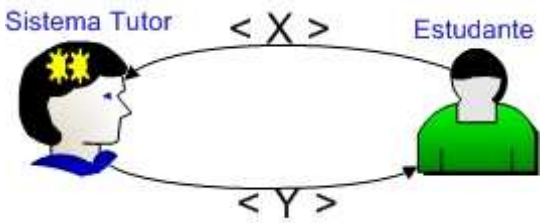

Figura 2.7: Modelo Mínimo do MATHEMA
Fonte: Extraída e adaptada de (COSTA, 1997).

De uma maneira mais abstrata, o processo de interação é visto como ocorrendo dentro de um esquema de troca de mensagens (X, Y) entre as duas entidades envolvidas, obedecendo a certo protocolo (COSTA, 1997). O grau de aprendizagem do estudante se dará através de ações sobre o conteúdo das mensagens. Dentre as exigências necessárias para garantir a efetividade no processo de interação, torna-se necessário, do lado do sistema tutor, suporte aos seguintes aspectos:

- conhecimento sobre o domínio;

- conhecimento pedagógico;
- conhecimento sobre o estudante;
- capacidade de interação.

As necessidades supracitadas remetem primeiramente à demanda por uma modelagem do conhecimento através de uma visão dimensional (visão externa) sobre este domínio, a qual ajudará o seu posterior particionamento (conduzindo a uma visão interna). A visão externa equivale a uma interpretação de um dado domínio de conhecimento, enquanto a visão interna equivale a um particionamento do domínio. Tal investimento de modelagem conduziu a uma visão tridimensional associada ao domínio, como mostrado na Figura 2.8, onde as dimensões são (COSTA; PERKUSICH; FERNEDA, 1998):

1. Contexto: que equivale a um ponto de vista sobre um domínio de conhecimento, ou seja, cada contexto diz respeito a um domínio;
2. Profundidade: é definida relativamente a um dado contexto, sendo concernente a alguma forma de refinamento na linguagem de percepção, ou seja, a estratificação dos vários níveis epistemológicos de percepção do objeto de conhecimento em apreciação;
3. Lateralidade: diz respeito aos conhecimentos afins de suporte a um dado objeto de conhecimento do domínio alvo, proveniente duma visão particular de contexto e profundidade, constituindo-se, no escopo do presente trabalho, em pré-requisitos da referida unidade.

Cada partição de D origina subdomínios, os quais passam a ser mapeados em estruturas curriculares. Uma visão interna do currículo é composta por unidades pedagógicas (*up*), da forma

$$Curric = \{up_1, up_2, \ldots, up_n\}, \tag{2.1}$$

onde $Curric$ denota um currículo, considerando que cada up_i denota uma unidade pedagógica do $Curric$. Estas unidades estão relacionadas segundo uma ordem definida com base

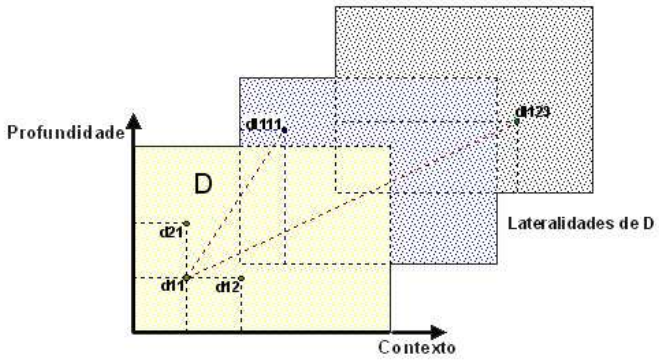

Figura 2.8: Visão tridimensional do modelo do domínio no MATHEMA

Fonte: Extraída de (COSTA, 1997).

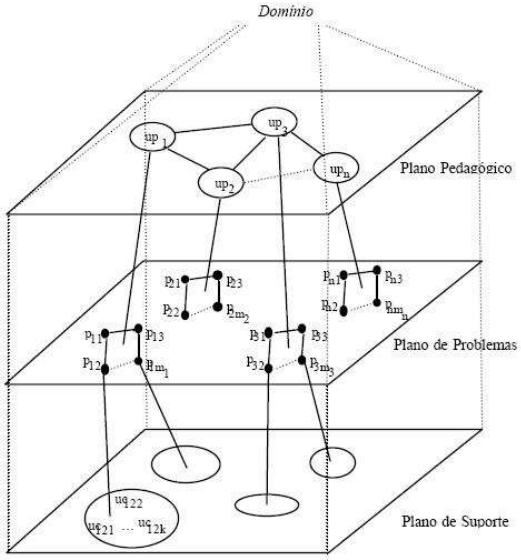

Figura 2.9: Estrutura pedagógica do conhecimento do domínio no MATHEMA

Fonte: Extraída de (COSTA, 1997).

em critérios pedagógicos. A cada up_i corresponde um conjunto de problemas. Cada problema está associado a um conhecimento de suporte à sua resolução, incluindo basicamente conceitos e resultados, como mostrados na Figura 2.9.

Cada subdomínio será representado por dois tipos de agente tutor (AT), um equivalendo ao currículo do subdomínio e outro pela lateralidade/pré-requisito do currículo (ATL). Com isso, o conjunto de todos os agentes do domínio forma uma sociedade de agentes tutores autônomos, sendo:

$$SATA = AT \cup ATL. \tag{2.2}$$

A inserção dos agentes faz com que a interação presente no jogo (Figura 2.7) evolua como mostrada na Figura 2.10

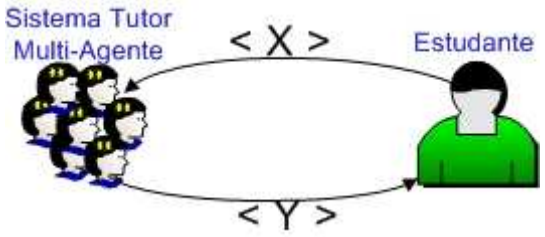

Figura 2.10: Modelo de Ambiente de Aprendizagem baseado em agentes no MATHEMA

Fonte: Extraída e adaptada de (COSTA, 1997).

Além do estudante e do sistema tutor multiagente, dois aspectos são fundamentalmente importantes para o processo de aprendizagem, sendo eles: *i)* um mediador extra-ambiente para interação com estudantes, e *ii)* a manutenção do conhecimento curricular presente nos agentes tutores por especialistas humanos. Tais necessidades evoluem para a arquitetura do MATHEMA (Figura 2.11).

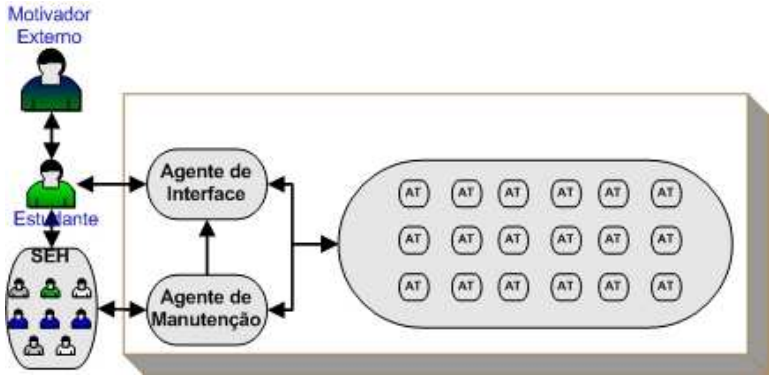

Figura 2.11: Arquitetura do MATHEMA

Fonte: Extraída e adaptada de (COSTA, 1997).

Ressalta-se ainda que a especificação interna do agente tutor é um aspecto elementar para possibilitar a interação estudante-tutor. Internamente, este agente é formado por três camadas (vide Figura 2.12), sendo elas:

1. Sistema Tutor: responsável pelas atividades de tutoria no processo de interação estudante-tutor. O resolvedor de tarefas representa as unidades pedagógicas no processo de resolução de problemas;

2. Sistema Social: possibilita ao Sistema Tutor interagir com outros agentes no processo de resolução de problemas. Além disto, é nesta camada que a manutenção dos agentes ocorre;

3. Sistemas de Distribuição: responsável pela manipulação das mensagens recebidas e enviadas.

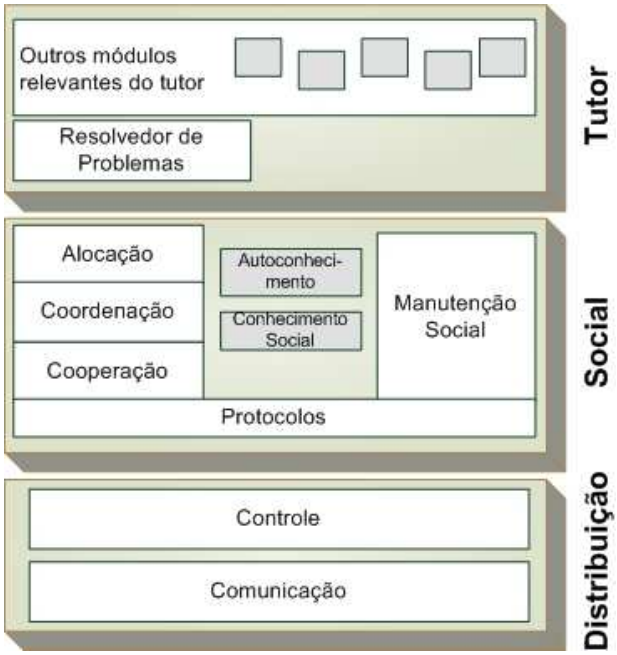

Figura 2.12: Arquitetura de um Agente Tutor no MATHEMA: visão interna

Fonte: Extraída e adaptada de (COSTA, 1997).

Com a especificação do modelo MATHEMA definida, serão abordados a seguir as adaptações e integrações necessárias para possibilitar a construção do arcabouço nos moldes definidos anteriormente.

2.4.2 Adaptações e Integrações ao Modelo MATHEMA

A primeira adaptação que ocorreu no modelo MATHEMA foi a definição das informações curriculares. Antes, um currículo era associado apenas a um subdomínio relacionado a uma dada partição, porém, duas necessidades foram identificadas: *i)* uma partição do domínio pode ter subpartições do domínio; *ii)* uma lateralidade passa a ser um currículo.

A mudança na lateralidade foi motivada pelo seguinte aspecto: antes, a lateralidade vista como um currículo não era uma partição do domínio. Porém, tal lateralidade, em outro

domínio, equivalerá a uma partição do domínio. Conseqüentemente, esta lateralidade deverá ser mapeada em um currículo, no outro domínio.

A segunda mudança ocorreu na estrutura pedagógica do conhecimento do domínio (Vide Figura 2.9). Dois aspectos foram adicionados à estrutura: i) uma camada abaixo de unidade do conhecimento, chamada de Plano Comportamental, e ii) uma camada presente em todos os níveis da hierarquia, chamada de Recurso de Aprendizagem.

O Plano Comportamental foi definido através da integração com (DILLENBOURG; SELF, 1992), onde é feita uma separação entre o comportamento conceitual e conhecimento conceitual. O Comportamento conceitual possui uma estrutura de inferência para o processo de resolução de problemas. O Conhecimento conceitual contém descrições conceituais dos problemas, tanto em largura quanto em profundidade.

A camada de Recurso de Aprendizagem foi adicionada para dar suporte, tanto as diversas estratégias pedagógicas quanto ao caráter evolutivo dos recursos de aprendizagem. Com isso, tal mudança faz com que a estrutura pedagógica fique como mostrada na Figura 2.13.

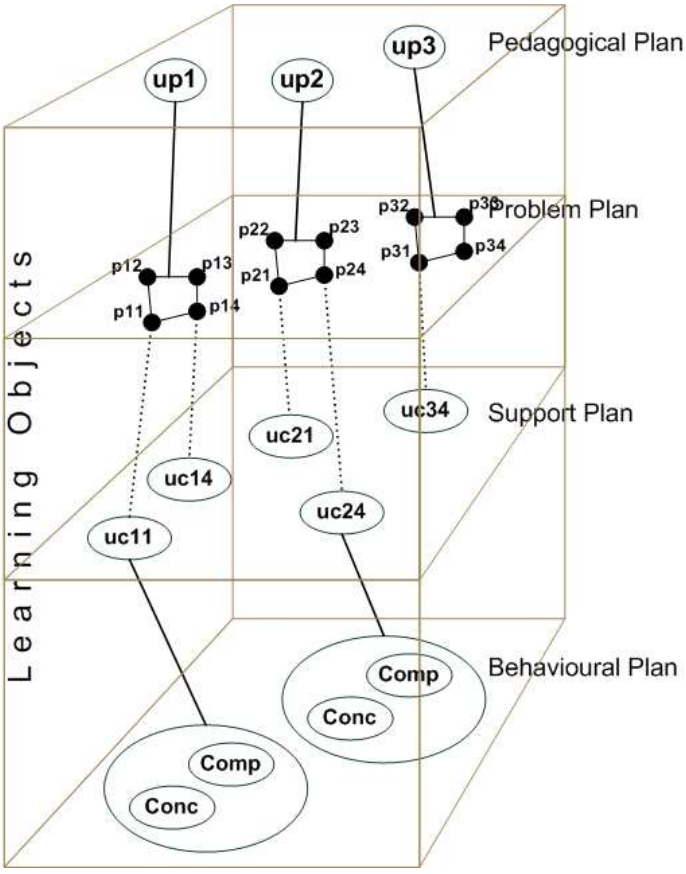

Figura 2.13: Estrutura pedagógica do conhecimento do domínio adaptada

As duas mudanças anteriormente citadas fazem com que haja um efeito cascata, tanto na sociedade de agentes quanto na arquitetura do MATHEMA.

A SATA equivalia à união dos agentes tutores com os agentes tutores de lateralidade (Vide Figura 2.2), porém, a primeira mudança faz com que o agente tutor de lateralidade seja um agente tutor. Além disso, a segunda mudança motiva a utilização de agentes inteligentes para a resolução de problemas. Com isso, a sociedade de agentes é definida como sendo:

$$SA = ATA \cup ADA, \tag{2.3}$$

onde ATA equivale aos agentes tutores autônomos e AS aos agentes de suporte. Além disso, um agente mediador foi adicionado para dar subsídio à sociedade de agentes, para a coordenação do processo de resolução de problemas e manutenção do conhecimento social.

As mudanças arquiteturais ocorridas no MATHEMA, quando impactado aos aspectos supracitados e à adição de conceitos de engenharia de software, remetem à arquitetura de agentes apresentada no Capítulo 4.

Outra mudança ocorreu na arquitetura interna do agente tutor. Anteriormente, um agente tutor possuía três camadas, sendo elas: distribuição, social e tutor (Vide Figura 2.12). A disponibilização de diversas infra-estruturas de agentes atualmente, faz com que a camada de distribuição não seja interna ao agente tutor. Além disso, a camada social sofreu algumas mudanças com a adição do agente mediador. Com isso, a arquitetura do agente tutor é definida na Figura 2.14.

Para finalizar, a arquitetura do agente mediador é definida na Figura 2.15. O agente mediador, além de possuir funcionalidades para dar subsídio à sociedade de agentes, passou a possuir funcionalidades presentes no agente de interface no modelo clássico do MATHEMA.

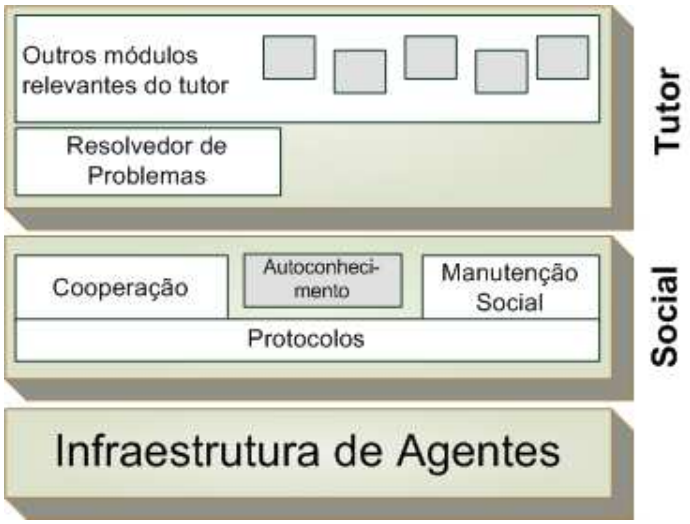

Figura 2.14: Arquitetura de um Agente Tutor adaptada

Figura 2.15: Arquitetura do Agente Mediador

Capítulo 3

Trabalhos Relacionados

De um modo geral, ambientes educacionais têm sido abordados na literatura, já fornecendo evoluções significativas de resultados de pesquisas. A literatura tem revelado ambientes com objetivos específicos, orientados a domínio (como ferramentas destinadas a apoiar o ensino de xadrez (HARTMANN et al., 2005), Direito (ALEVEN, 2003), Matemática (WEBBER; PESTY, 2004; VIRVOU; MOUNDRIDOU, 2000), entre outros e ambientes com objetivos genéricos, de propósitos gerais, buscando independência de domínio. Um importante relato do estágio, relativamente atual da pesquisa pode ser visto em (MURRAY, 1999) e na continuação em uma versão atualizada (MURRAY, 2003).

Em particular, a discussão aqui apresentada considera apenas como relacionados a família dos sistemas tutores inteligentes ou dos ambientes interativos de aprendizagem focalizando aspectos de engenharia de software (como *framework*, autoria e *shell*) e inteligência artificial (como agentes inteligentes e mecanismos de inferência).

Nessa perspectiva, descreve-se e comenta-se aqui um apanhado de trabalhos considerados relacionados à proposta desta dissertação. Assim, trabalhos que investem unicamente em um dos aspectos citados (só de IA ou só de Engenharia de Software) como (GEROSA et al., 2004) e (OTSUKA; ROCHA, 2005) não são descritos a seguir.

Nas seções abaixo, são abordados alguns destes trabalhos. A explanação que é dada sobre tais trabalhos leva em consideração as características e arquitetura presentes em cada ambiente, objetivando facilitar o seu entendimento e fluxo de controle, além de suas vantagens e limitações.

Após a exposição dos trabalhos relacionados a seguir, os mesmos são sendo comparados

através das características acima.

3.1 *I-Help*

I-Help pode ser entendido como um ambiente educacional baseado em agentes que objetiva assistir estudantes no processo de resolução de problemas. O ambiente foi desenvolvido pelo Laboratório de Pesquisas Avançadas em Sistemas Educacionais Inteligentes[1] (AIRES, 1992) da Universidade de *Saskatchewan* (SASK, 1994).

Este ambiente é baseado em uma arquitetura multiagente, consistindo de agentes pessoais (de agentes humanos) e aplicações de agentes (de aplicações de software), que tem como objetivo localizar recursos para os estudantes, como mostrado na Figura 3.1. Tais recursos são localizados a partir do momento em que o estudante solicita determinado recurso.

I-Help possui dois componentes (GREER et al., 2001), sendo eles:

- Discussão Pública: neste componente, o estudante pode postar questões, comentários e respostas em uma ferramenta de fórum;
- Discussão Privada: ocorre quando o estudante solicita uma discussão privada com um ajudante. Neste componente, a interação ocorre da seguinte forma:
    1. o estudante vai entrar em contato com o seu agente pessoal para que ele possa emitir uma solicitação;
    2. o agente pessoal negocia com outros agentes pessoais, objetivando localizar potenciais ajudantes;
    3. uma lista de N estudantes é notificada que há um estudante esperando por ajuda;
    4. a partir do momento que o primeiro estudante aceitar a requisição, uma interação ajudante-ajudado começa;
    5. após a interação ser completada, cada estudante recebe um formulário de avaliação para que eles avaliem o outro estudante.

Algumas vantagens que são identificadas no *I-Help* equivalem ao aspecto de autoria, além de fazer uso de uma abordagem baseada em agentes e ontologia.

[1] Do Inglês: Laboratory for the Advanced Researches in Intelligent Educational Systems.

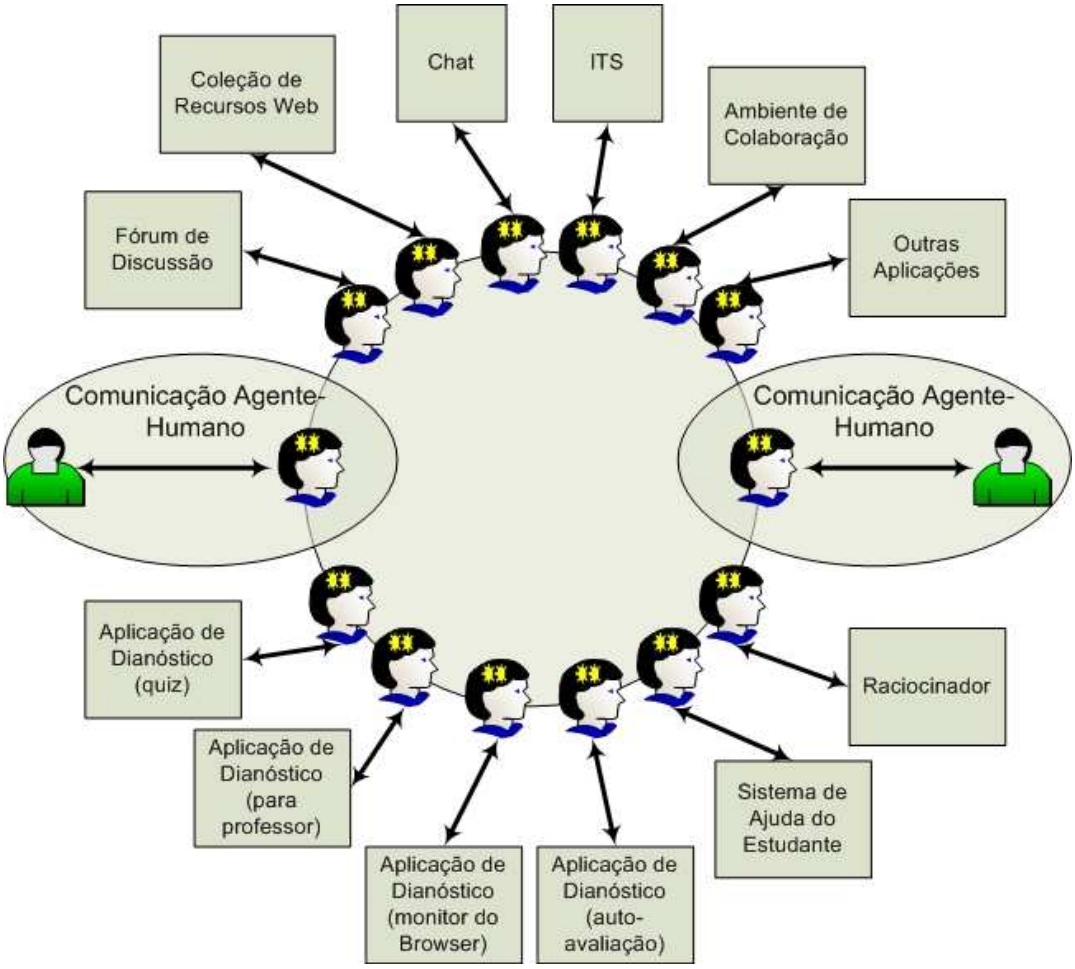

Figura 3.1: Arquitetura do *I-Help*

Fonte: Extraída e adaptada de (VASSILEVA; DETERS, 2001).

Porém, *I-Help* falha no aspecto evolutivo, não dando suporte à extensibilidade. Ressalta-se, também, que o aspecto cooperativo do ambiente é precário, além de não possuir tutoria personalizada.

3.2 *FAmCorA*

FAmCorA é um *Framework* para a Construção de Ambientes Cooperativos de Aprendizagem na *Web*, desenvolvido pela Universidade Federal do Espírito Santo (UFES, 2006).

A construção deste ambiente se deu através da avaliação feita no desenvolvimento de um ambiente de aprendizagem chamado *AmCorA* (MENEZES; CURY; BERNARDINO, 1999). O objetivo do *AmCorA* é servir de plataforma para apoiar comunidades virtuais, que se formam

nas diversas atividades da Universidade Federal do Espírito Santo, e também, em cursos de Educação a Distância envolvendo outras instituições (NETTO; MENEZES; JúNIOR, 2004).

FAmCorA busca contemplar as necessidades da construção de ambientes cooperativos de aprendizagem, através de um conjunto de aplicações provedoras de serviços (Web Services (BARRY, 2003)) e de uma arquitetura que possibilita a comunicação inter-aplicações (PESSOA; NETTO; MENEZES, 2002). Através da avaliação feita sobre o *AmCorA*, foi identificado um conjunto inicial de aplicações que estão contempladas no framework: Provedores de Serviços Básicos, Provedores de Engenho e Provedores de Aplicação, como mostrado na Figura 3.2.

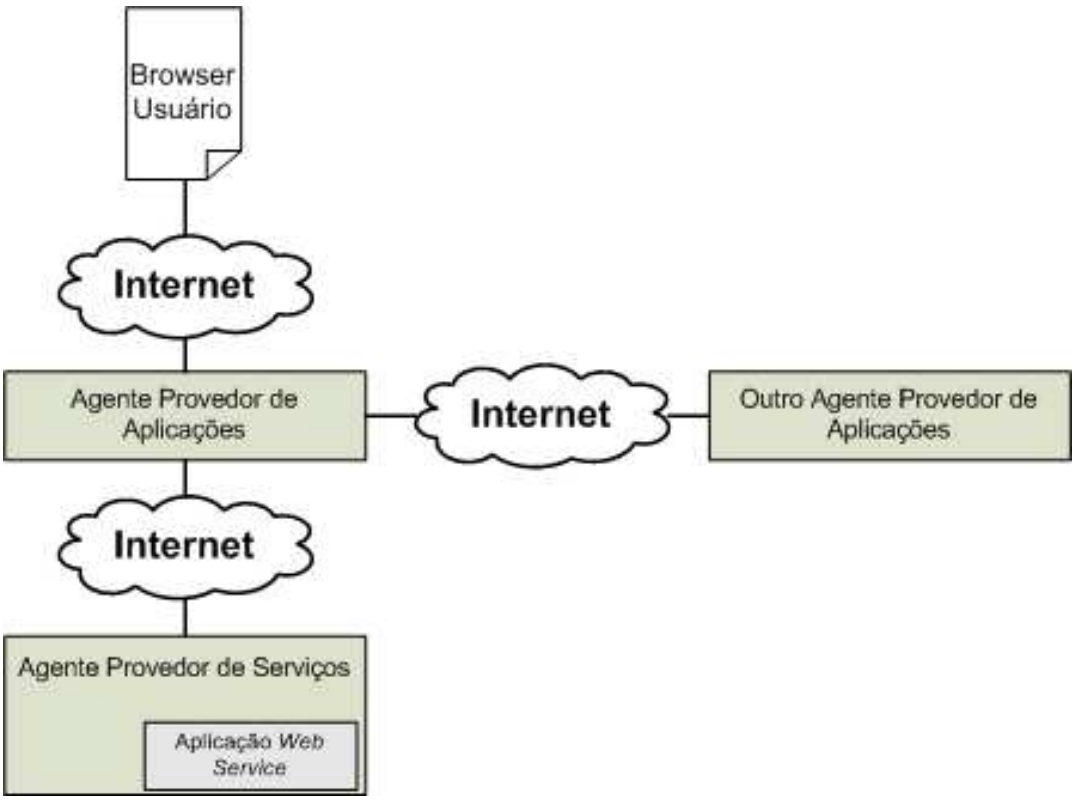

Figura 3.2: Arquitetura do *FAmCorA*
Fonte: Extraída e adaptada de (PESSOA; NETTO; MENEZES, 2002).

Dentre as características presentes em cada uma das aplicações, têm-se:

- Provedores de Serviços Básicos: possui um conjunto de aplicações básicas, como Gerenciador de Banco de Dados, Servidor de *Email*, Servidor de Lógica e de um subconjunto de *APIs* do Sistema Operaracional (Sistema de Arquivos);

- Provedores de Engenho: encapsula um tipo de aplicação capaz de realizar algum tipo de raciocínio inteligente/automático em benefício do usuário, como *IndexSearch Engine, LatentSemantic Indexing Engine* (DEERWESTER et al., 1990) e *MBR Engine*[2] (STANFILL; WALTZ, 1986);

- Provedores de Aplicação: provê ferramentas encapsuladas em agentes de software. As ferramentas presentes são: Agente Fórum, Agente *Chat*, Agente Mural (Quadro de Avisos), Agente Agendador de Reuniões, Agente Editor de Questionários, Agente Editor de Mapas Conceituais, Agente de *Email*, Agente de Arquivo, Agente *QSabe*[3], Agente Mediador de Discussão Centrada em Documentos, Agente *BigBrother*[4], Agente *Link* e o Agente Sabiá[5].

A extensibilidade que foi dada ao ambiente através da construção do framework é uma vantagem do *FAmCorA*, além da disponibilização de máquina de inferência.

Porém, *FAmCorA* falha no aspecto de autoria, não provendo facilidades ao autor. Ressalta-se, também, que o aspecto cooperativo do ambiente é precário, além de não possuir tutoria personalizada. Além disso, o ambiente não fornece suporte para disponibilizar serviços de resolução de problemas e tutoria personalizada.

3.3 CTAT

CTAT é uma ferramenta de autoria para construção de tutores cognitivos[6] que vem sendo desenvolvido desde 2002 pela Universidade de *Carnegie Mellon* (CMU, 2006) e está disponível para *download* em (CTAT, 2003).

O Ambiente objetiva dar suporte tanto para engenheiros de softwares quanto para autores. O desenvolvimento e distribuição no ambiente CTAT ocorre de duas formas:

- Tutoria baseada em Exemplos: responsável pela criação de aplicações sem programação, porém a especificação do problema tem que ser definida. O usuário deverá utilizar

[2]Memory Based Reasoning

[3]Ferramenta para aprendizagem automática do perfil de cada participante.

[4]Ferramenta que notifica quais os participantes estão *online* para troca de mensagens instantâneas.

[5]ferramenta que permite modificar, destacar e comentar partes de textos e armazenar as anotações com referências para o documento.

[6]Do Inglês Cognitive Tutor Authoring Tools

as ferramentas localizadas em (ALEVEN et al., 2006b):

1. Registro de Comportamento: cria grafos de comportamentos, os quais são mapeados no espaço de soluções;
2. Facilidade de produção em massa: é produzido através de modelos de autoria;
3. Loja de Tutor[7]: um componente usado para seqüência de problemas de tutoria em um ambiente *Web*, além de suportar o uso experimental dos mesmos;
4. Loja de Dados: ferramenta com serviços de *log* e relatórios.

- Tutor Cognitivo: requer programadores de Inteligência Artificial para construir o modelo cognitivo de resolução de problemas do estudante. As ferramentas existentes são (ALEVEN et al., 2006a):

  1. Construtor de GUI[8]: equivale a *interface* do ambiente de resolução de problemas do estudante;
  2. Registro de comportamentos: possui três funcionalidades, *i)* registro de exemplos de comportamentos corretos e incorretos; *ii) interface* do estudante na forma de grafos de comportamento e *iii)* suporte a planejamento e testes do modelo cognitivo;
  3. Editor de memória de trabalho: suporte a edição do conteúdo;
  4. Árvore de conflitos e "Porque não Janelas"[9]: ferramenta para depuração do modelo cognitivo, provendo informações sobre regras;
  5. Console Jess (JESS, 2003): suporte para o programador interagir diretamente com o Interpretador de regras;
  6. Editor Externo: utilizado para edição de regras Jess para o modelo cognitivo;

Além disto, uma importante característica que vem sendo explorada pelo grupo de pesquisa refere-se à integração com uma ferramenta de tutoria cognitiva em um ambiente colaborativo baseado na Web (HARRER et al., 2005), chamada *Cool Modes*[10]. A Figura 3.3 aborda a arquitetura do ambiente de colaboração integrado com *CTAT*.

[7] Do inglês: Tutor Shop.
[8] Do Inglês Graphical User Interface
[9] Do Inglês Conflict Tree and Why Not Window
[10] Do Inglês Collaborative Open Learning and MODEling System

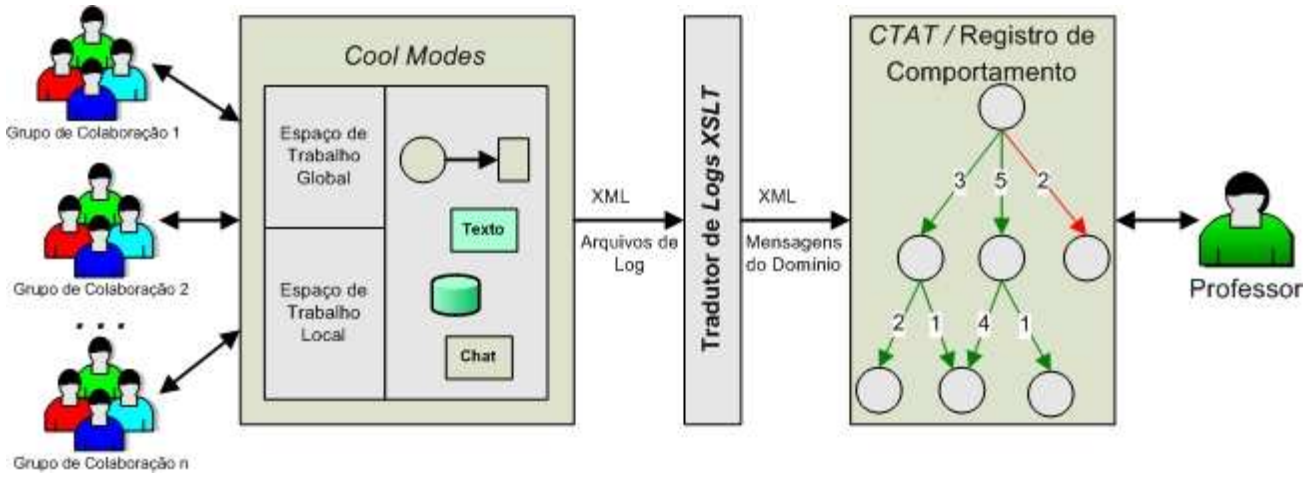

Figura 3.3: Arquitetura do *CTAT*

Fonte: Extraída e adaptada de (MCLAREN; KOEDINGER; SHNEIDER, 2005).

Cool Modes é uma ferramenta colaborativa de *design* para suportar "conversas" entre estudantes (MCLAREN; KOEDINGER; SHNEIDER, 2005). Todo processo de avaliação encontra-se em CTAT que está integrado ao ambiente.

Apesar de *CTAT* prover um ambiente de autoria, tais facilidades não são bem evidenciadas, visto que o desenvolvedor sempre terá que estar presente no processo de construção de ambientes educacionais. Além disso, tal ambiente não provê facilidades para o papel do professor no ambiente.

3.4 EASE

EASE[11] é uma ferramenta de autoria desenvolvida pelo Departamento de Matemática e Ciências da Computação (TU/E,) da Universidade de Tecnologia de *Eindhoven* (EINDHOVEN,) em parceria com a Universidade de Osaka (OSAKA, 2005).

O Ambiente *EASE* é evolutivo, possuindo características de raciocínio sobre o próprio comportamento e fazendo uso de regras, para manutenção nas diferentes partes do processo de autoria (dividido em camadas), como mostra a Figura 3.4.

[11] Do Inglês Evolutional Authoring Support Environment.

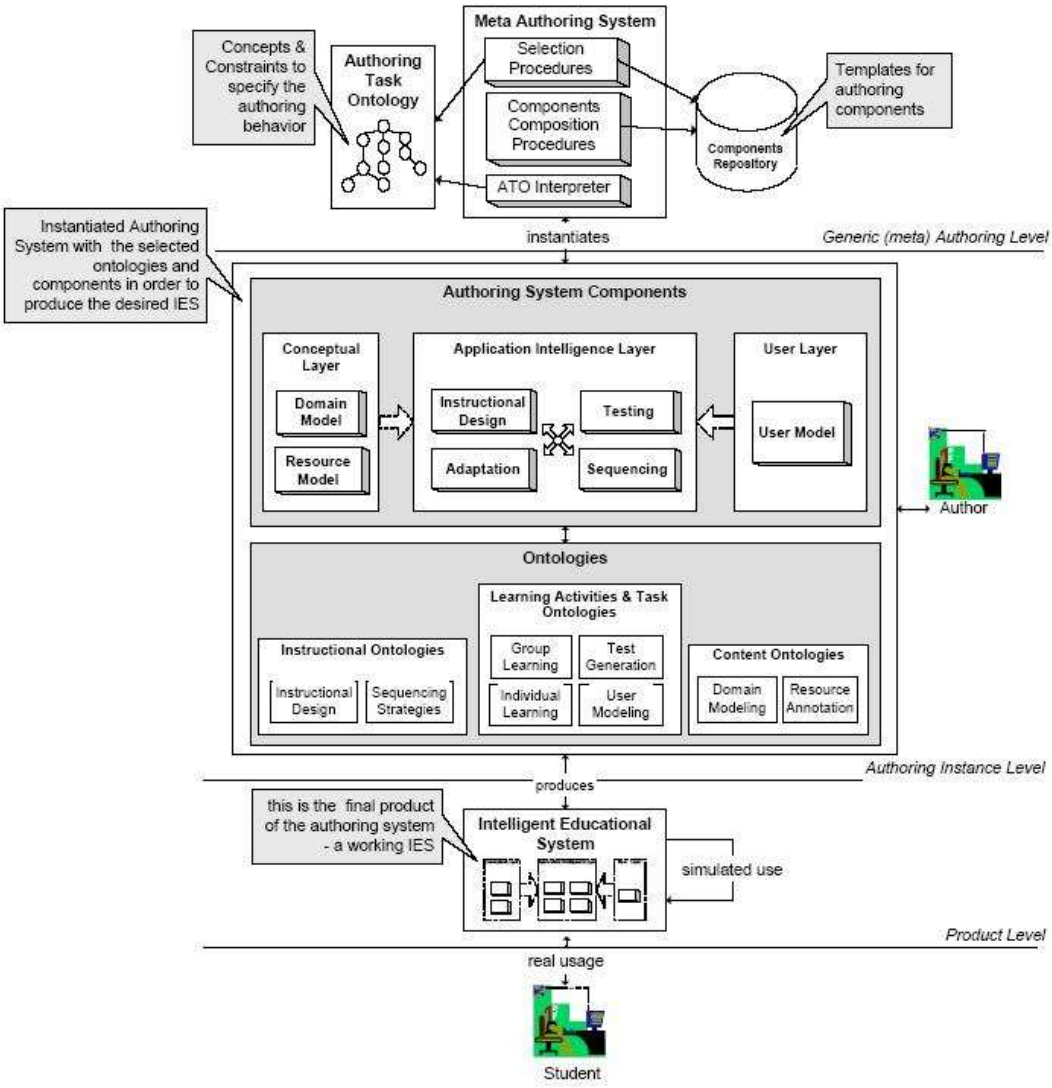

Figura 3.4: Arquitetura do *EASE*

Fonte: Extraída de (AROYO et al., 2004).

Como abordado na Figura 3.4, as três camadas presentes no ambiente são:

- Nível de Meta-autoria: equivale ao meta-conhecimento representado no ambiente educacional. Além disso, esta camada possui a estrutura conceitual, para que o processo de autoria seja iniciado. Este meta-nível foi construído baseado em uma ontologia de tarefas (AROYO; MIZOGUCHI; TZOLOV, 2003);

- Nível de autoria de instância: nesta camada, o ambiente de autoria é iniciado instanciando um *meta-schema* com os conceitos, modelos e comportamentos;

- Nível de Produção: equivale ao ambiente educacional propriamente dito. Ou seja, após a configuração/definição do conteúdo, tem-se um ambiente educacional em nível de produção.

Este ambiente equivale a uma ferramenta de autoria, dando suporte à disponibilização/utilização de regras para configuração do sistema educacional, além da utilização de ontologias.

Porém, o ambiente, apesar de objetivar o aspecto evolutivo, não possui características de extensibilidade, provendo novas funcionalidades ao ambiente.

3.5 SAILE

SAILE[12] propõe um *framework* para aprendizagem colaborativa desenvolvido por diversos grupos americanos (MITRE, 1997; GATECH, 2006; PREDICTIVENETWORKS, 2006; CLASSROOM, 2006).

O *Framework SAILE* tem por objetivo prover serviços baseados na *Web*, através de ferramentas síncronas e assíncronas. No ambiente os estudantes interagem através de uma ferramenta de *chat*, buscando a resolução de um determinado problema (GOODMAN et al., 2001).

O *Framework* foi construído utilizando tecnologia Java e um paradigma Cliente-Servidor, como mostrado na Figura 3.5.

As camadas presentes em *SAILE* são:

- *Interface* do Usuário: camada de interação do sistema com os estudantes, possibilitando a colaboração;

- *JOSIT*[13]: tecnologia Java que permite a captura das informações do usuário. *JOSIT* trabalha totalmente com componentes Java Swing (ZUKOWSKI, 2005) e AWT [14] (ZUKOWSKI, 1997);

[12]Do Inglês Synchronous and Asynchronous Interactive Learning Environment

[13]Do Inglês The Java Observation, Scripting, and Inspection Tool

[14]Do Inglês Abstract Window Toolkit

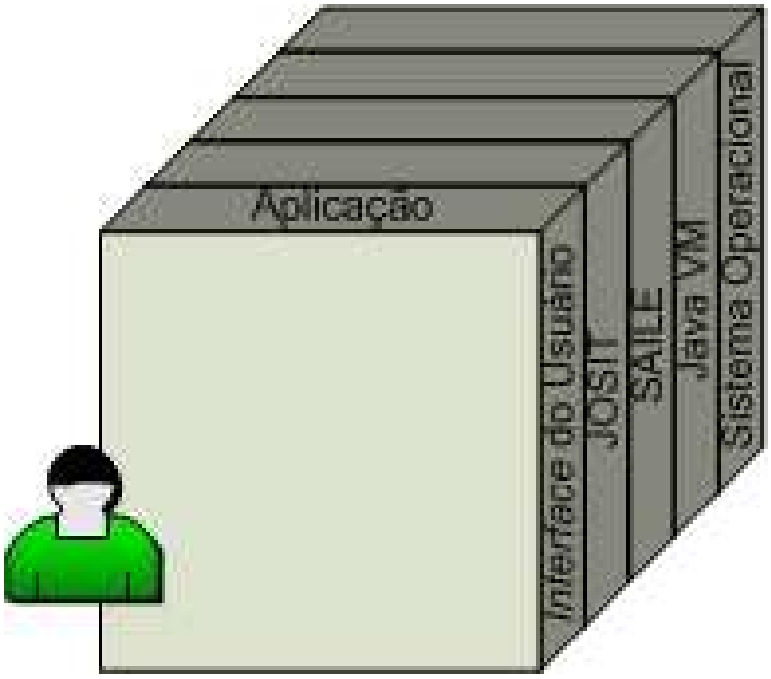

Figura 3.5: Arquitetura do *SAILE*
Fonte: Extraída e adaptada de (GOODMAN et al., 2001).

- *SAILE*: é composto por um *chat* e uma palheta para compartilhamento de aplicações em Java;
- SAILE RMI Server: utiliza a tecnologia de RMI[15] para gerenciar a conexão de colaboração entre os usuários e a ferramenta assíncrona ART[16], permitindo aos usuários assíncronos participar do processo de resolução de problemas.

Tal ambiente falha na tentativa de prover facilidades ao papel do professor.

3.6 COLE

COLE[17] é um ambiente de suporte à aprendizagem colaborativa desenvolvido pela Pontifícia Universidade Católica do Paraná (PUC-PR,) e pela Universidade Tecnológica Federal do Paraná (UTTPR, 2005).

O ambiente *COLE* tem por objetivo aumentar as competências sociais durante o ciclo de

[15]Do Inglês Remote Method Invocation
[16]Do Inglês Asynchronous Replay Tool
[17]Do Inglês Collaborative Online Learning Environment

vida da aprendizagem. Uma abordagem multiagente foi adotada para permitir ao educador a manipulação de uma grande quantidade de dados. Conceitos como interação humana, aprendizagem baseada em problemas e portfolios são recursos utilizados para atingir os objetivos do ambiente (AZEVEDO; SCALABRIN, 2005).

É definida uma metodologia para desenhar sistemas multiagentes, através da análise de sistemas para sistemas de agentes (AZEVEDO, 1997). A metodologia é dividida em oito passos, porém *COLE* utiliza apenas as fases de cinco até oito. Os passos são:

1. Inserir pontos de vistas: consiste em adquirir informações relevantes, através de entrevistas;

2. Classificar atividades e recursos: organiza as informações adquiridas na entrevista;

3. Obter validação do grupo: grupo determina os serviços potenciais que devem ser implementados;

4. Descrição dos serviços: a partir do momento em que diversos grupos definiram os serviços que cada um terá, o engenheiro do conhecimento gera uma tabela com os serviços finais que serão produzidos;

5. Escrever cenários: para cada serviço é definido um cenário, descrevendo como será o funcionamento de cada serviço;

6. Construir o modelo: de acordo com o cenário, são definidas telas para simulação do modelo;

7. Identificar Competências: competências são definidas para cada cenário, ou seja, informações como nome, descrição e parâmetros de entrada e saída são especificadas para cada competência, em cada cenário;

8. Sintetizar Competências: nesta fase, as competências são agrupadas, caso haja redundância entre grupos diferentes.

Além disso, agentes inteligentes com funcionalidades particulares podem ser implementados no ambiente, através do agente genérico, como mostrado na Figura 3.6.

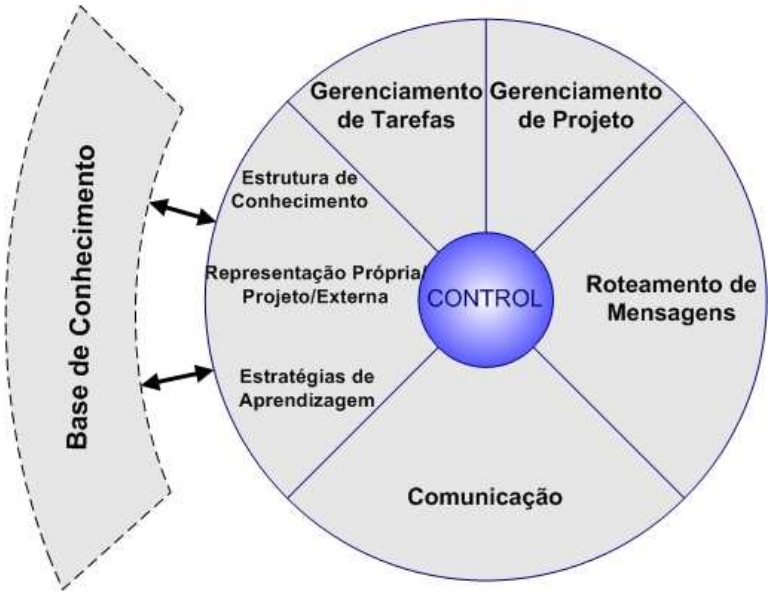

Figura 3.6: Arquitetura do *COLE*

Fonte: Extraída e adaptada de (AZEVEDO; SCALABRIN, 2005).

Algumas funcionalidades estão encapsuladas no agente genérico, como os mecanismos de processamento básico, estrutura, habilidades, garantia de comportamento externo e interno e capacidade de se adaptar em novos ambientes (SCALABRIN et al., 1996).

Tal ambiente é uma ferramenta colaborativa de autoria, possuindo uma arquitetura aberta para adição e deleção de agentes no ambiente, porém, o sistema não provê facilidades ao autor na configuração do processo de tutoria personalizada.

3.7 DoCTA

DoCTA[18] é um ambiente de suporte à aprendizagem colaborativa desenvolvido pela Universidade de *Bergen* (UIB, 1999), desde 1999. Atualmente, a equipe trabalha no projeto DoCTA-NSS (DOCTA-NSS, 2006)

O ambiente *DoCTA* tem por objetivo dar suporte à colaboração entre estudantes, através de cenários de aprendizagem, onde estudantes colaboram através de *groupwares*. O ambiente possui agentes inteligentes tanto para ferramentas síncronas quanto assíncronas (CHEN; WASSON, 2005). As ferramentas são divididas através de duas ferramentas diferentes, sendo elas:

- *FLE*: é uma ferramenta assíncrona (MUUKKONEN; HAKKARAINEN; LAKKALA, 1999),

[18]Do Inglês Design and use of Collaborative Telelearning Artefacts

equivalendo a um grupo de trabalho baseado na Web para dar suporte à aprendizagem colaborativa. Esta ferramenta foca na resolução de problemas, onde os estudantes podem construir suas teorias, hipóteses e interpretações de acordo com o seu conhecimento. A partir do processo de resolução de problemas, o estudante desenvolverá suas capacidades meta-cognitivas, refletivas e críticas. Para que a colaboração seja possível, diversos módulos são disponibilizados, como módulo de construção do conhecimento, *chat*, administração e *WebTop*[19] (DOLONEN W. CHEN, 2003). Esta ferramenta possui um agente cujo objetivo é monitorar o processo de colaboração, dando dicas aos estudantes, acompanhando o estudante e monitorando as atividades tanto dos professores quanto dos estudantes. A Figura 3.7 aborda a estrutura interna do agente;

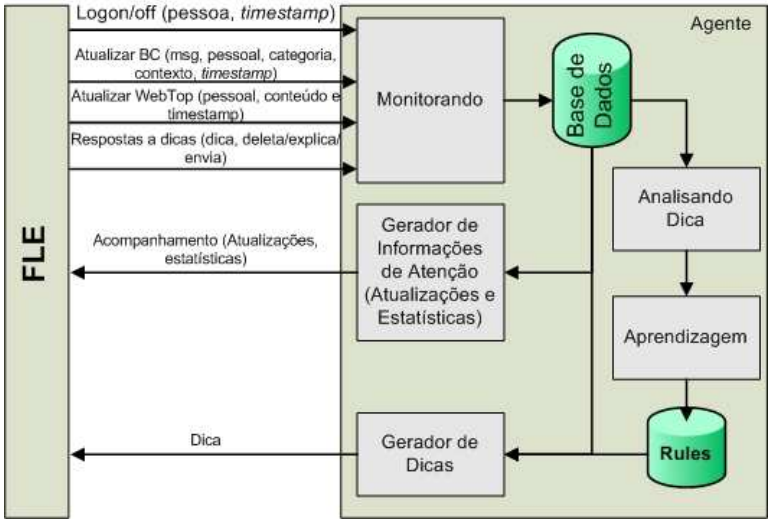

Figura 3.7: Arquitetura do *DoCTA-FLE3*

Fonte: Extraída e adaptada de (CHEN; WASSON, 2005).

- *Mindmap Building*: equivale a uma ferramenta síncrona para o ambiente colaborativo, onde o usuário modela o entendimento conceitual de certo tópico abordado, onde o espaço de trabalho consiste de duas ferramentas de Mindmap (uma individual e uma compartilhada). Além disso, esta ferramenta possui um agente cujo objetivo é gerar dicas, enviar dicas e mostrar o módulo selecionado (vide Figura 3.8).

[19]Módulo para estudantes e professores poderem compartilhar recursos, como documentos, notas, links, entre outros.

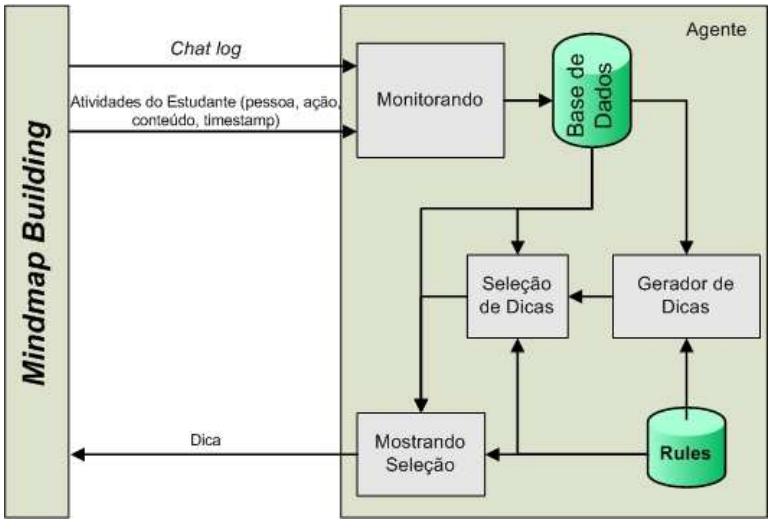

Figura 3.8: Arquitetura do *MindMap Building Tool*

Fonte: Extraída e adaptada de (CHEN; WASSON, 2005).

Tal ambiente não cumpre com o provimento das facilidades ao professor.

3.8 Tabela Comparativa

Esta seção objetiva comparar os trabalhos supracitados. Da mesma forma que foi avaliado acima, a Tabela 3.1 levará em consideração aspectos de engenharia de software, inteligência artificial e educação e paradigmas educacionais.

Tabela 3.1: Tabela Comparativa dos Trabalhos Relacionados

Ferramenta	**Engenharia de Software**		**Inteligência Artificial e Educação**			**Paradigma Educacional**			**Outros**
-	Framework	Autoria	Agentes Inteligentes	Ontologia	RP[1] Automático	*ILE*	STI	Outro	Outros[2]
I-Help		x	x			x			Motivação
FAmCorA	x	x	x			x			PDAs
CTAT		x				x	x		Jess
EASE		x		x		x			Evolução
SAILE						x			
COLE		x	x			x			
DoCTA		x	x			x			Metodologia
Esmahi	x		x			x			*Overlay Fuzzy*

[1] Resolução de Problemas Automático

[2] Outros Aspectos relevantes

Capítulo 4

Plataforma Proposta

4.1 Considerações Preliminares

A fim de facilitar o entendimento mais abrangente da plataforma proposta, segue-se uma descrição dos seus alicerces conceituais, isto é, um arcabouço conceitual para ambientes interativos de aprendizagem.

4.1.1 Arcabouço Conceitual

O objetivo desta seção é definir os papéis presentes em um ambiente interativo de aprendizagem, bem como a interação entre os papéis. Com isso, objetiva-se resolver alguns dos problemas atualmente relatados pela literatura da área.

Os papéis envolvidos em AIAs vão desde a concepção, desenvolvimento e utilização de um ambiente. Com isso, papéis concernentes a concepção e desenvolvimento são: *i) desenvolvedor/programados*: desempenha o papel de desenvolver e adicionar funcionalidades ao ambiente educacional; *ii) Autor*: desempenha o papel de personalização do ambiente, como cadastramento do conteúdo do domínio, definição das informações/características a serem consideradas do estudante, estratégias de ensino, entre outras e *iii) Engenheiro do Conhecimento*: este papel é responsável pela configuração de mecanismos baseados em conhecimento, como raciocínio baseado em regras.

Já na utilização dos AIAs, normalmente, os papéis presentes são: *i) estudante*: desempenha o papel de aprender através de interações com recursos pedagógicos e com outros

estudantes; *ii) professor*: desempenha o papel de colaborar/dar suporte ao estudante no processo de aprendizagem; *Agente Tutor Artificial*: agentes computacionais que interagem com o estudante, ajudando-o de forma cooperativa para a resolução de problemas, além de cooperar com o professor na gerência das interações de diversos usuários.

Basicamente, um ambiente interativo de aprendizagem deve ser capaz de prover um ambiente de boa usabilidade disponibilizando ferramentas para que *i)* o sistema possa ser definido de maneira fácil e *ii)* o estudante disponha de um ambiente com características de tutoria personalizada e resolução de problemas.

Com a identificação dos papéis fundamentais em AIA, diversas preocupações surgem, as quais se citam abaixo:

- Camada de Infra-estrutura: implementa o tipo de tecnologia que é responsável pela comunicação entre os papéis, sendo feita de acordo com a abordagem utilizada para o desenvolvimento do AIA. Por exemplo, se o paradigma utilizado for baseado em serviços, esta camada poderia ser baseada em *Web Services* (NEWCOMER, 2002), porém se o paradigma for baseado em agentes, tal camada poderia utilizar um *framework* para agentes, a exemplo do Jade (TILAB, 2005), Jadex (BRAUBACH; POKAHR; WALCZAK, 2002) ou Compor (ALMEIDA; COSTA; PERKUSICH, 2005);

- Camada Social: representa o nível de conhecimento entre os diversos pares de papéis, onde cada par possui a sua característica social. A forma de definição/identificação de sociabilidade nos diversos papéis pode variar, por exemplo: *i) Social (Estudante, Estudante)*: pode ser definido através das turmas/grupos que cada estudante participa; *ii) Social (Agente Tutor, Estudante)*: o assunto que o estudante está estudando, equivalendo a mesma competência do Agente Tutor; *iii) Social (Agente Tutor, Professor)*: pode ser definido através da manutenção feita por um professor no conteúdo instrucional do Agente Tutor. A tecnologia de representação do conhecimento social adotada depende da abordagem utilizada no ambiente, onde, por exemplo, pode ser definido o uso de XML ou ontologias, dentre outras;

- Recursos: dependem do tipo de papel associado. Por exemplo, para o estudante, os recursos disponibilizados para interação entre estudantes são as ferramentas colaborativas como Fórum, *chat* etc. Para o professor são os recursos para avaliação do estu-

dante, bem como manutenção do plano instrucional, entre outros. Os Agentes Tutores são os agentes de ensino[1], responsáveis por atividades de suporte instrucional.

- Interface: é o meio de interação entre os papéis, a exceção da interação entre agentes artificial.

4.2 Concepção da Plataforma Proposta

Com a definição dos principais conceitos sobre ambientes interativos de aprendizagem, esta seção especifica as características do arcabouço proposto, de acordo com os aspectos basilares para i) interação e/ou triangulação, entre os três papéis fundamentais (estudantes, professores e agentes tutores) e ii) disponibilizar de um ferramental para os papéis presentes na concepção e desenvolvimento de AIAs.

Com isso, objetiva-se construir uma plataforma que i) garanta a rápida construção de ambientes interativos de aprendizagem, através de facilidades de autoria; ii) distribua ambientes educacionais de forma personalizada e iii) evolua seu conhecimento e capacidades de inferência. Além disto, tal plataforma deve prover recursos de extensibilidade para o desenvolvedor poder adicionar novas funcionalidades.

A plataforma proposta é chamada ForBILE[2] e possui uma abordagem baseada em agentes. Para o desenvolvimento da plataforma, houve uma separação entre a implementação dos agentes e da aplicação, como mostrado na Figura 4.1. Tal separação permite a fácil mudança tanto do ambiente de agentes utilizado quanto do paradigma baseado em agentes.

As tecnologias que estão sendo utilizadas para a proposição da plataforma são *TOMCAT* (BRITTAIN; DARWIN, 2003), *Jade* e *OWL-DL* (OWL, 2006).

Na avaliação dos ambientes educacionais, as características identificadas foram:

1. Funcionalidades: todo ambiente possui funcionalidades específicas, podendo variar desde um recurso de *interface* gráfica a ferramentas inteligentes;

2. Mecanismos de Inferência: representam mecanismos de inferência inerentes a ambientes educacionais, podendo ser utilizado de diversas formas diferentes (THOMP-

[1]Dependendo da complexidade do domínio de ensino, os agentes de ensino podem requerer métodos sofisticados de resolução de problemas/tomada de decisão e aprendizagem.

[2]Do inglês **F**ramework f**or B**uilding **I**nteractive **L**earning **E**nvironment

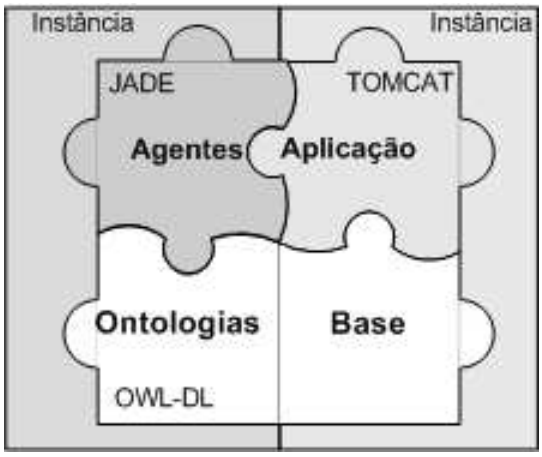

Figura 4.1: Separação Agentes - Framework

SON; GÖKER; LANGLEY, 2004; BLOEDORN; MANI; MACMILLAN, 1996; FERGUSON et al., 2006);

3. Aspectos: diz respeito às informações que tornam o processo de ensino personalizado. Um aspecto inerente a qualquer ambiente educacional equivale ao cognitivo, que representa o domínio de ensino;

4. Modelos: são peças fundamentais no processo de ensino-aprendizagem, sendo eles o modelo do domínio, estudante e pedagógico. Além disto, em ambientes de suporte à colaboração, adiciona-se mais um módulo, chamado de colaborativo.

Com as características identificadas, a seguir são descritas as arquiteturas de agentes e de implementação.

4.2.1 Arquitetura de Agentes

Na Figura 4.2 é apresentada a arquitetura do arcabouço baseado em agentes para construção de ambientes de aprendizagem interativa. A arquitetura é dividida em camadas, destacando-se os recursos presentes no núcleo do sistema. Tais recursos equivalem ao conjunto de ferramentas interativas, agente mediador, agente de persistência e uma sociedade de agentes composta por agentes tutores autônomos (ATA) e agentes de suporte (AS). Na infra-estrutura de agentes, foi utilizado o *framework* Jade. Jade foi escolhido por implementar os padrões de

interoperabilidade para comunicação entre agentes (FIPA), além da credibilidade adquirida através da utilização por diversos grupos de pesquisa.

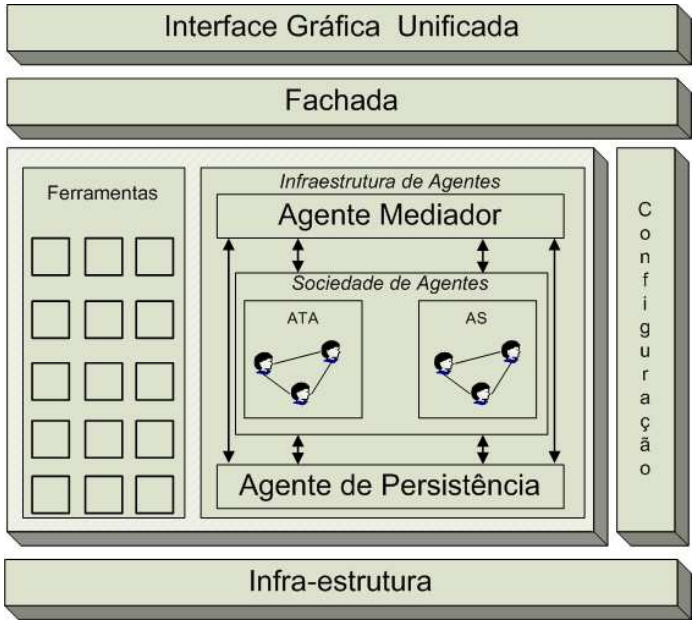

Figura 4.2: Arquitetura do sistema de autoria.

Ambientes de aprendizagem colaborativa, a exemplo de comunidades virtuais, necessitam de ferramentas que possibilitem a interação entre os usuários dos papéis. As ferramentas interativas disponíveis no sistema são:

- *fórum*: permite aos usuários postarem tópicos com discussões relativas aos interesses de determinada comunidade;
- *scrap*, *blog*, *enquete* e *e-mail*: para interação entre usuários;
- *biblioteca digital*: onde podem ser armazenadas informações, em formato de arquivo digital, que poderão ser compartilhadas entre os usuários de uma comunidade virtual. Esses arquivos devem estar relacionados com o contexto da comunidade e os usuários podem iniciar discussões sobre eles.

Três tipos de concepções de agentes estão presentes na arquitetura:

- Agente Mediador: responsável por mediar a interação entre os agentes da sociedade de agentes;

- Agente de Persistência: possui características de todos os mecanismos de persistência presentes no arcabouço;

- Sociedade de Agentes: representa uma sociedade de agentes heterogêneos. Os agentes são divididos em dois grupos: i) agentes de suporte (AS): possuindo informações sobre mecanismos de inferência, e ii) agentes tutores autônomos (ATA): agentes possuindo informações sobre aspectos educacionais.

Com as características identificadas, a Figura 4.3 ilustra de forma clara a separação e o meio de comunicação entre a aplicação e os agentes.

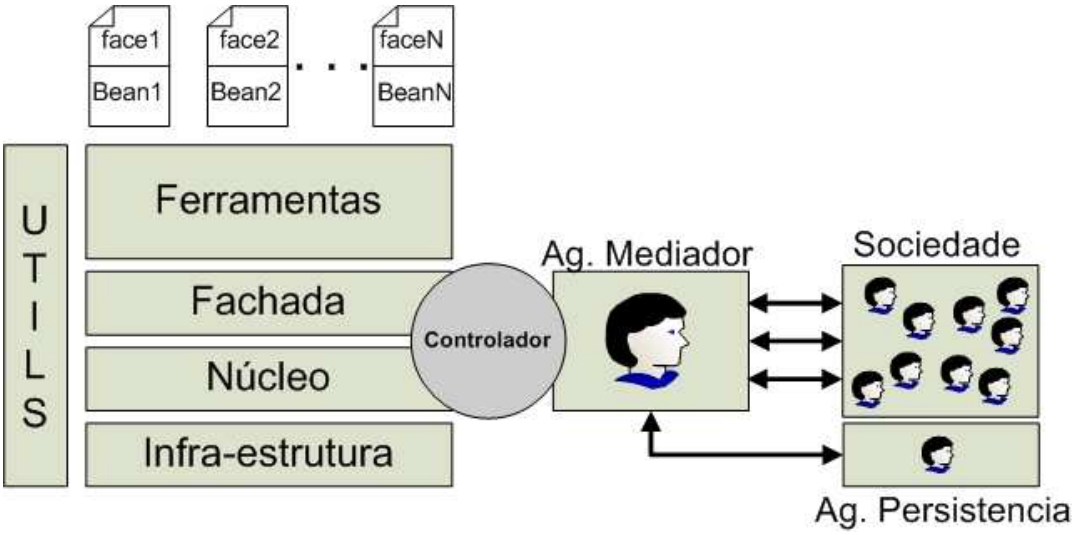

Figura 4.3: Aplicação / Agentes do ForBILE

Para que a comunicação entre a aplicação e os agentes fosse possível, foi adicionado mais um agente ao ambientes para controlar o ciclo de vida dos agentes, adição remoção e exclusão

4.2.2 Arquitetura de Implementação

Para que a especificação da plataforma fosse feita, utilizou-se a metodologia de construção de *framework bottom-up*. Com isso, a partir da avaliação de diversos trabalhos, a análise do domínio foi feita buscando identificar os *frozens* e *hotspots* da plataforma, de acordo com as características identificadas.

Na Figura 4.4 é apresentada a arquitetura de implementação da plataforma.

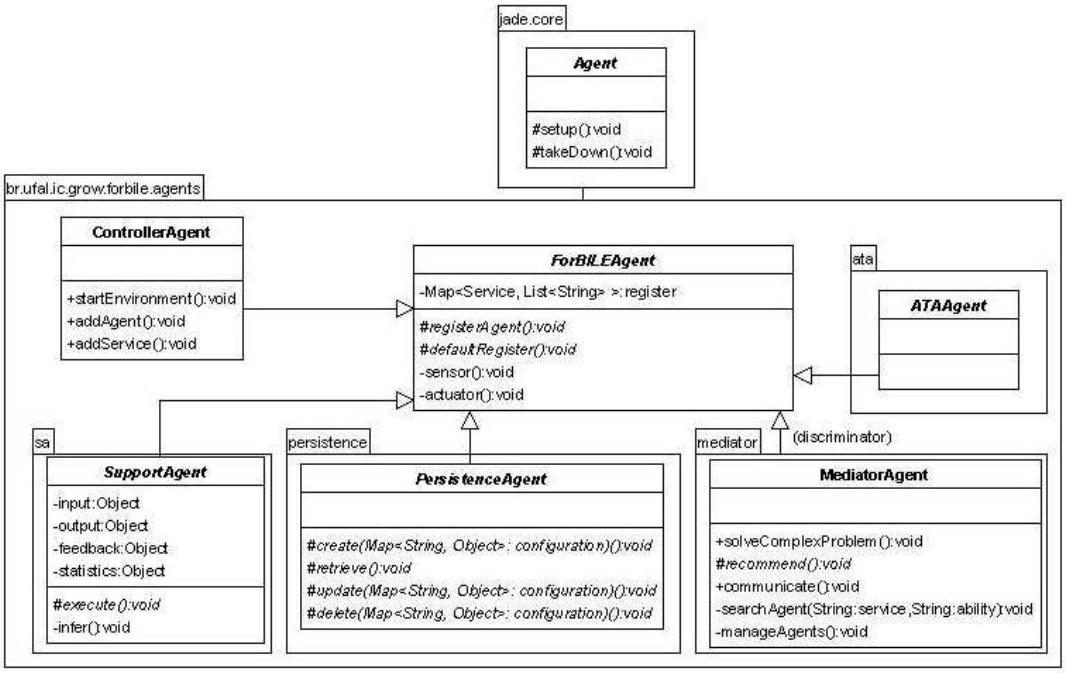

Figura 4.4: Diagrama de Classes do *Kernel* da plataforma

O ponto de acesso entre a plataforma e a aplicação equivale a um agente controlador que faz a comunicação do agente mediador com a fachada/núcleo da aplicação.

Além disso, o agente controlador possui várias responsabilidades no ambiente, sendo elas:

- construir todos os agentes quando o ambiente é iniciado, como mostrado o diagrama de seqüência na Figura 4.5;
- Adicionar e remover agentes da sociedade;
- Adicionar, remover e atualizar $< Serviço, Habilidade >$ de cada agente.

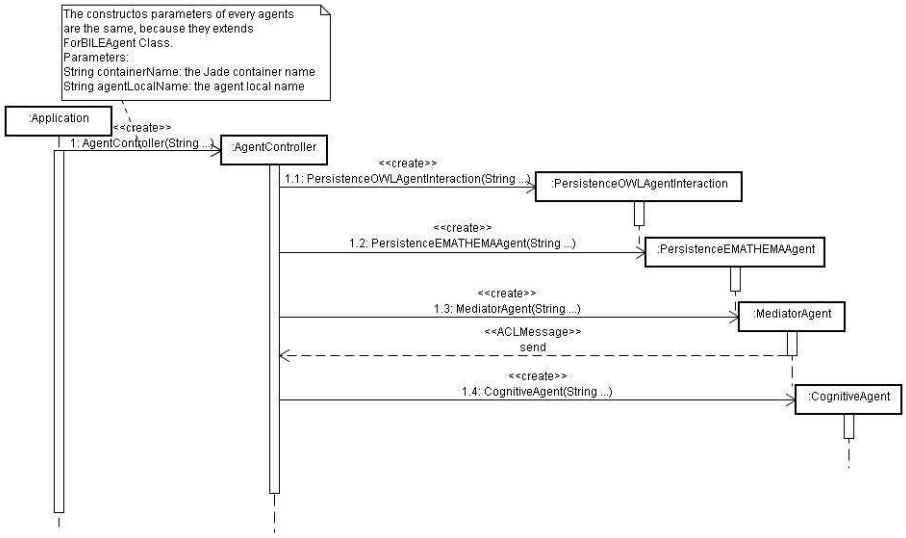

Figura 4.5: Diagrama de seqüência sobre iniciar o ambiente.

4.3 *A*gente *M*ediador - AM

Na Seção 4 foram apresentadas características gerais da arquitetura de agentes e implementação da plataforma proposta. A plataforma permite subsidiar a interação entre os papéis dos estudantes, professores e agentes tutores. Para que esta interação seja possível, é necessário definir o conhecimento social dos diversos agentes da plataforma. Ou seja, o agente mediador é responsável pela interação, tanto na especificação do domínio de ensino, quanto na utilização pelo estudante.

O agente mediador é motivado pelo conhecimento social que deve estar presente no paradigma baseado em agentes. AM possui quatro funcionalidades, sendo elas:

1. gerenciar os agentes tutores autônomos da sociedade de agentes.
2. garantir a comunicação entre os agentes da sociedade, permitindo aos agentes solicitarem ao agente mediador qual agente tem habilidade para determinada tarefa;
3. recomendação de acordo com as necessidades do agente solicitante;
4. coordenar o processo de resolução de problemas complexos. Ou seja, através das habilidades dos agentes, identificar qual agente poderá cooperar no processo de resolução de problemas;

4.3.1 Gerenciamento de ATAs

Este funcionalidade é responsável por todo o gerenciamento dos agentes tutores autônomos. O gerenciamento ocorre de forma dinâmica (Vide Figura 4.6) dos agentes tutores autônomos, possuindo as seguintes funcionalidades:

1. A partir do momento em que a plataforma é iniciada, o agente mediador é iniciado e ele verifica quais serão os ATAs que serão construídos. Para isto, o processo ocorre em três fases (Vide Figura):
   (a) Solicita ao agente de persistência a lista de currículos do domínio. Com isso, verifica se a ontologia com os protocolos de interação já foi povoada com os agentes tutores autônomos e se os indivíduos estão corretos;

(b) com a lista de currículos, o agente mediador povoa/atualiza a ontologia, mapeando os currículos em habilidades, com isso, é definido o par $< Agente, Habilidade >$. Este par é feito com a verificação pelo agente mediador, se algum agente possui determinada habilidade;

(c) logo após, o agente mediador passará a lista de currículos para o Agente Controlador para definir os serviços de cada ATA, definindo assim a tripla $< Agente, Serviço, Habilidade >$.

2. Validar a entrada de novos ATAs na sociedade;

3. Verificar se algum agente ATA saiu da sociedade.

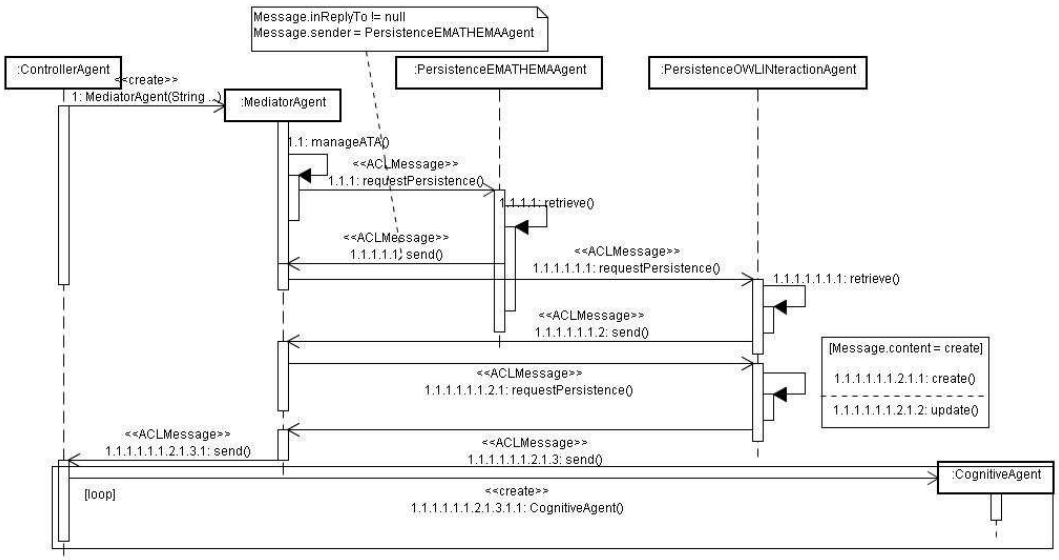

Figura 4.6: Diagrama de Seqüência com gerenciamento dos ATAs

4.3.2 Comunicação entre os Agentes

Foi definido um protocolo de comunicação entre os agentes presentes. Tal protocolo foi especificado através da construção de uma ontologia desenvolvida no Protégé (PROTéGé, 2006).

A ontologia possui três informações fundamentais, sendo elas:

1. Agente: informações básicas sobre os agentes, como nome, descrição, entre outros;

2. Serviço: equivale aos tipos de serviços disponibilizados na plataforma;

3. Habilidade: refere-se à habilidade presente em cada par $< Agente, Serviço >$.

A especificação da ontologia está descrita na Figura 4.7.

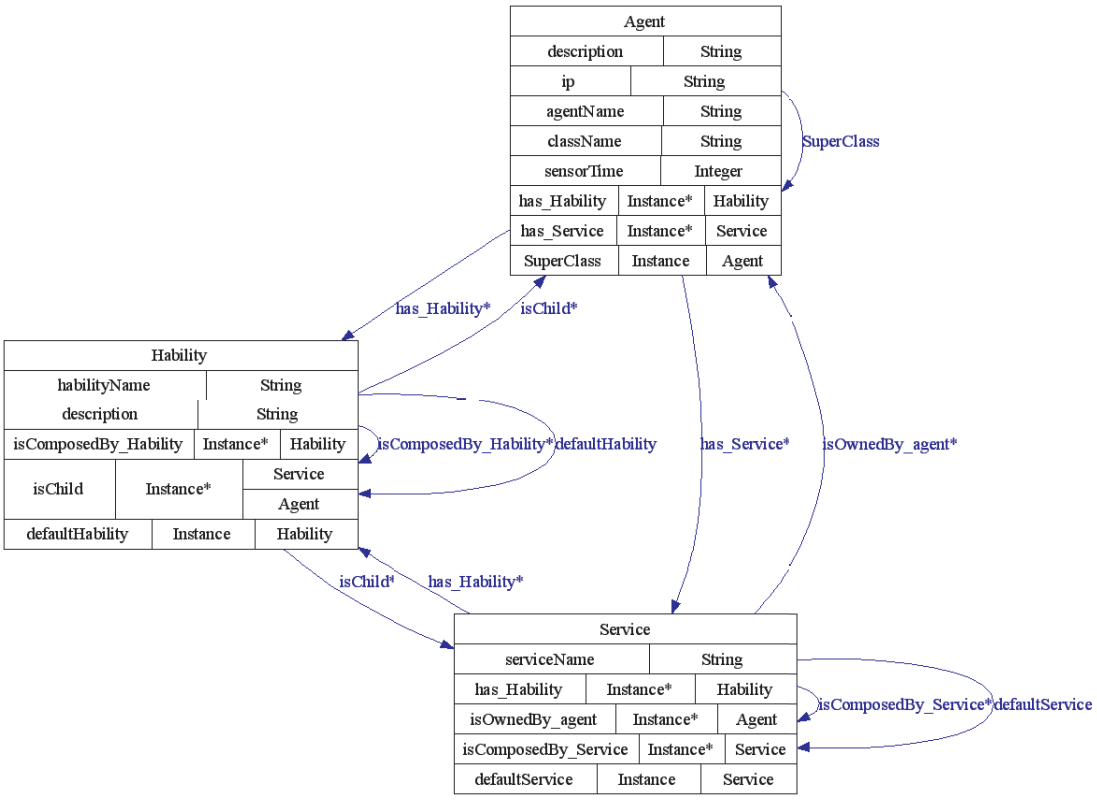

Figura 4.7: Ontologia com os protocolos de interação.

Além disto, a ontologia possui o pacote e os nomes das classes referentes a cada serviço/habilidade. Caso algum serviço/habilidade tenha mais de um tipo de implementação, deve-se definir qual é a implementação default. Com isso, a ontologia permite que o *framework* possa fazer a inversão de controle.

4.3.3 Recomendação de Agentes

A recomendação é a característica fundamental do agente mediador, pois é ela que define como a interação com o estudante irá ocorrer. A recomendação equivale a um ponto de extensibilidade, pois a implementação pode variar de acordo com as funcionalidades de um determinado sistema educacional.

Algumas implementações para a recomendação são:

- Agentes tutores: o AM pode recomendar agentes tutores de acordo com as necessidades do estudante e habilidades dos agentes tutores;
- Agentes Humanos: o AM pode recomendar especialistas humanos de acordo com as necessidades do estudante e habilidades dos agentes humanos;
- Conteúdo: a recomendação de conteúdo pode ser de diversas formas e de diversas fontes diferentes.

Além disso, cada uma das formas de recomendação pode variar de acordo com o algoritmo utilizado, onde a recomendação pode ser tanto por conteúdo quanto por perfil.

No ambiente, a recomendação é vista como um serviço, enquanto as diversas implementações é vista como uma habilidade, como mostrado na Figura 4.3.3.

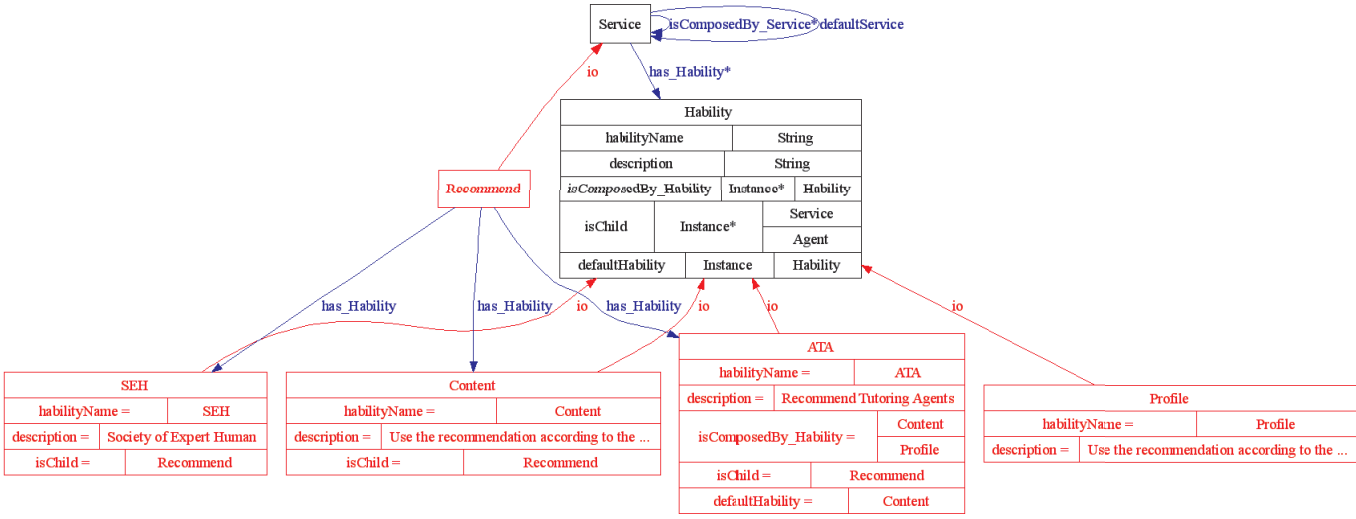

Figura 4.8: Ontologia com os protocolos de interação.

Pode-se observar na Figura que o indivíduo *ATA*, possui no *slot* chamado *isComposedBy_Hability* dois valores (*Content e Profile*). Isto quer dizer a recomendação de agentes tutores pode ser feita tanto por conteúdo ou por perfil. Porém, percebe-se também que o *slot defaultHability* possui o valor *Content*, ou seja, a implementação padrão de recomendação de ATAs é de conteúdo.

4.3.4 Resolução de Problemas Distribuída

A resolução de problemas refere-se aos problemas presentes nas unidades pedagógicas de cada agente tutor, como mostrado na Figura 2.13.

Caso um Agente Tutor AT_1 não saiba resolver determinado problema, então ele envia uma mensagem para o AM, informando que necessita solucionar o problema. São informados ao AM quais são os pré-requisitos necessários para a resolução do problema. Com isso, o problema é particionado em diversas tarefas, como mostrado na Figura 4.9. O AM verifica na ontologia quem possui a habilidade de resolver determinada tarefa e, então, alocar a tarefa de acordo com a tripla $< Agente, Habilidade, Serviço >$.

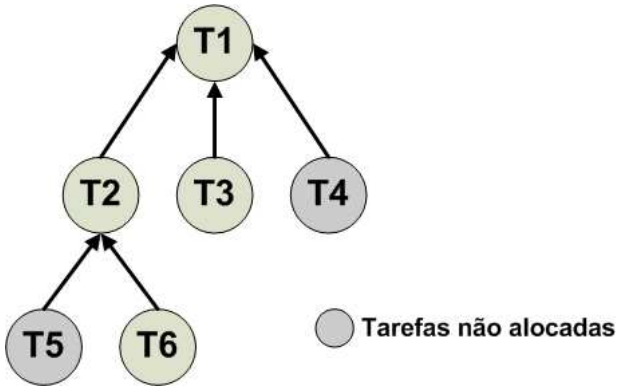

Figura 4.9: Representação gráfica da resolução de problema subdividido em tarefas.

Com isso, o agente Mediador é responsável pela coordenação do processo de resolução de problemas, enquanto cada agente tutor coopera para a resolução de uma determinada tarefa T_i. O sucesso da resolução de problemas ocorre pela capacidade de todos os agentes solucionarem a tarefa. A Figura 4.10 aborda o diagrama de seqüências do processo de resolução de problemas distribuídos.

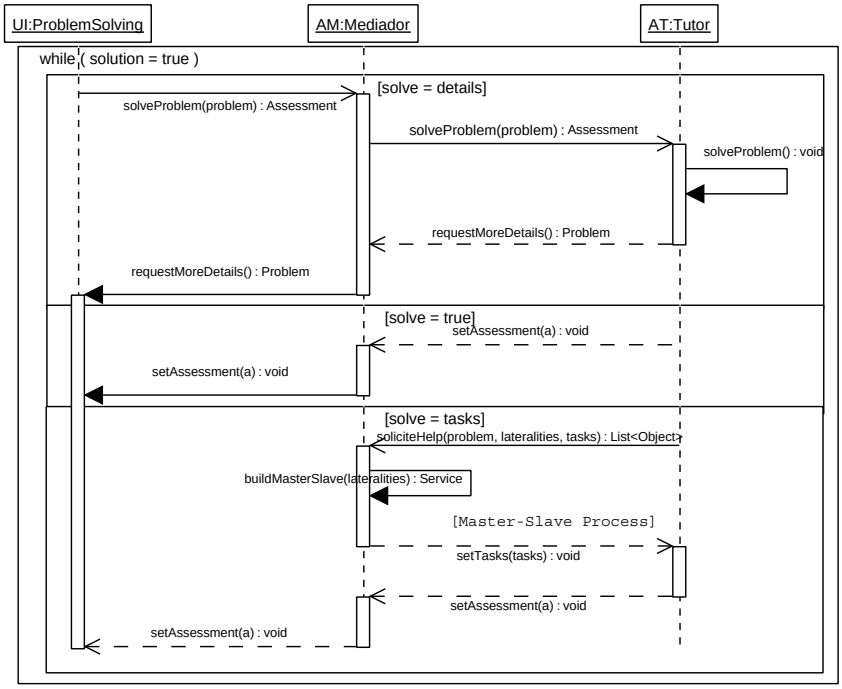

Figura 4.10: Diagrama de Seqüência do processo de resolução de problemas.

4.4 *Agentes Tutores Autônomos - ATA*

Os Agentes Tutores Autônomos são responsáveis pelo conhecimento sobre os aspectos envolvidos no processo de ensino-aprendizagem. Porém, é fundamental especificar os aspectos inerentes aos agentes tutores autônomos. A definição destes aspectos foi proveniente do estudo dos diversos ambientes educacionais, nos quais foram identificados três tipos, sendo eles: cognitivo, motivacional e afetivo (Vide Figura 4.11).

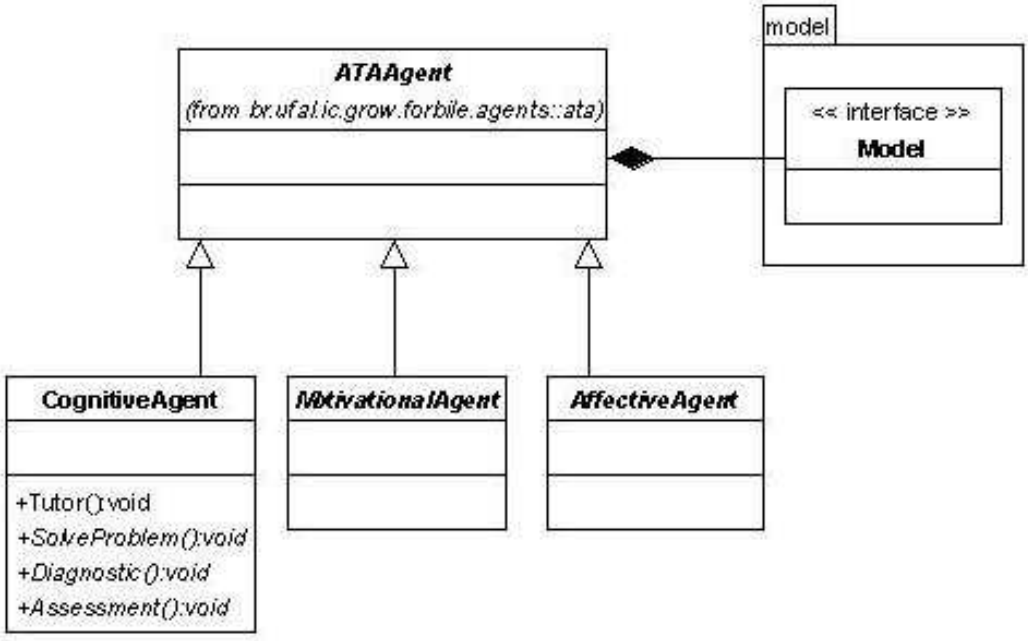

Figura 4.11: Diagrama de Classes dos Agentes Tutores Autônomos

Observa-se na Figura 4.11 que existe uma relação de composição entre a classe abstrata *ATAAgent* a interface *Model*. Isto quer dizer que todos os agentes que estenderem a classe abstrata *ATAAgent* poderá utilizar os modelos presentes em agentes tutores.

Além disso, são mostrados três extensões da classe abstrata *ATAAgent* (*CognitiveAgent*, *MotivationalAgent* e *AffectiveAgent*), porém, o único aspecto que deve está presente em todos os ambientes é o aspecto cognitivo.

Independente do tipo de implementação do Agente Tutor, o mesmo seguirá a arquitetura adaptada do Modelo Mathema e, com isso, terá seus componentes mapeados em Jade, como é mostrada na Figura 4.12.

As subseções que seguem mostram detalhes referentes ao agente cognitivo e os modelos.

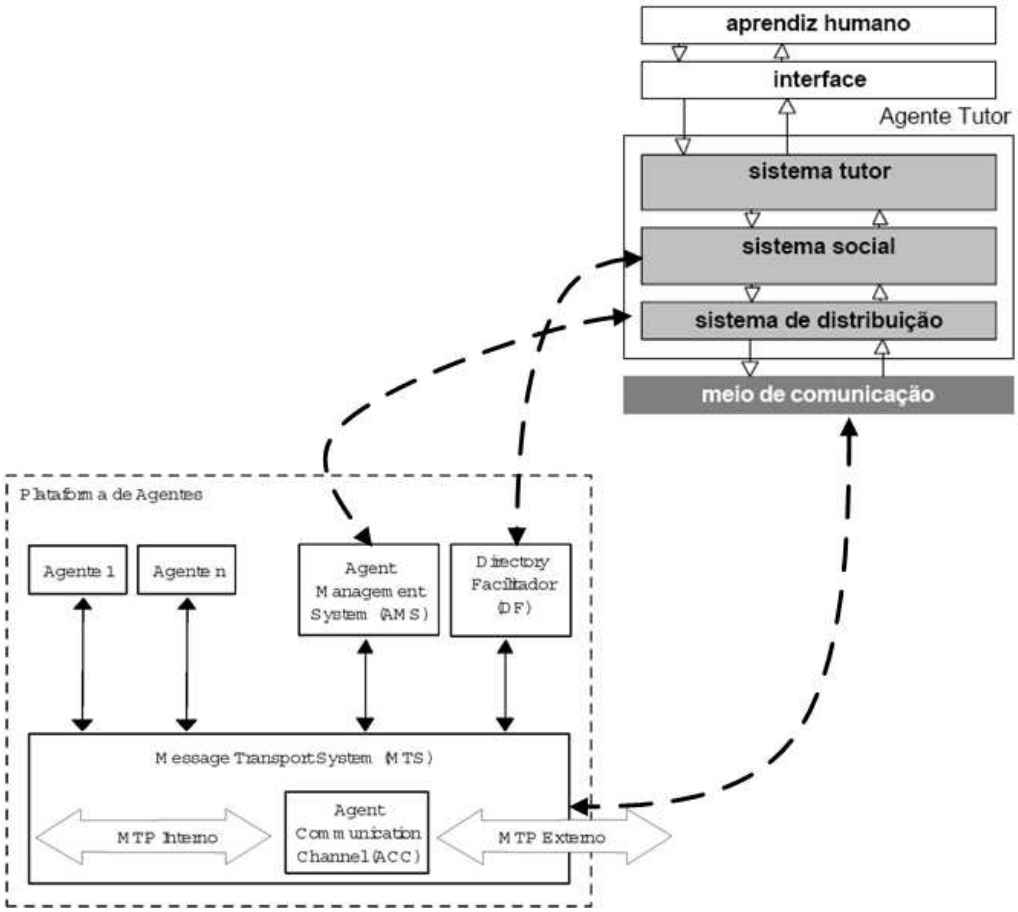

Figura 4.12: Implementação de um ATA em Jade.

Fonte: Extraída e adaptada de (JR., 2005).

4.4.1 Modelos

Os modelos equivalem à abordagem clássica de sistemas tutores inteligentes, sendo composto pelo modelo do domínio, estudante e pedagógico. Além destes, adiciona-se o modelo colaborativo, como mostrado na Figura 4.13

Os Agentes tutores autônomos foram modelados baseados no modelo MATHEMA (COSTA, 1997), através da construção de uma ontologia. Frisa-se ainda ser a ontologia construída segundo a integração com os trabalhos (DILLENBOURG; SELF, 1992; CHEN; MIZOGUCHI, 2004).

Objetivando construir uma ontologia para ambientes interativos de aprendizagem, propôs-se neste trabalho uma solução utilizando OWL-DL, na ferramenta Protégé, onde diversos aspectos foram levados em consideração, dentre os quais podem ser citados: *i) modelo do domínio:* são as informações relevantes ao que é ensinado; *ii) modelo do estudante:* res-

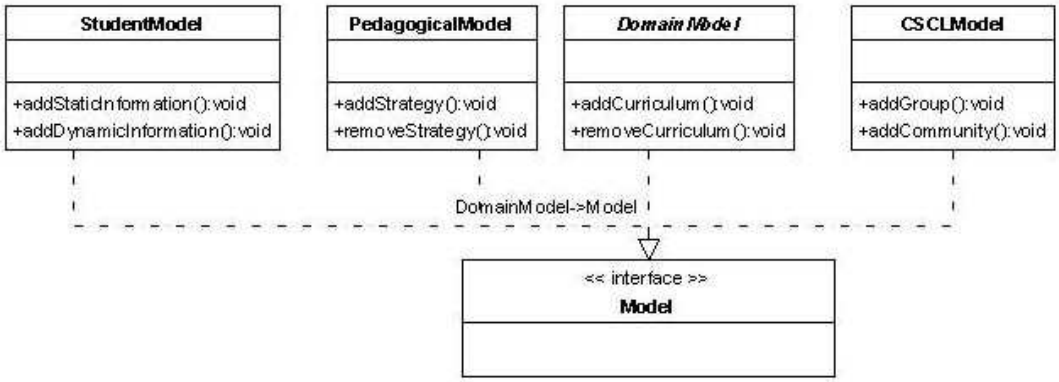

Figura 4.13: Diagrama de Classes dos modelos

ponsável pelo conhecimento sobre quem é ensinado; *iii) modelo de colaboração:* equivale ao tipo de informação relevante para o aprendizado em grupo, ou seja, como os estudantes irão interagir, objetivando a aprendizagem; *iv) modelo pedagógico:* representa a informação sobre como é conduzida à interação, objetivando o aprendizado do estudante tanto de forma individualizada quanto em grupo.

A seguir, detalhes referentes aos modelos presentes na ontologia proposta. A correlação entre os modelos segue a abordagem padrão de ambientes interativos de aprendizagem/sistemas tutores inteligentes.

Modelo do Domínio

O modelo do domínio de um sistema educacional é responsável pelo conhecimento sobre *o que ensinar*. Dentre as preocupações referentes ao módulo do domínio, citam-se:

- *Representação do conhecimento do domínio*: é a forma com que o conhecimento alvo a ser ensinado é estruturado e organizado. A Figura 4.14 ilustra a organização da representação do conhecimento do domínio;

- *Resolução de Problemas*: uma característica importante presente na construção de sistemas educacionais é a capacidade de resolução de problemas. Como abordado nas considerações preliminares do Capítulo 4, um problema está associado a uma unidade pedagógica. Além disto, possui um plano de conhecimento e comportamental para subsidiar o processo de avaliação. Para o conhecimento comportamental, diversas técnicas da Inteligência Artificial podem ser utilizadas, entretanto, a escolha dessas

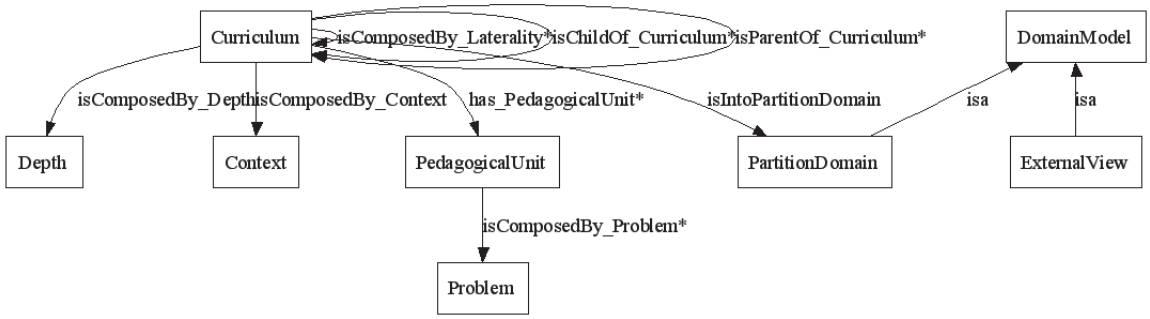

Figura 4.14: Ontologia do Modelo do Domínio

técnicas está diretamente ligada ao tipo de problema que se pretende resolver e ao domínio de ensino contextualizado. A Figura 4.15 ilustra a organização dos problemas.

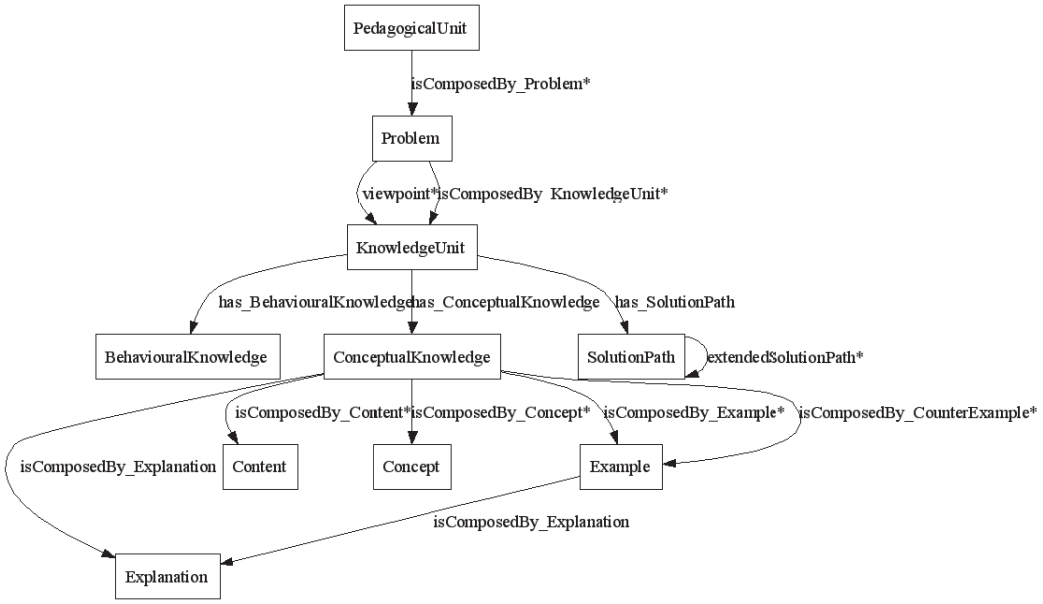

Figura 4.15: Organização dos Problemas na Ontologia

Com a ontologia do domínio, o sistema educacional é capaz de responder perguntas, como:

1. O que está sendo ensinado?
2. Qual o currículo que está sendo ensinado?
3. Quais são os objetos de aprendizagem presentes no currículo?
4. Quais são os recursos de aprendizagem (textos, lista de termos, exemplos,...) presentes no currículo?

5. Quais problemas estão presentes em cada currículo?
6. Qual o nível de dificuldade (fácil, médio, difícil) do problema?
7. Qual é o tipo de problema (Objetivo, Subjetivo, Múltipla escolha, ...)?
8. Qual é a solução do problema?
9. Qual é o Conhecimento Conceitual (termos, exemplos, contra-exemplos, problemas similares,...) envolvido no problema?
10. Qual é o Conhecimento Comportamental (Mecanismo de Resolução do problema, por exemplo, regras) envolvido no problema?
11. Quais são os erros, *bugs* e faltas de conceitos mais comuns na resolução de um problema?

Modelo do Estudante

O modelo do estudante é responsável pelo conhecimento sobre *para quem ensinar*, ou seja, esse módulo contém informações sobre o estudante que é ensinado. Dentre as informações relevantes ao modelo do estudante, têm-se:

- Informações Estáticas: são as informações do estudante que, em princípio, não mudam de acordo com a interação estudante-sistema, como mostrado na Figura 4.16. Algumas informações estáticas são, nome, telefone, endereço, login, senha, características de personalidade, estilo de aprendizagem, entre outros;

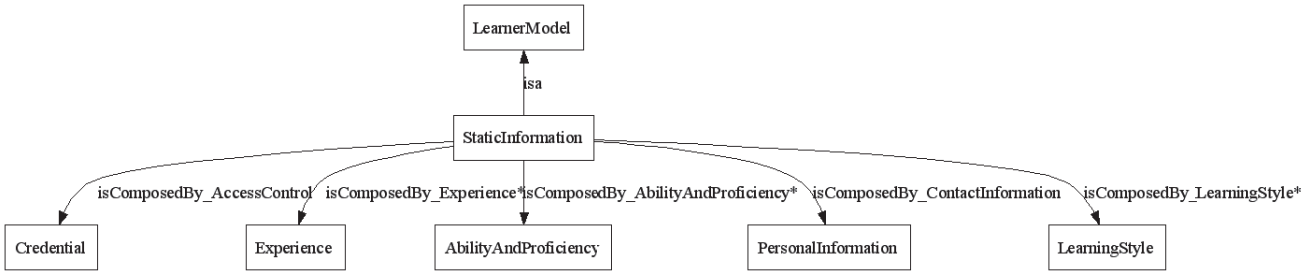

Figura 4.16: Tipos de Informações estáticas relevantes do estudante

- Informações Dinâmicas: são as informações do estudante que mudam durante a interação estudante-sistema. Normalmente, estas informações estão relacionadas com

informações do domínio, como o diagnóstico cognitivo do estudante. Porém, podem se estender ao modelo pedagógico, além dos aspectos emocional e afetivo (Triste, alegre, motivado, entre outros) do estudante, como mostrado na Figura 4.17. Algumas informações dinâmicas são: desempenho geral do estudante, nível de conhecimento atual do estudante, variações de estados mentais, etc.

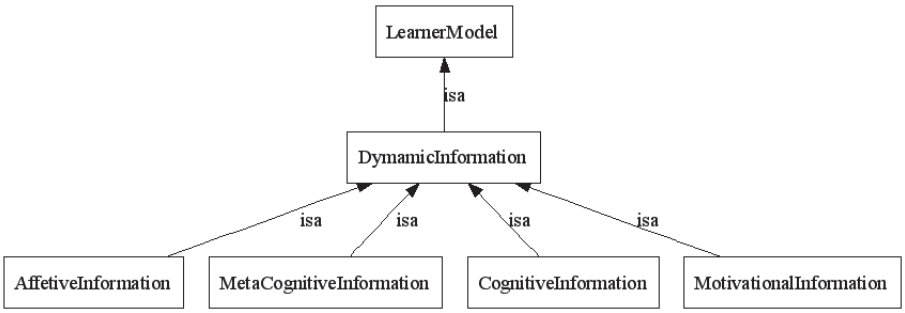

Figura 4.17: Tipos de Informações dinâmicas relevantes do Estudante

A construção do modelo do estudante (Figuras 4.16, 4.17 e 4.18) foi feita através da avaliação dos trabalhos (CHEN; MIZOGUCHI, 2004; CHEPEGIN,). Com isso, diversas perguntas podem ser feitas, algumas delas são:

1. Qual o nome do estudante?
2. O estudante é do sexo masculino ou feminino?
3. Qual a idade do estudante?
4. Qual a experiência profissional do estudante?
5. Quais habilidades (escrita, leitura,...) o estudante possui?
6. Qual o estilo de aprendizagem (Orientado a princípios, orientado a exemplo, do geral para o específico, questões de múltipla escolha,...) preferido do estudante?
7. Qual o nível educacional do estudante?
8. Quais são as características (curioso, cooperativo, imaginativo, reservado, criativo,...) motivacionais e afetivas do estudante?
9. Quais são os estados mentais (depressivo, irritação, pressão de tempo,...) do estudante?

10. Quais são os estados emocionais (ansioso, confuso, excitado, triste, satisfeito,...) do estudante?
11. Qual a personalidade (extrovertido, introvertido, sentimental, nervoso, otimista, pensativo, controlado,...) do estudante?
12. O que exatamente o estudante quer aprender?
13. Quais unidades o estudante já estudou?
14. Qual a unidade que o estudante está estudando?
15. O que o estudante já sabe?
16. O que o estudante ainda não sabe?
17. Quantas questões o estudante acertou?
18. Quantas questões o estudante errou?
19. Qual o tempo médio de resolução dos problemas?
20. Qual conhecimento conceitual o estudante utilizou na resolução do problema?
21. Qual conhecimento comportamental o estudante utilizou na resolução do problema?

As informações dos estudantes podem ser adquiridas, tanto de forma explícita (sistema solicita informações ao estudante), quanto de forma implícita (sistema infere sobre o estudante).

Modelo de Colaboração

O modelo de colaboração refere-se às características identificadas no processo de colaboração entre os estudantes do sistema. Atualmente, a abordagem de CSCL é bastante utilizada nos ambientes educacionais vigentes. A construção do modelo de colaboração (Figura 4.19) foi feita através da avaliação dos trabalhos (BARROS et al., 2002; BARROS; VERDEJO; MIZOGUCHI, 2001).

O processo de colaboração é guiado através do uso de comunidades virtuais de aprendizagem, com isso, certos aspectos devem estar presentes nas comunidades, como as ferramentas

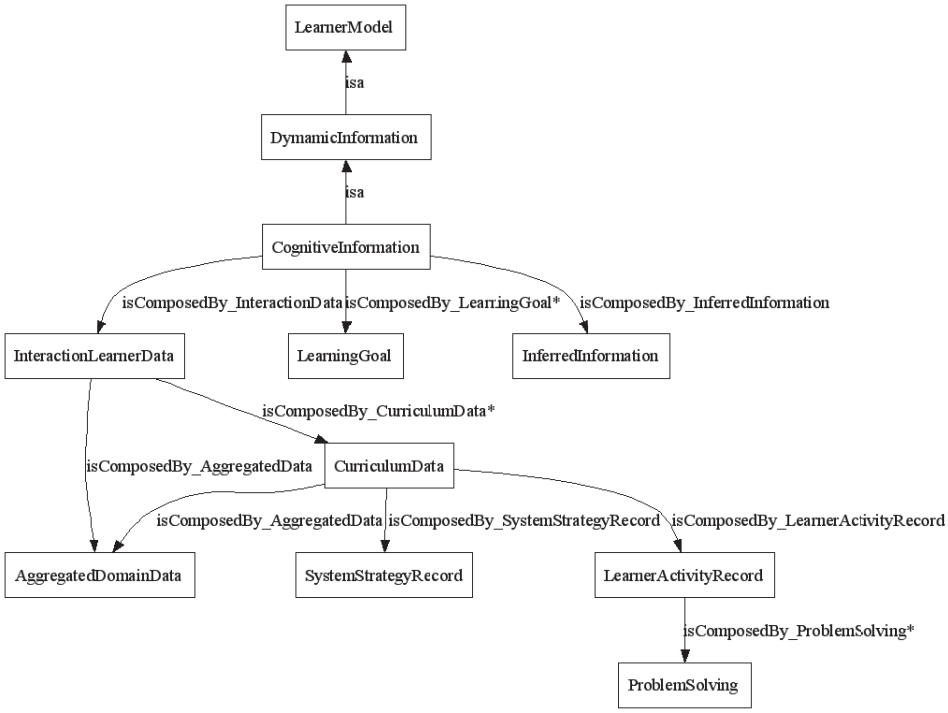

Figura 4.18: Tipos de Informações dinâmicas relevantes do estudante

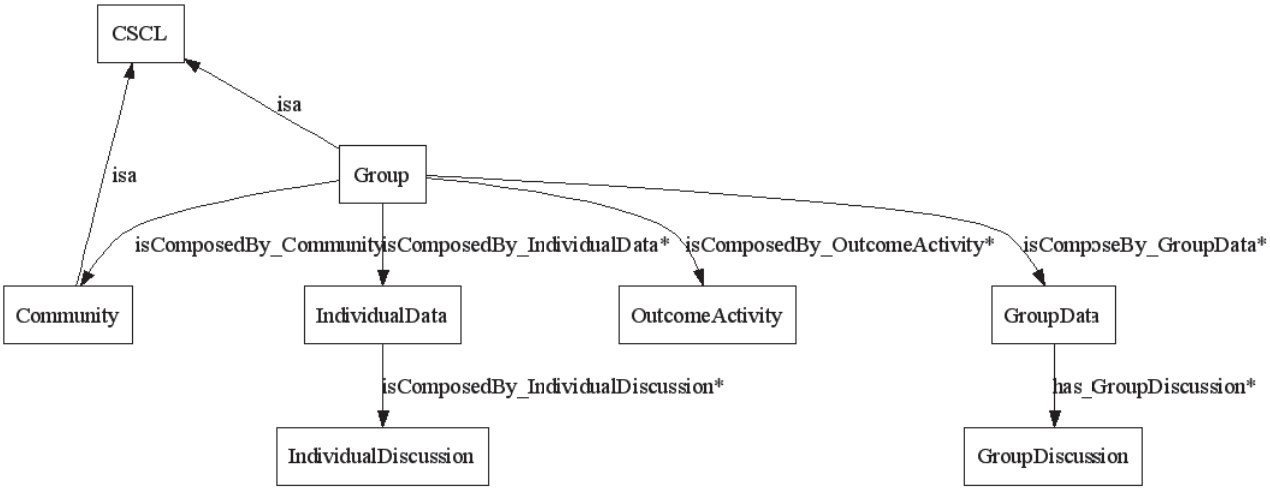

Figura 4.19: Informações referentes ao modelo de colaboração

de uso na comunidade, as normas que regem a comunidade e atividades relacionadas com o tema/descrição da comunidade.

Além disto, uma comunidade pode ser dividida em grupos de aprendizagem, por exemplo, grupos de alunos de medicina da Universidade Federal de Alagoas e outro grupo com os alunos de medicina da Universidade Católica de Brasília.

Com isso, diversas perguntas podem ser feitas, algumas delas são:

1. Quais são as comunidades presentes?
2. Quais são os grupos de cada comunidade?
3. Quais estudantes estão em determinado grupo?
4. Quais são as normas (responsabilidades do sistema) que regem determinada comunidade?
5. Quais são as ferramentas presentes na comunidade?
6. Quais foram as interações dos alunos?
7. Como está o desempenho do aluno em determinado grupo?
8. Quais os atuais temas de discussão do grupo?

Modelo Pedagógico

O modelo pedagógico é responsável pelo conhecimento sobre *como ensinar*, ou seja, como uma interação pode ser desenvolvida. Normalmente, tal modelo realiza esta interação através de um planejamento instrucional, considerando aspectos cognitivos sobre os estudantes.

A construção do modelo pedagógico (Figura 4.20) foi feita através da avaliação dos trabalhos (BOULAY; LUCKIN, 2001; KUMAR et al., 2004; MAJOR; AINSWORTH; WOOD, 1997).

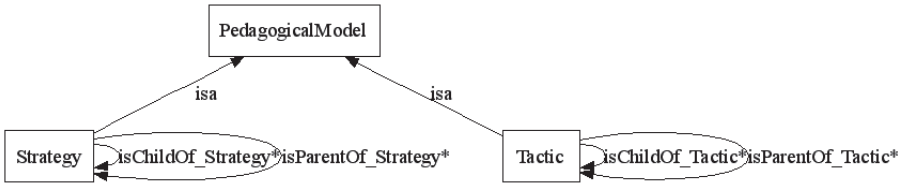

Figura 4.20: Ontologia do Modelo Pedagógico

Plano Instrucional O plano instrucional refere-se a forma que o conteúdo é ensinado, fazendo uso de estratégias pedagógicas, para o estudante ou grupo de estudantes. Para definição do plano instrucional, faz-se uso da ontologia do domínio, através da definição das estratégias pedagógicas e táticas inerentes ao domínio.

A Figura 4.21 define a construção do plano instrucional.

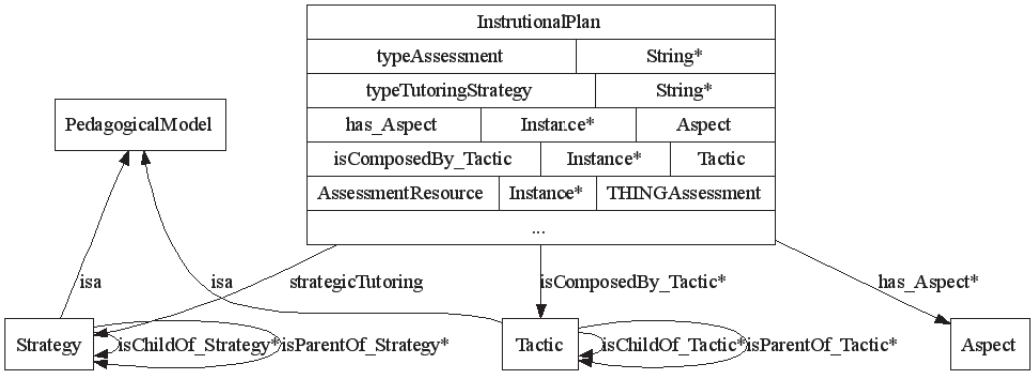

Figura 4.21: Ontologia do Plano Instrucional

4.4.2 Agente Cognitivo

O agente cognitivo possui quatro funcionalidades básicas (como mostrado na Figura 4.11), sendo elas:

1. *Tutor*: equivale a tutoria. Este método é chamado automaticamente quando o agente é requisitado para interagir com um estudante. Quando o sistema inicia, um dos agentes tutores será recomendado para iniciar o processo de tutoria. Existem diversas formas do processo de tutoria ocorrer, entretanto, tais variações são configuradas/especificadas através da ontologia citada nesta seção;

2. *SolveProblem*: é responsável pela implementação do processo de resolução de problemas;

3. *Diagnostic*: responsável por diagnosticar alguma resolução de problemas, ou seja, este funcionalidade identifica qual foi o tipo problema (caso o estudante não acerte a resolução de algum problema) pode ter ocorrido com o estudante no processo de resolução de algum problema;

4. *Assessment*: equivale a avaliação do estudante, podendo ter diversas implementações diferentes, como por exemplo, a avaliação através de uma prova, exercícios em grupos, dentre outros.

Resolução de Problemas

A resolução de problemas é um aspecto bastante explorado pela comunidade de *AI-ED*. Porém, este aspecto possui bastante diversidade, tanto na modelagem de problemas quanto no processo de resolução. A resolução de problemas[3] é uma característica importante que está presente na construção de sistemas educacionais. Para isso, diversas técnicas de Inteligência Artificial podem ser utilizadas.

Os problemas das aplicações podem variar desde questões simples (múltipla escolha, relacionar colunas, preencher lacunas, entre outras) até estruturação do conhecimento do problema de forma bastante complexa. Um exemplo que pode ser dado é a forma que um problema é abordado no domínio do direito (será abordado em detalhes no Capítulo 5), podendo ser uma questão simples, ou bastante complexa, como a definição de um problema como um conjunto de relações definidas.

Além disso, não somente o problema pode variar, mas também a própria solução. O mesmo exemplo anterior é dado, onde uma solução pode variar segundo uma visão do promotor ou do advogado de defesa.

Estes dois aspectos (Problema, Solução) tornam o processo de resolução de problemas bastante restrito ao domínio que se está trabalhando, e conseqüentemente, tornando-se um ponto de extensibilidade do arcabouço.

[3]A diferença entre a resolução de problemas do Agente Mediador e cada Agente Tutor Autônomo é que, enquanto o AM coordena o processo de resolução de problemas (utilizando vários agentes tutores autônomos), os ATAs possuem conhecimento para resolver uma tarefa do problema, ou seja cooperando.

4.5 *Agentes de Suporte - AS*

Os agentes de suporte correspondem a agentes com características de inferir de acordo com um determinado mecanismo pré-concebido. Para que tais agentes sejam utilizados, duas medidas têm que ser tomadas: *i)* definir qual mecanismo de inferência será construído; *ii)* associar um mecanismo de inferência com um agente e, conseqüentemente, fazer a chamada do agente.

Com isso, o autor/professor pode atuar como um engenheiro do conhecimento para configuração de determinado mecanismo de inferência que deseje ser utilizado.

4.5.1 Mecanismos de Inferência

Os agentes de suporte permitem a integração de diversas técnicas de Inteligência Artificial, objetivando a resolução de problemas.

Visando facilitar o desenvolvimento de ambientes educacionais inteligentes, dois dos mecanismos de inferência mais utilizados pela comunidade de *AI-ED* foram implementados e disponibilizados no arcabouço.

Dentre os algoritmos presentes atualmente em AS, encontram-se:

- Raciocínio Baseado em Casos (RBC): trata-se de uma abordagem que se utiliza de um tipo de raciocínio analógico, contrastando com o convencional paradigma de resolução de problemas baseado em regras. Portanto, a abordagem para a solução de problemas no Raciocínio Baseado em Casos é baseada na resolução de problemas valendo-se de experiência passada (BITTENCOURT et al., 2006a);

- Raciocínio Baseado em Regras (RBR): trata-se de uma abordagem para solucionar problemas. Para isto, o sistema precisa ter uma base de regras consistente armazenada, por exemplo, em um banco de dados e, também, de fatos declarados ou obtidos para se chegar à conclusão do problema. Um sistema baseado em regras é composto por base de conhecimento, ferramenta de inferência e o subsistema de explanação;

A Figura 4.22 mostra o diagrama de classes dos agentes de suporte.

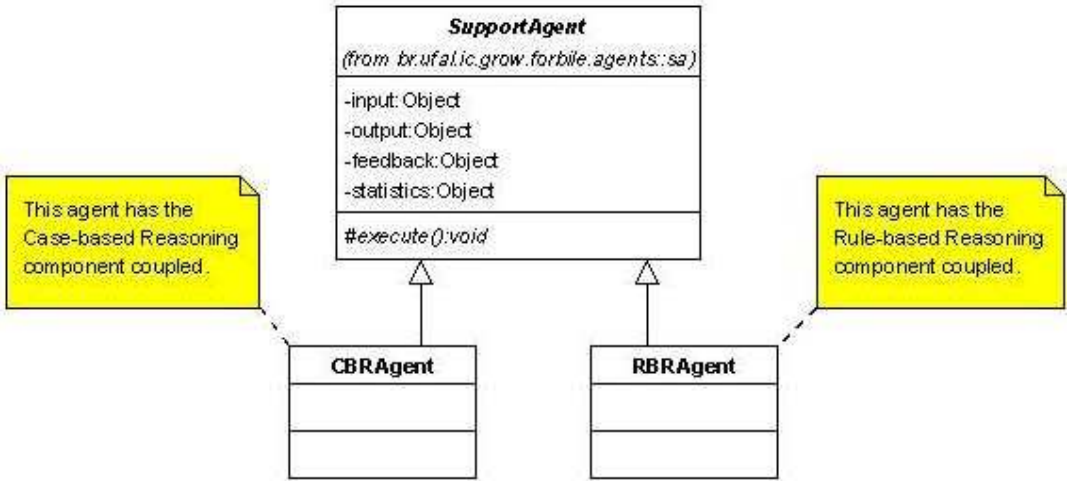

Figura 4.22: Diagrama de Classes dos Agentes de Suporte

4.5.2 Agentes de Raciocínio Baseado em Casos

Segue nesta seção um passo-a-passo sobre a utilização do componente RBC. Os passos são:

- **Passo 1**: O usuário seleciona a base de casos;
- **Passo 2**: O usuário seleciona os atributos;

Figura 4.23: Passos para seleção da Base de Casos e atributos

- **Passo 3**: O usuário define o peso de cada atributo. O valor padrão dos pesos é 1, pois o usuário pode não querer diferenciar os atributos;
- **Passo 4**: Logo após são definidas as funções de similaridade local, ou seja, o cálculo de similaridade entre os atributos. Além disso, é definida nesta fase a função de similaridade global, ou seja, o cálculo de similaridade entre os casos;

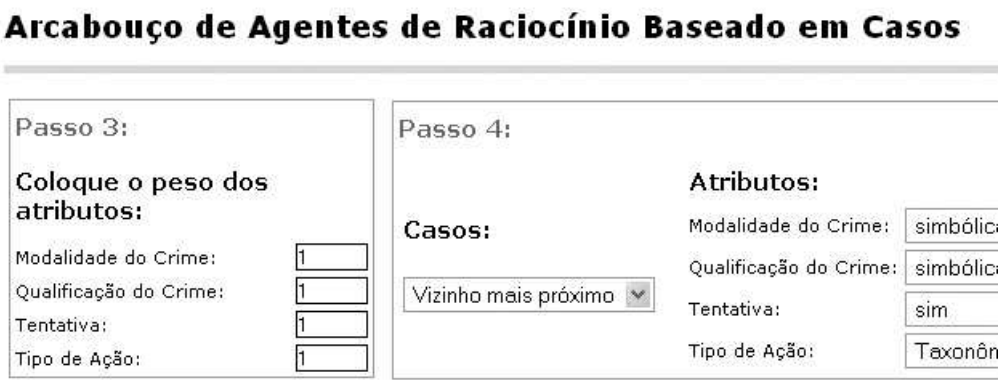

Figura 4.24: Passos para definição dos pesos de cada atributo e das funções de similaridade

- **Passo 5**: Define a forma de recuperação dos casos, podendo ser seqüencial, dois níveis, entre outras;
- **Passo 6**: Tela para definir um arquivo Java para a fase de reúso e revisão dos casos similares. Estas fases não são disponibilizadas na plataforma em forma de configuração padrão, devido à grande especificidade das mesmas;
- **Passo 7**: Define o limiar de retenção do caso.

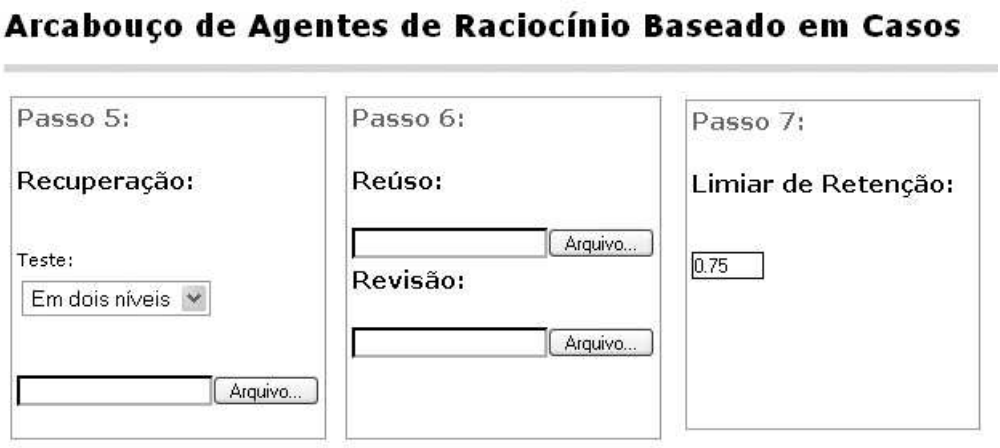

Figura 4.25: Passos para definição da forma de recuperação, reúso e revisão e limiar de retenção

4.5.3 Agentes de Raciocínio Baseado em Regras

Segue-se abaixo o passo-a-passo do agente de raciocínio baseado em regras.

- **Passo 1**: o usuário registra as variáveis e seus respectivos valores;
- **Passo 2**: as regras específicas do domínio são criadas;
- **Passo 3**: é escolhido o mecanismo de inferência, podendo ser encadeamento para frente e encadeamento para trás;
- **Passo 4**: equivale a uma explanação relacionada a uma conclusão, ou seja, para cada conclusão, associa-se uma explanação.

4.5.4 Integração de técnicas

A integração das técnicas ocorre de forma dinâmica através da utilização da ontologia ilustrada na Figura 4.26. Para isso, quatro tipos de informações têm que ser levadas em consideração para cada algoritmo:

1. Entrada/Saída: equivale aos dados de entrada e saída do algoritmo e seus tipos.
2. Processamento: refere-se à forma na qual o algoritmo é processado internamente. Cada fase do Ciclo 4R presente em um algoritmo de RBC seria um exemplo.
3. *Feedback*: equivale a um retorno dado para o algoritmo após determinado processamento. Este *feedback* pode se dar de forma síncrona ou assíncrona, conseqüentemente, tanto o usuário quanto outro agente pode dar esta informação para o agente.
4. Estatística: equivale a dados pré-estabelecidos para avaliar o desempenho do algoritmo.

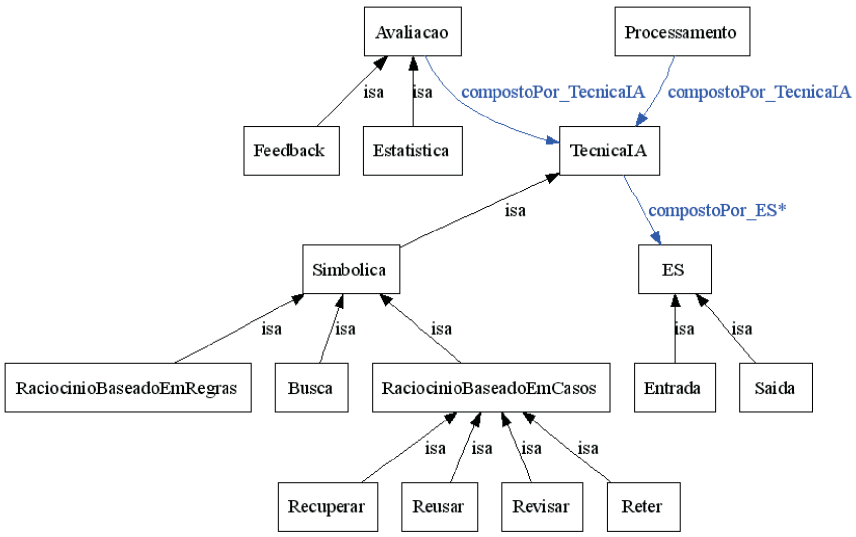

Figura 4.26: Ontologia Das Técnicas de Inteligência Artificial

4.6 *Agente de Persistência - AP*

O Agente Mediador e a Sociedade de Agentes, apresentados nas Seções 4.3, 4.4 e 4.5, possuem características inerentes ao processo de ensino-aprendizagem. Porém, há a necessidade de armazenamento das informações de interação.

Os Agentes de Persistência são responsáveis pela persistência dos dados e recuperação do conhecimento das interações dos diversos papéis presentes na plataforma.

Os agentes de persistência inerentes ao ambiente são:

- *PersistenceEMATHEMAAgent*: é responsável por persistir o dados referente aos modelos discutidos na Seção 4.4.1;
- *PersistenceOWLAgentInteraction*: agente que persiste dados inerentes a interação entre os agentes;
- *PersistenceSupportAgent*: equivale ao agente responsável por recuperar dados referentes a integração de algoritmos inteligentes.

Para que estes agentes possam fazer parte do ambiente, eles devem estender da Classe *PersistenceAgent*. Esta agente possui os comportamentos inerentes a qualquer agente de persistência, sendo ele, criação, recuperação, atualização e deleção. A Figura 4.27 aborda o diagrama de classes dos agentes de persistência.

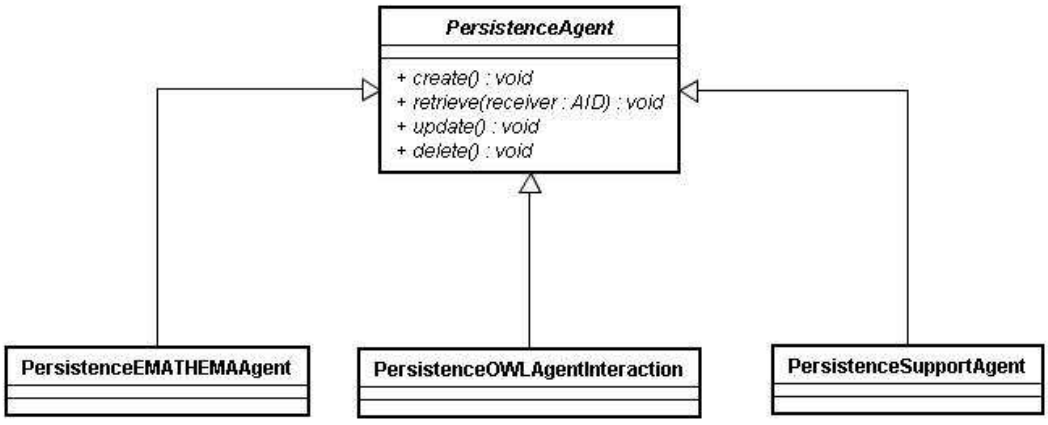

Figura 4.27: Diagrama de Classes dos agentes de persistência

A camada de infra-estrutura presente na plataforma dá subsídio a utilização de ontologias (Vide Apêndice **??**). As ontologias foram construídas em Protégé e utilizaram a linguagem

OWL-DL. O Protégé possui um *plugin* (Vide Apêndice A) que faz o mapeamento da ontologia em classes Java.

Este capítulo abordou sobre a concepção de uma plataforma para construção de ambientes interativos de aprendizagem. Foram abordadas funcionalidades de cada agente na plataforma, de modo a poderem ser explorados de forma mais efetiva nos estudos de casos que seguem.

Capítulo 5

Estudos de Casos

Objetivando experimentar e validar a aplicabilidade da plataforma proposta neste trabalho, este capítulo aborda a utilização da plataforma no desenvolvimento de dois sistemas tutores, sendo um na área de direito e outro de medicina. Com isso, este capítulo apresenta as facilidades providas pela plataforma.

5.1 Estudo de Caso 1: Sistema *THEMIS*

5.1.1 Análise de requisitos

O Sistema *Themis* aborda, dentre outras funcionalidades, um sistema tutor inteligente aplicado ao domínio jurídico, provendo os estudantes com casos reais, leis e diferentes pontos de vistas[1] sobre um determinado conhecimento (BITTENCOURT et al., 2006d). Além disto, a tutoria presente no sistema refere-se aos Crimes Contra a Vida do Código Penal Brasileiro (JESUS, 2005).

A idéia central é engajar estudantes de direito na interação com o sistema baseado na resolução de problemas legais, além de explorar outras atividades de ensino. A interação ocorre de duas formas: i) quando o sistema envia um conteúdo do assunto e um problema para o estudante responder e ii) quando o estudante envia um problema para o sistema resolver (BITTENCOURT; COSTA, 2005).

Com isso, foram verificadas as seguintes necessidades no sistema:

[1] Entende-se como pontos de vistas as diversas interpretações relacionadas ao domínio do direito.

- O processo de aprendizagem dos estudantes se dá baseado em três tipos de conhecimentos (maiores detalhes conceituais referentes aos conhecimentos abordados abaixo, vide (ACQUAVIVA, 1988)):

    1. Legislação: equivale a um recurso jurídico, onde o magistrado toma suas decisões baseados em códigos e leis, originando uma solução. A Legislação pode ser definida como uma declaração de que algum ato *deve*, *não deve*, *pode* ou *não pode* em uma sociedade (KRALINGEN, 1997);
    2. Jurisprudência: são os casos que já foram julgados pela sociedade ou órgãos competentes;
    3. Doutrina: representa uma fundamentação teórica sobre determinado assunto ou termo jurídico. Tal fundamentação é descrita por profissionais renomados da área, abordando uma visão pessoal sobre o termo.

- A interação ocorre entre o Estudante e o Tutor;
- O início do processo de aprendizagem ocorre através do envio de conceitos ao estudante, seguido de problemas relacionados;
- O sistema avalia a resposta do estudante e, dependendo da resposta, inicia o processo de explicação;
- O Estudante pode solicitar a resolução de algum problema ao sistema.

Com o processo de ensino finalizado, o estudante obtém duas habilidades fundamentais no processo de resolução de problemas. Primeiro, saber como identificar os casos e seus conceitos legais. Segundo, saber como usá-los de forma efetiva como exemplos para justificar uma interpretação jurídica (BITTENCOURT; COSTA, 2006).

Com os aspectos fundamentais do Sistema *Themis* explicitados, necessita-se definir os requisitos funcionais e mapeá-los nos casos de uso. Os casos de uso são interações entre os atores e o sistema (maiores detalhes em (ALHIR, 2003)). Com isso, na Figura 5.1 são mostrados os casos de usos do sistema quando o usuário se iniciar no sistema.

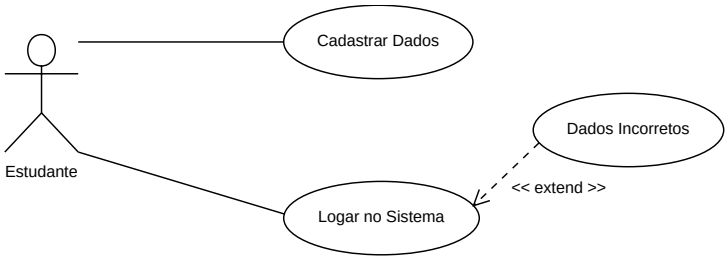

Figura 5.1: Casos de uso iniciais do Sistema Themis

Com o estudante conectado ao sistema, o processo de ensino-aprendizagem começa. Os demais requisitos, porém não menos importantes, estão definidos como mostrado nos casos de uso da Figura 5.2.

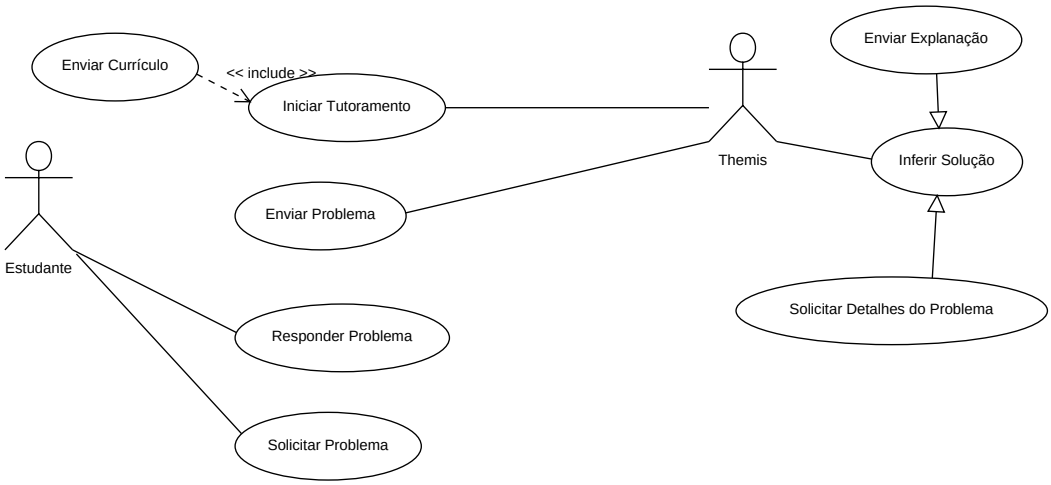

Figura 5.2: Casos de uso de Tutoria do Sistema Themis

Dentre os componentes presentes no sistema *Themis*, têm-se:

- Interface do Usuário: é responsável por permitir comunicação entre o estudante e o sistema tutor;
- Mecanismo Cognitivo: representa o núcleo do sistema tutor inteligente, onde cada agente age em um subdomínio;
- Ontologia: representa a base de conhecimento de ensino;

Abaixo seguem detalhes referentes aos componentes presentes na arquitetura.

5.1.2 Ontologia

A ontologia equivale a base de conhecimento sobre o domínio de direito. Dentre as informações presentes na ontologia, têm-se:

- Conteúdo Pedagógico: equivale às informações pedagógicas, como o conteúdo que é ministrado pelo sistema tutor;
- Problemas: refere-se a problemas que guiarão o processo de ensino-aprendizagem;
- Plano Instrucional: equivale ao mecanismo de interação estudante-tutor.

Conceitos relevantes ao Código Penal Brasileiro foram considerados para especificação do Conteúdo Pedagógico, como proposto na Tabela 5.1.

Tabela 5.1: Dados do Domínio de Direito - Código Penal Brasileiro

Classe	Conteúdo
DomainModel	Direito
LearningResource(Content)	Introdução
PartitionDomain	Direito Penal
Currículo $_1$	Artigo 121
Unidade Pedagógica $_1$	Geral
Problema $_1$	Fácil
Problema $_1$	Difícil
...	...
Currículo $_{12}$	Homicídio Qualificado
Unidade Pedagógica $_1$	Envenenamento
...	...
Currículo $_2$	Artigo 122
Unidade Pedagógica $_1$	Lesão Corporal Grave
...	...
Currículo $_8$	Artigo 128
Unidade Pedagógica $_1$	Estupro
...	...

Na Tabela 5.1, pode-se observar que existem currículos filhos de outros currículos. Esta relação hierárquica pode ser interpretada como pré-requisito. Além do pré-requisito hierárquico, outros pré-requisitos podem ser definidos através da visão de lateralidade (como abor-

dado na Seção 2.4.1). Com isso, observa-se também que o domínio é formado por oito currículos principais, equivalendo aos artigos do código penal (dos crimes contra a vida).

Um aspecto importante no domínio de direito equivale à especificidade dos problemas (BITTENCOURT et al., 2006c). Um problema é definido como uma 3-Tupla $\langle P, I, F \rangle$, onde:

- *P*: representa uma situação penal real;
- *I*: representa um conjunto de interpretações (soluções) do problema *P*. As interpretações são baseadas em duas visões: i) Visão do Advogado e ii) Visão do Promotor de Justiça;
- *F: P x I*: representa uma fundamentação teórica da relação *P x I*, que pode ser uma doutrina, Jurisprudência ou Legislação.

Além disso, outro aspecto que a ontologia é responsável de manter equivale ao conhecimento das estratégias pedagógicas e o plano instrucional. O Sistema *Themis* possui a estratégia pedagógica de Aprendizagem Baseada em Problemas[2](SAVERY; DUFFY, 1995). A partir do momento que a estratégia pedagógica é definida, resta apenas definir os pesos que fazem com que o estudante "passe" de um currículo para outro[3]. Na Tabela 5.2 segue o conhecimento que deve ser definido com relação aos aspectos pedagógicos.

Tabela 5.2: Dados do Domínio de Direito - Código Penal Brasileiro

Classe	Conteúdo
$Strategy_1$	Problem-based Learning
$LearningGoal_1$	$Currículo_1$(Art. 121)
levelKnowledge	*Advanced*
rate	*0.80*
...	...
$LearningGoal_8$	$Currículo_8$(Art. 128)
LevelKnowledge	*Advanced*
rate	*0.80*

Com o conhecimento pedagógico definido na ontologia, deve-se agora definir os mecanismos cognitivos do sistema.

[2]Do Inglês, Problem-Based Learning

[3]O cálculo da nota/taxa do aluno é feita através da quantidade de respostas corretas no processo de resolução de problemas.

Mecanismo Cognitivo

O mecanismo cognitivo equivale ao aspecto cognitivo do sistema educacional. A arquitetura do mecanismo cognitivo é formada pelos agentes tutores, onde cada agente tutor possui os modelos do estudante, pedagógico e do domínio, como mostrado na Figura 5.3

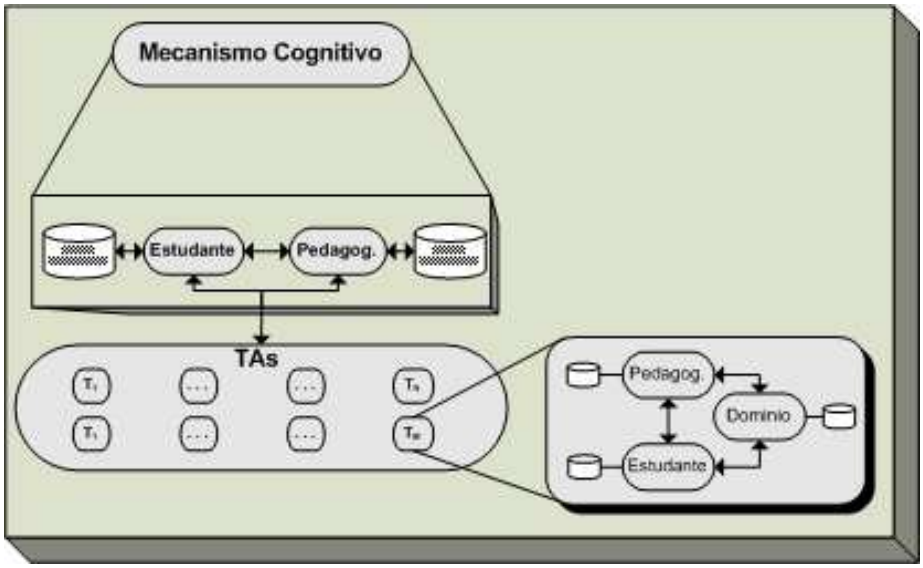

Figura 5.3: Arquitetura do Mecanismo Cognitivo do Sistema *Themis*

Na arquitetura do mecanismo cognitivo, cada agente tutor é responsável pela tutoria de um currículo do modelo do domínio. Com isso, o Agente mediador (presente na arquitetura da plataforma) é responsável por selecionar qual agente tutor que possui competências para tutorar o estudante. A escolha é feita através da lista que o agente mediador possui, contendo a tripla $< Agente, Serviço, Habilidade >$, onde habilidade equivale à lista de características curriculares do tutor.

Além disto, este agente é responsável por coordenar o processo de resolução problemas, pois um problema pode envolver diversos níveis de conhecimento do sistema. Isto faz com que a resolução do problema seja subdividida entre os agentes tutores.

Cada agente tutor possui mecanismos de inferência para resolução de problemas. Os mecanismos definidos no sistema foram Raciocínio Baseado em Casos e Raciocínio Baseado em Regras (mais detalhes sobre os mecanismos, vide Subseções 2.2 e 2.2). Estes mecanismos de inferência foram escolhidos por sua grande aceitação na comunidade de Inteligência

Artificial e Direito[4], para a implementação de sistemas baseados em conhecimento.

5.1.3 Interface do Usuário

A Interface[5] é composta por sete telas, como mostrado na Figura 5.4

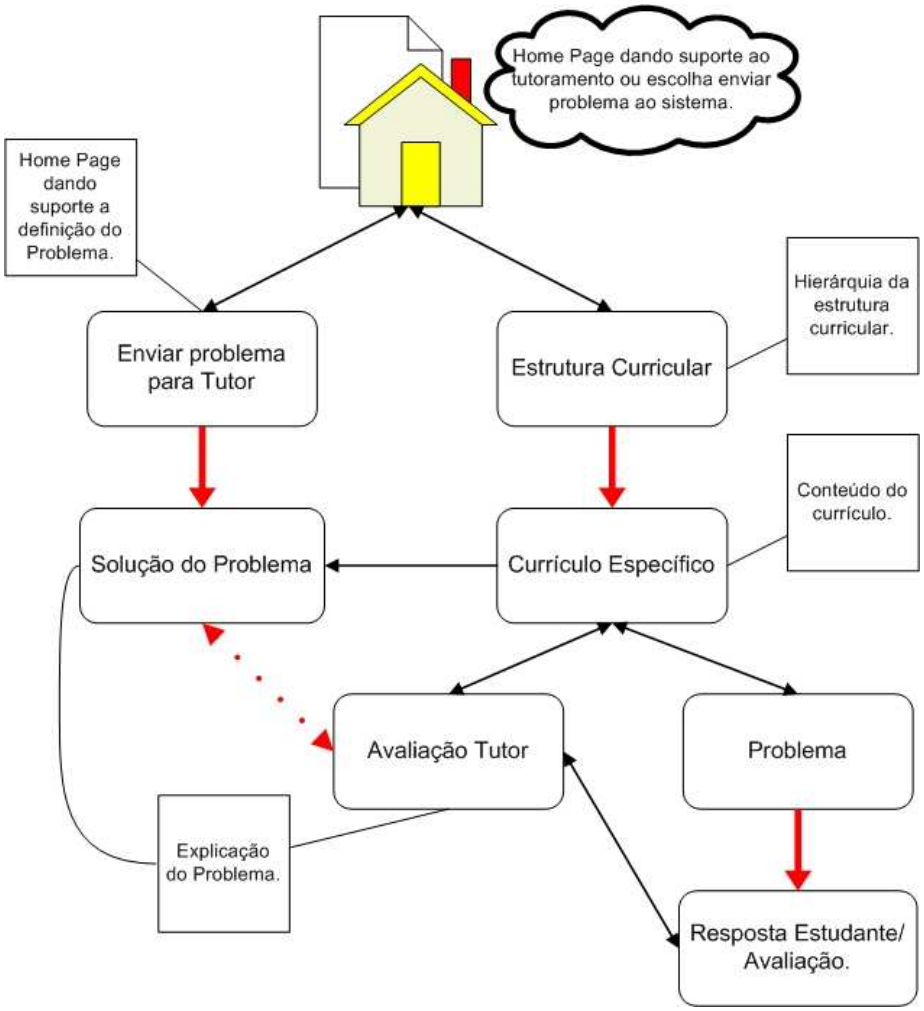

Figura 5.4: Diagrama Web do sistema *Themis*

As telas presentes são:

- Home Page: equivale à página inicial do sistema, onde o estudante se conecta;
- Estrutura Curricular: tela onde é guiado o processo de ensino-aprendizagem. O currículo específico está presente na mesma tela;

[4]Do Inglês Artificial Intelligence and Law.

[5]A plataforma não dá suporte a autoria de interface, entretanto, não é foco deste trabalho a autoria de interface.

- Enviar problema para estudante: esta tela apresenta um problema para o estudante responder;
- Resposta estudante: para cada problema, o estudante define:
  - Visão (Advogado ou Promotor);
  - Artigo (121, 122,...);
  - Parágrafo;
  - Inciso (Opcional);
  - Alínea (Opcional);
  - Item (Opcional);
  - Sub-Item (Opcional).
- Avaliação do tutor: esta tela mostra a solução do estudante e avaliação do sistema tutor. A avaliação ocorre através da comparação com as soluções presentes no problema;
- Enviar problema ao tutor: os passos para a especificação do problema são:
  1. Especificação dos personagens (nome, idade, deficiências, ...);
  2. Relação entre os personagens (pai, mão, filho, irmão, cunhado, amigo, ...);
  3. Especificação do fato: a) Houve o assassinato?; b) quais são os papéis (vítima, assassino, cúmplice, testemunha,...) dos personagens; c) qual foi a arma (objeto cortante/perfurante, veneno, revólver, ...) usada ? ; d) qual foi o motivo (vingança, encomendado,...) do crime ?; e) quais são os estados (embriagado, forte emoção, dormindo,...) dos personagens.
- Solução do problema: a solução do problema se dá através da execução dos algoritmos de raciocínio baseado em casos e raciocínio baseado em regras seguido de sua explanação.

5.1.4 Construção do Sistema

Com o sistema especificado, é necessário definir quais papéis irão interagis com a plataforma, de modo a construir o sistema tutor. A complexidade do domínio de direito faz com

que haja três tipos de responsabilidades diferentes, sendo elas:

1. Adição de nova funcionalidade: a especificidade de um problema jurídico torna necessária a implementação de uma novo forma de resolução de problemas;
2. Configuração de mecanismos de inferência: para que um problema possa ser resolvido, há a necessidade da utilização dos mecanismos de inferência RBC e RBR. Com isso, para que os mecanismos possam ser utilizados, deve-se configurá-los;
3. Especificação dos domínios: na construção de STI, os modelos são peças essenciais para o seu funcionamento, necessitando ser especificados.

Tais responsabilidades remetem à necessidade de interação dos papéis do desenvolvedor, do engenheiro do conhecimento e do autor. O processo de construção é dividido de acordo com as preocupações abaixo:

- Preocupações do Autor:
    1. Conteúdo curricular;
    2. Estratégias Pedagógicas;
    3. Plano Instrucional;
    4. Objetivos de Aprendizagem;
    5. Especificação do problema;
- Preocupações do Engenheiro do Conhecimento:
    1. Mecanismos de Inferência;
- Preocupações do Desenvolvedor:
    1. Implementação do problema.

A plataforma possui tanto o módulo de *framework* quanto o módulo de autoria, ou seja, o desenvolvedor irá utilizar o *framework* enquanto o autor e o engenheiro do conhecimento irão utilizar o módulo de autoria.

Utilização do Framework

O framework será utilizado pelo desenvolvedor para implementar o processo de resolução de problemas. Para tal, o autor deve antes especificar a estrutura do problema jurídico de acordo com o que foi abordado na seção 5.1.1.

As funcionalidades de um problema em direito devem ser estendidas, pois o tipo de problema como foi especificado neste estudo de caso, não é previamente disponibilizado na ontologia. Uma visão anterior do problema era vista como mostrada na Figura 5.5.

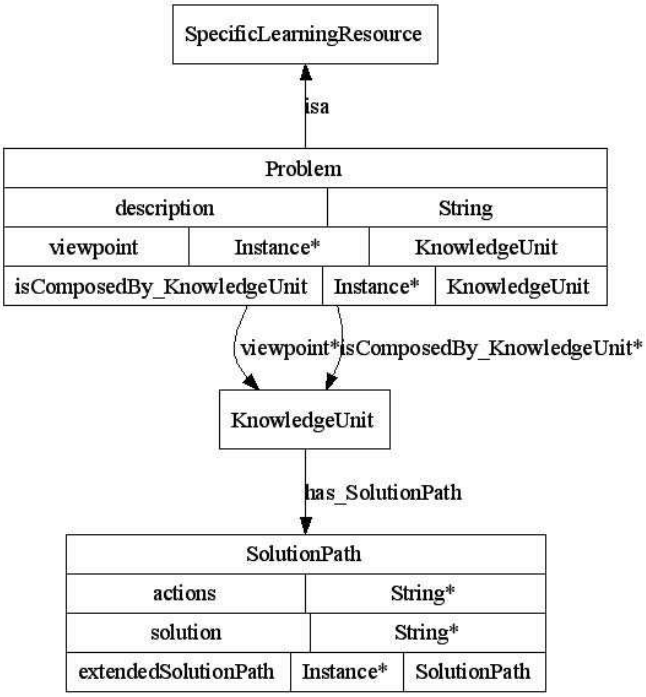

Figura 5.5: Visão anterior de um problema

Características inerentes a problema jurídico não são tratados, motivando, desta forma, a extensibilidade da ontologia para o modelo proposto na Figura 5.6. As classe *Legal* e *CaseProblem* foram construídas para dar suporte a especificações do domínio.

Com a especificação do problema definida, o desenvolvedor possui três preocupações na implementação da nova funcionalidade, sendo elas:

1. Fazer o mapeamento Ontologia-Objeto (maiores detalhes, Vide Apêndice A);
2. Estender o comportamento *SolveProblem* do framework e implementá-lo;
3. Definir na ontologia de interação a implementação padrão da resolução de problemas;

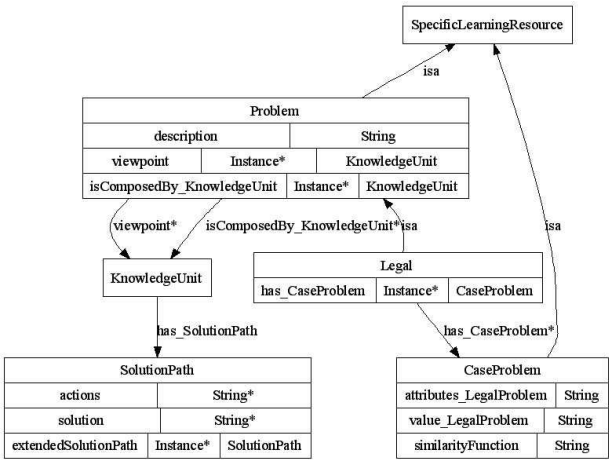

Figura 5.6: Visão de um problema de direito

Mapeamento Ontologia-Objeto Com a nova estrutura da ontologia definida, o desenvolvedor necessita extrair este conhecimento para a aplicação. Tal extração é possível através do mapeamento Ontologia-Objeto, provido pela API do Protégé-OWL. O mapeamento é feito como mostrado na Figura 5.7

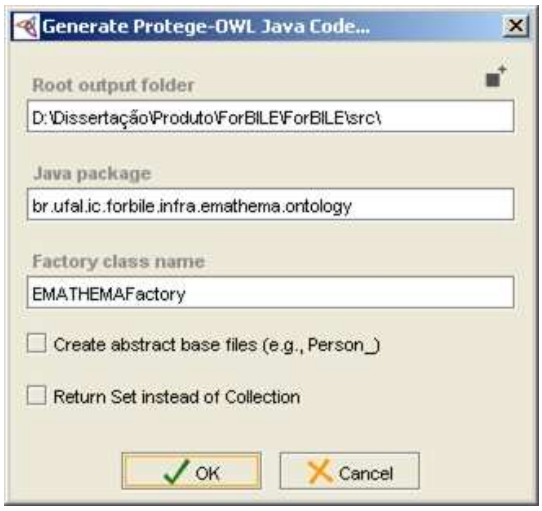

Figura 5.7: Mapeamento Ontologia-Objeto

Classe *Problem* O mapeamento sendo feito da forma supracitada faz com que seja gerado, para cada classe da ontologia, uma *interface* com os métodos de manipulação da classe e uma classe java. A Figura 5.8 aborda o diagrama das interfaces geradas pelo mapeamento Ontologia-Objeto.

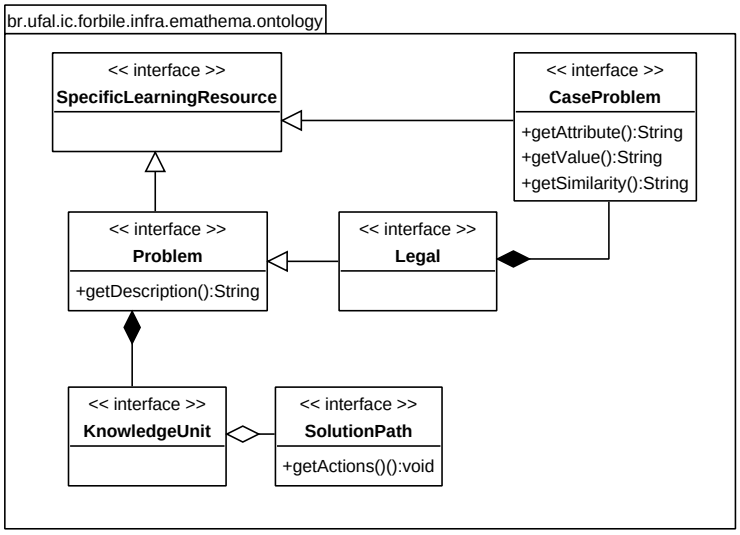

Figura 5.8: Diagrama das interfaces geradas pelo mapeamento Ontologia-Objeto

Para que a resolução do problema seja concretizada, é necessário estender a classe *SolveProblem*. Esta classe equivale a um *Hotspot* da plataforma, onde o desenvolvedor estende a classe e implementa o processo de resolução de problemas utilizando os mecanismos de inferência e a nova estrutura de um problema. O diagrama de classes é mostrado na Figura 5.9

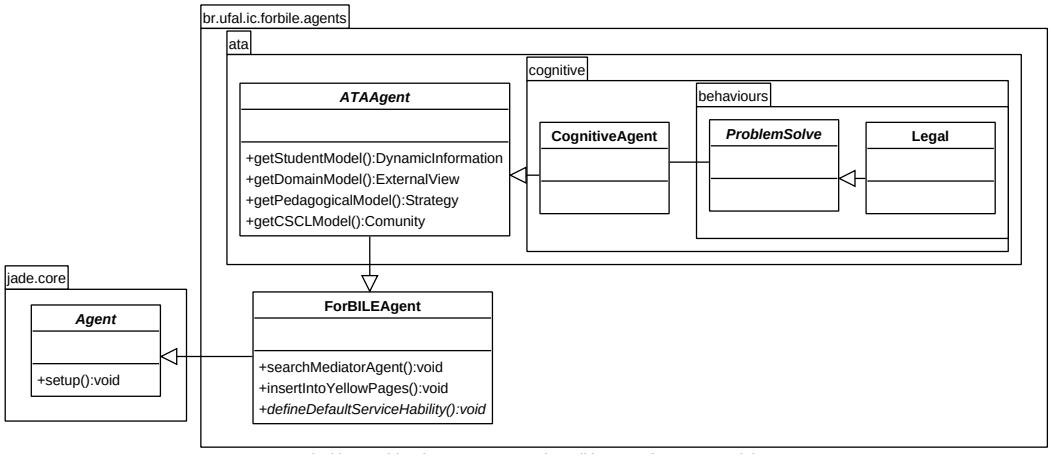

Figura 5.9: Diagrama de classe abordando a extensibilidade da plataforma

Definição da ontologia de interação A ontologia de interação possui todos os serviços e habilidades que são mapeados para cada agente na plataforma. Como uma nova implementação foi feita, é necessário configurar a ontologia definindo que a classe *Legal* (como

mostrado na Figura 5.10) é uma habilidade do agente cognitivo.

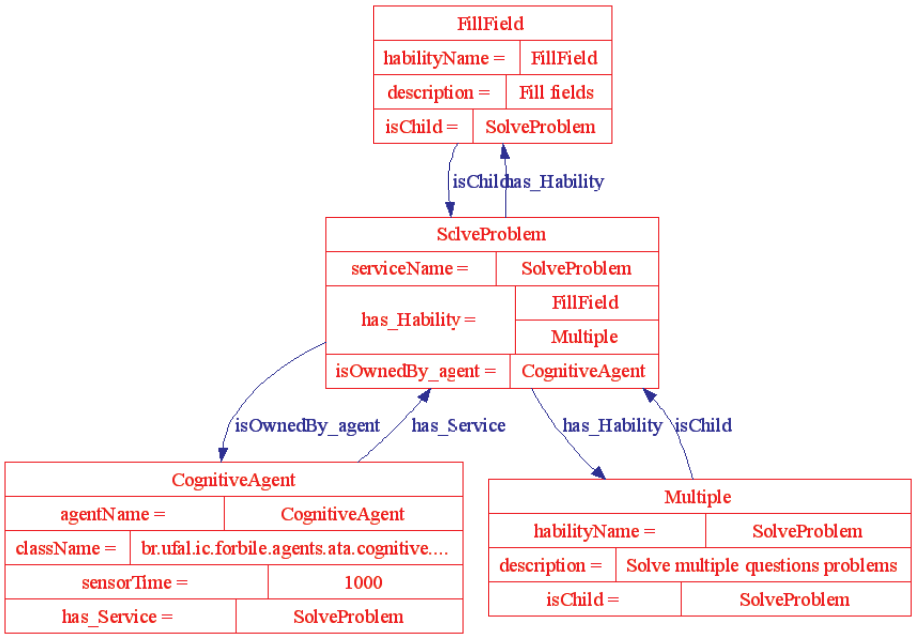

Figura 5.10: Indivíduo da ontologia com a definição da implementação padrão.

Utilização da Ferramenta de Autoria

O módulo de autoria é utilizado tanto pelo autor quanto pelo engenheiro do conhecimento.

Preocupações do Autor Dentre os aspectos de que devem ser enfatizados pelo autor, destacam-se o conteúdo curricular, objetivos de aprendizagem, estratégias pedagógicas e plano instrucional, tendo que ser especificados na ontologia.

Abaixo são abordadas as preocupações do autor no processo de construção do conhecimento do domínio de Direito Penal.

Conteúdo curricular Dentre as informações referentes ao conteúdo curricular, têm-se informações sobre os artigos do código penal brasileiro que são considerados, como mostrado na Figura 5.11.

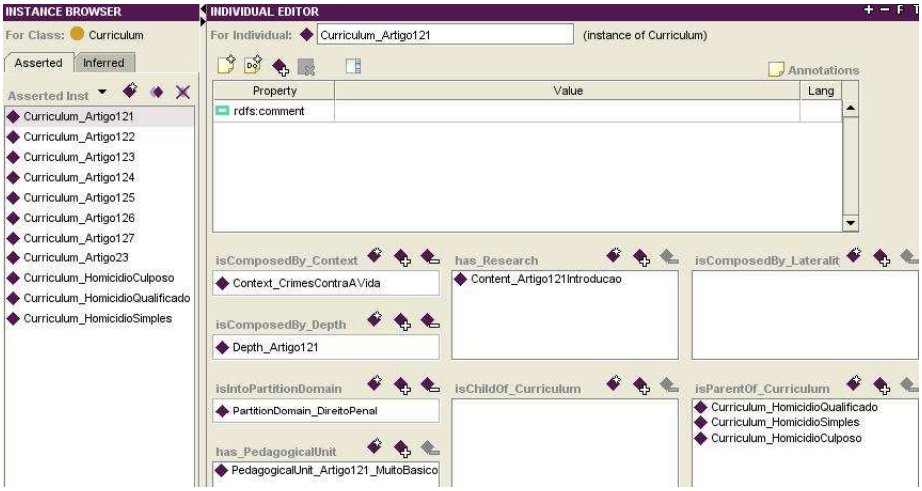

Figura 5.11: Conteúdo Curricular do Sistema *Themis*

Estratégias Pedagógicas e Plano Instrucional O plano instrucional possui forte dependência com as estratégias pedagógicas definidas, portanto, há a necessidade de se definir primeiramente a estratégia pedagógica que se pretende utilizar, e, logo após, definir o plano instrucional do sistema. A Figura 5.12 aborda a especificação das estratégias pedagógicas inerentes ao domínio do problema.

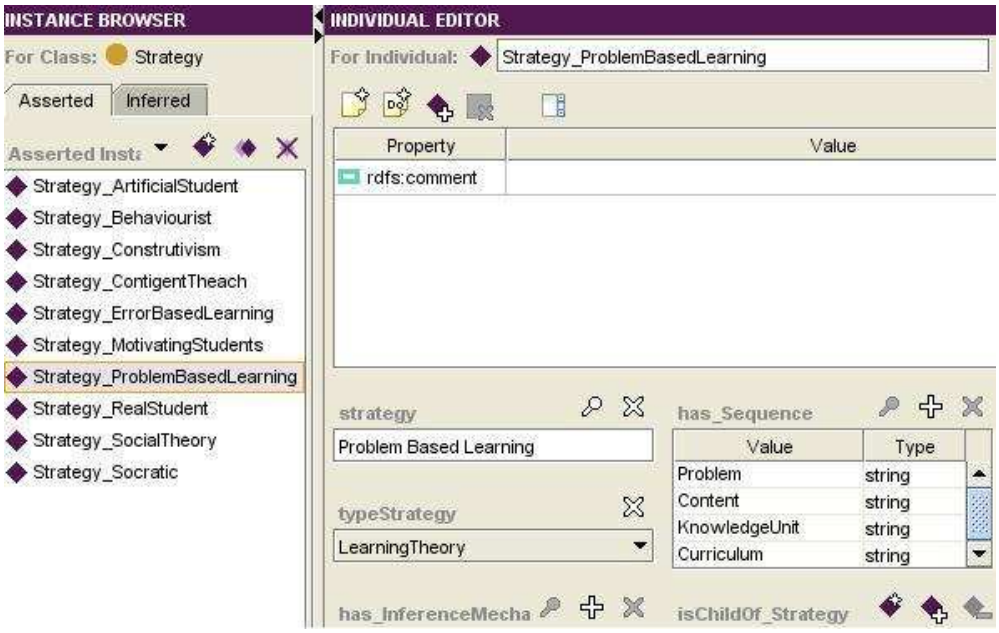

Figura 5.12: Estratégia Pedagógica do Sistema *Themis*

A especificação das estratégias pedagógicas com o plano instrucional se faz como mo-

strado na 5.13.

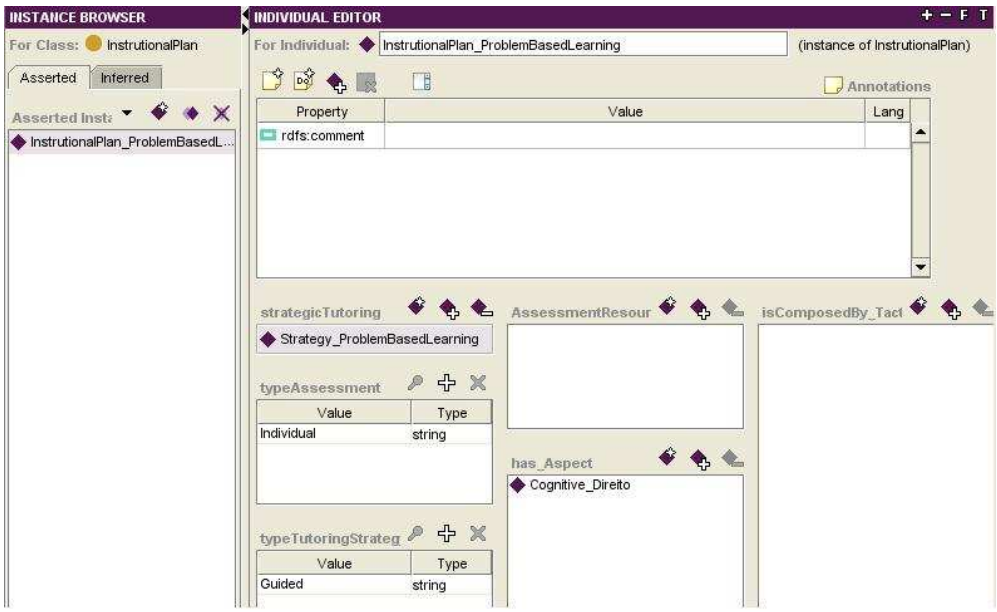

Figura 5.13: Plano Instrucional do Sistema *Themis*

Objetivos de Aprendizagem Os objetivos de aprendizagem referem-se aos currículos que o estudante deve aprender e em qual nível de profundidade, ou seja, se o estudante aprende o básico ou avançado. Para o domínio de direito, todos os currículos previamente definidos são estudados pelos estudantes. Além disto, exige-se um nível de conhecimento aprofundado. Para isso, a Figura 5.14 aborda a ontologia com os objetivos de aprendizagem.

Mecanismos de Inferência Os mecanismos de inferência de raciocínio baseado em casos e raciocínio baseado em regras já são providos na plataforma, não havendo a necessidade de implementação dos algoritmos. Porém, caso o engenheiro de software queira utilizar outro mecanismo terá que implementá-lo. A configuração dos mecanismos de inferência é descrita abaixo.

Preocupações do Engenheiro do Conhecimento O engenheiro do conhecimento deve configurar os mecanismos de inferência, os quais são mostrados a seguir.

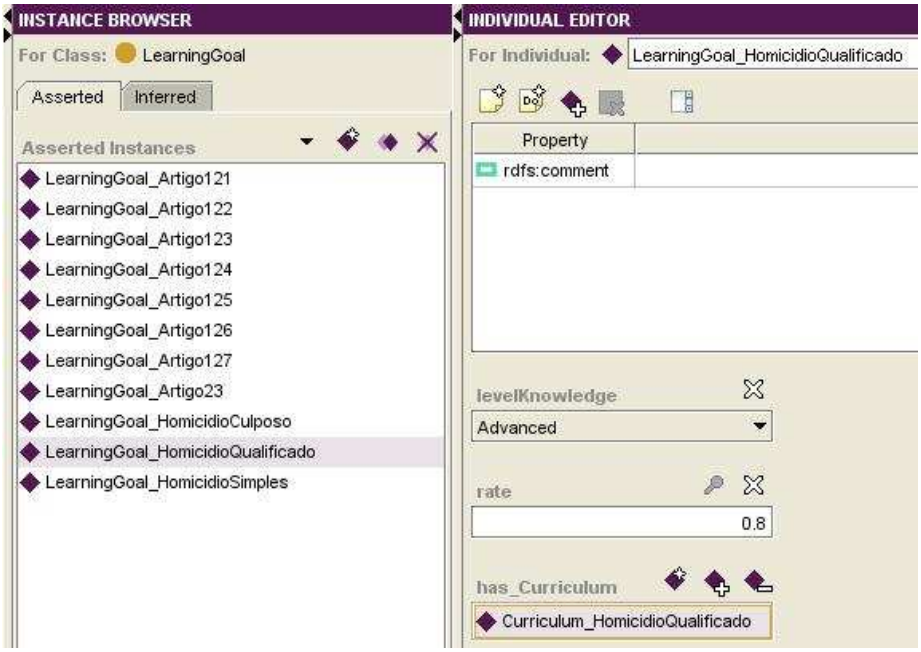

Figura 5.14: Objetivos de Aprendizagem do Estudante de Direito

Configuração do RBC Segue-se nesta seção um passo-a-passo sobre a utilização do componente RBC[6]. Os passos são:

- **Passo 1**: o usuário seleciona a base com os casos jurisprudênciais;
- **Passo 2**: define os atributos (Relação entre os personagens, Especificação do fato, etc), como foram especificados anteriormente.
- **Passo 3**: é definido o peso de cada atributo, tendo sido definido 1;
- **Passo 4**: as funções de similaridade local já foram previamente definidas;
- **Passo 5**: a recuperação ocorre de forma seqüencial;
- **Passo 6**: as fases de adaptação e avaliação da solução não foram tratadas;
- **Passo 7**: o limiar de retenção equivale a 0.8;

Configuração do RBR Os passos para configuração do componente RBR são:

[6]Os casos foram adquiridos de sites que possuem jurisprudências disponíveis para consultas.

- **Passo 1**: o usuário registra as variáveis e seus respectivos valores, referentes ao código penal;
- **Passo 2**: o usuário cria as regras específicas do domínio penal;
- **Passo 3**: o usuário escolhe o método de encadeamento para frente;
- **Passo 4**: são definidas as explanações relacionadas com as regras, baseadas na doutrina.

As regras do sistema levam em consideração os seguintes atributos:

- Personagens envolvidos;
- Idades dos personagens envolvidos: é um importante atributo quando algum dos envolvidos é menor de idade $(age <= 18)$;
- Deficiências dos personagens: especifica se algum personagem possui alguma deficiência que pode ser considerada, por exemplo, em caso de vítima não poder se defender;
- Condições dos personagens: define se algum personagem está, por exemplo, bêbado, drogado, dentre outras condições;
- Fato (tentativa foi bem sucedida): define se o crime foi concretizado;
- Arma usada: a arma é fundamental, pois pode caracterizar a gravidade do crime;
- Razão do Crime: especifica se o crime foi perpetrado por revanche, mandado, dentre outras.

Foram utilizadas 13 variáveis e 27 regras, onde algumas regras seguem abaixo.

1. Rule

   IF AccusedCondition = 'Puerperal' and Victim = 'son' and VictimAge <= 0 THEN Article = 123

2. Rule

   IF VictimAge <= 18 and CrimeReason in ['recompense', 'futile', 'nasty', 'betrayal', 'ambuscade'] THEN Article = 121 and Paragraph = 4

5.2 Estudo de Caso 2: Medicina

Este estudo de caso aborda a construção de um ambiente interativo de aprendizagem no domínio da Anatomia Óssea do Crânio. O sistema tem sido desenvolvido pelo departamento de ciências biomédicas e de tecnologia da Universidade Católica de Brasília (UCB, 2006)em parceria com o Instituto de Computação da Universidade Federal de Alagoas.

5.2.1 Análise de requisitos

O Sistema educacional em questão aborda, dentre outras funcionalidades, um sistema tutor inteligente aplicado ao domínio da medicina, provendo aos estudantes casos sobre anatomia.

A idéia central do ambiente é interagir com estudantes de medicina através do estudo da anatomia óssea do crânio. Dentre as funcionalidades presentes no ambiente, têm-se:

- O processo de aprendizagem dos estudantes se dá através do estudo de:
    1. conceitos anatômicos: teorias referentes ao que é ensinado em sala de aula;
    2. resolução de problemas: o estudante pode resolver problemas de anatomia bem como solicitar ao sistema a resolução;
    3. Modelo 3D: o sistema possui uma funcionalidade adicional referente a um modelo de visualização tridimensional.
- A interação é da forma estudante-professor-agente tutor;
- O estudante pode interagir com outros estudantes através das ferramentas de fórum, chat e comunidade virtual.

Nesta perspectiva, investiga-se (i) prover um ambiente de estruturas anatômicas através de visualização, navegação e interação em um modelo tridimensional e (ii) criar um sistema tutor para educação médica para ensino sobre anatomia humana (BITTENCOURT et al., 2006b).

Os componentes deste sistema são similares aos do sistema *Themis*, porém com uma diferença fundamental. Este sistema disponibiliza novas funcionalidades através de três ferramentas de colaboração (fórum, *chat* e comunidade virtual) e de um modelo de visualização 3D da estrutura óssea do crânio.

Com isso, o diagrama *Web* do sistema fica como o exibido na Figura 5.15

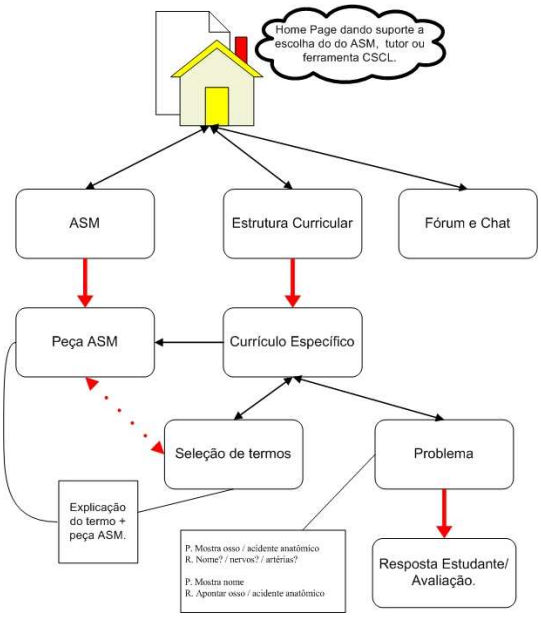

Figura 5.15: Diagrama Web do Sistema de Medicina

5.2.2 Construção do Sistema

Com o sistema especificado, é necessário agora definir quais são os papéis que irão interagir na construção do sistema. As preocupações são referentes a especificação do modelo e integração do modelo 3D ao sistema. Com isso, apenas o autor está inserido nas preocupação citadas.

- Preocupações do Autor:

    1. Conteúdo curricular;
    2. Estratégias Pedagógicas;
    3. Plano Instrucional;
    4. Objetivos de Aprendizagem;
    5. Modelo de visualização 3D;

Preocupações do Autor

Dentre as preocupações inerentes ao autor, destaca-se o modelo de visualização 3D. As outras especificações referem-se às mesmas definidas no Sistema *Themis*, mudando apenas o domínio.

Especificações via Ontologia A especificação via ontologia é bastante similar ao sistema *Themis*, mudando apenas o conteúdo curricular. A quantidade de conhecimento do domínio de anatomia óssea do crânio é enorme, onde houve um trabalho temporal considerável para a especificação do domínio e definição de indivíduos da ontologia. Parte do conteúdo curricular é mostrado na Figura 5.16

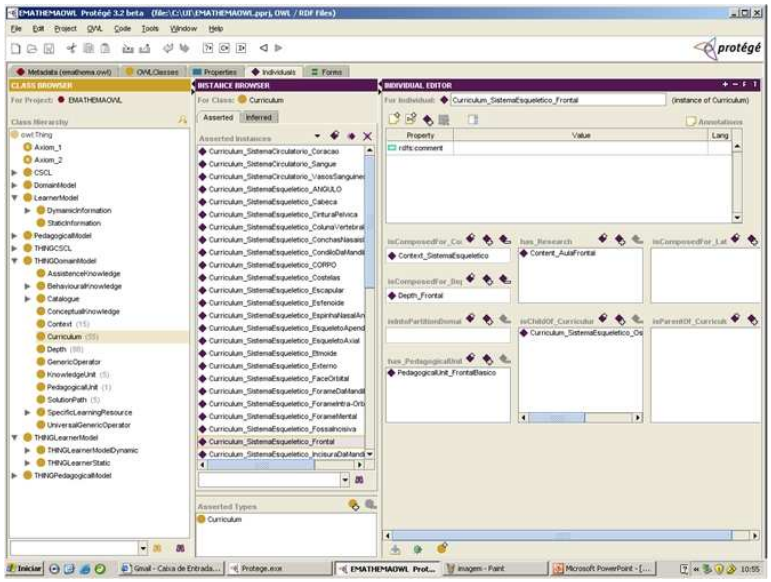

Figura 5.16: Conteúdo Curricular da Ontologia do Sistema de Medicina

Modelo de visualização 3D O modelo de visualização está sendo desenvolvido pela Universidade Católica de Brasília. Para construção do modelo, está sendo utilizado o Software Borland Visual C++. Tal ambiente foi utilizado pelo sistema através da construção de um arquivo de OCX que foi encapsulado na *Interface Web*. O modelo não possui nenhuma relação com o sistema educacional, ou seja, não requisita nenhum tipo de serviço aos agentes do arcabouço.

Abaixo seguem uma imagem do modelo 3D na web (ver Figura 5.17 sobre o modelo de visualização 3D.

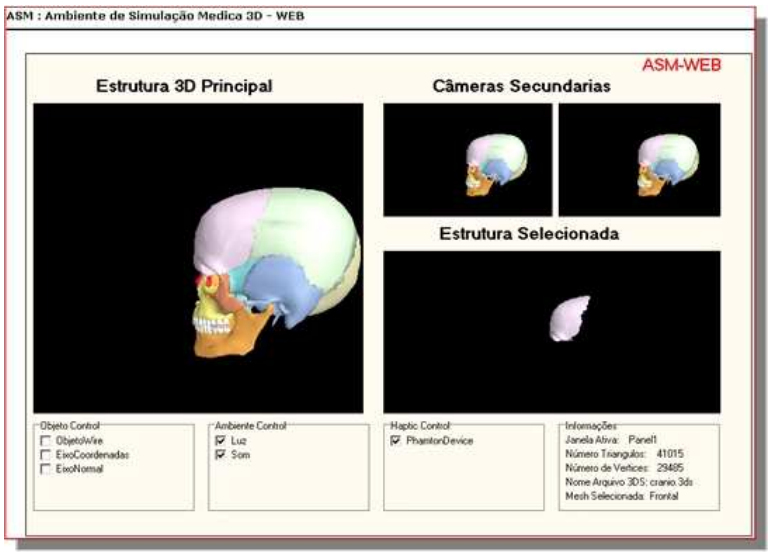

Figura 5.17: Modelo de Visualização 3D na Web

Para que o modelo 3D seja utilizado, precisa-se definir um recurso de aprendizagem diferenciado, pois tal estrutura não esta prevista na plataforma, havendo a necessidade de especificar tal funcionalidade na ontologia, como mostrado na Figura 5.18.

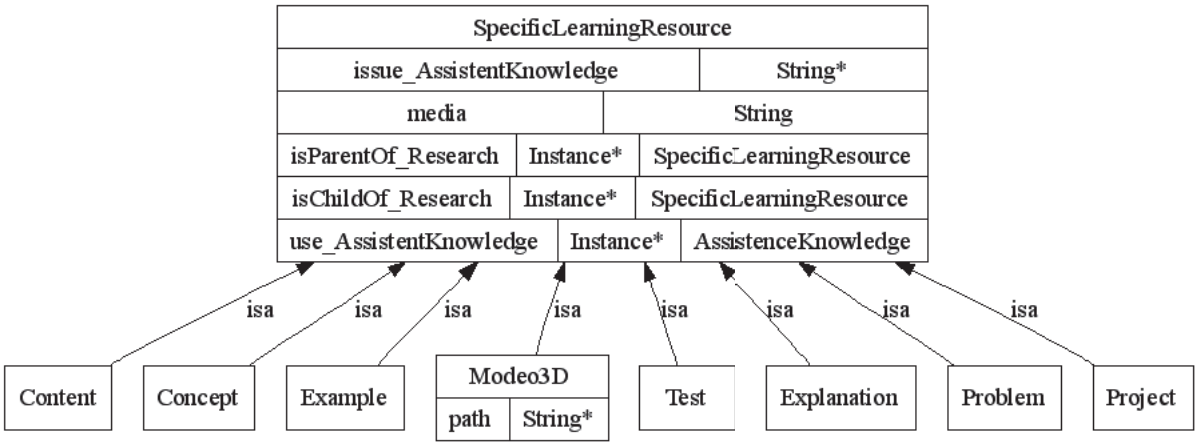

Figura 5.18: Modelo 3D especificado na Ontologia

Especificação do Problema Os problemas neste sistema equivalem a questões de múltipla escolha, sendo mais simples de se implementar. A plataforma já disponibiliza a avaliação dos problemas de múltiplas escolhas, relacionar colunas, preencher lacunas e subjetivos. Ou seja, o autor se preocupa apenas na especificação dos problemas na ontologia, como mostrado na Figura 5.19.

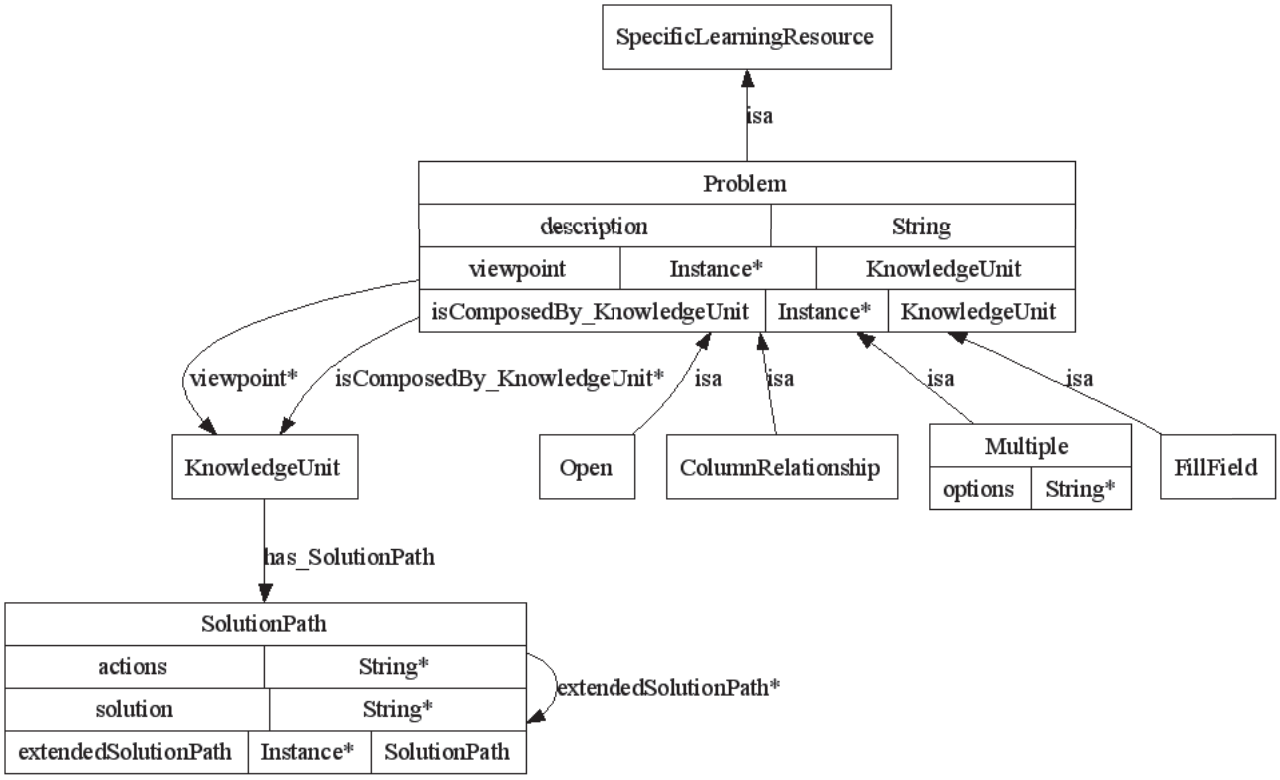

Figura 5.19: Problemas especificados na Ontologia

Os estudos de casos apresentados neste capítulo objetivaram abordar funcionalidades inerentes a ambientes educacionais, através da utilização do módulo de framework e autoria.

Capítulo 6

Conclusões e Perspectivas Futuras

6.1 Conclusões

Este trabalho objetivou abordar os aspectos de engenharia de software e inteligência artificial para os ambientes educacionais, mais especificamente focalizando tais aspectos sobre os ambientes interativos de aprendizagem. Para isto, uma investigação sobre diversos tipos de ambientes educacionais foi feita. Tal investigação buscou identificar quais características são comuns e relevantes para a educação mediada por computador.

Após identificação das características, o modelo MATHEMA revisto e ampliado, juntamente com a integração de outros modelos, foram utilizados como alicerces conceituais do trabalho.

Com isso, espera-se ter contribuído no que diz respeito aos usuários autores, através da noção de autoria e quanto aos usuários desenvolvedores através de um *framework*. Enfim, o presente trabalho passou por dois tipos de avaliações. Um deles foi a aprovação da comunidade científica a alguns de seus resultados que foram aceitos em conferências nacionais e internacionais (BITTENCOURT; COSTA, 2005; BITTENCOURT; BEZERRA; COSTA, 2006; SIBALDO; LOREIRO; COSTA, 2006; BITTENCOURT et al., 2006; BITTENCOURT; COSTA, 2006; BITTENCOURT et al., 2006b; BITTENCOURT; COSTA, 2006; BITTENCOURT et al., 2006a, 2006b, 2006d), além de um periódico (BITTENCOURT et al., 2006c). A outra avaliação foi na etapa de experimentação, onde se pode confirmar a viabilidade da plataforma.

6.2 Perspectivas Futuras

Sem dúvidas, ainda há muito a evoluir e aprimorar no trabalho aqui proposto.

Uma das perspectivas futuras do trabalhar é aprimorar o ambiente, dando suporte a características da Web Semântica, dotando o sistema de maior eficiência, maior capacidade de adaptação ao estudante, semântica estruturada de conteúdo e identificação de características e necessidades dos estudantes através de agentes inteligentes (STOJANOVIC; STAAB; STUDER, 2006).

A seguir são levantadas algumas das perspectivas futuras referente aos agentes e ferramentas, e, conseqüentemente, limitações da plataforma.

6.2.1 Agente Mediador

Como trabalhos futuros do agente mediador, têm-se:

- Metodologia: é utilizada pelos autor/engenheiro de *software* para construir o sistema educacional. Entende-se por metodologia, dois vieses diferenciados, sendo eles: processos de Engenharia de Software e metodologias educacionais. Processo de Engenharia de Software é voltado para equipes de desenvolvimento, objetivando a implementação de novas funcionalidades. Como exemplo pode ser definida uma metodologia RUP (KRUCHTEN, 2003) ou XP (BECK, 2001). Metodologias educacionais são voltadas para o autor que define qual a seqüência de construção do ambiente educacional e do conhecimento;

- Tipo de Ambiente Educacional: pode ser pré-definida a construção de diversos tipos de ambientes educacionais, como Sistemas Tutores Inteligentes, Sistemas Colaborativos, entre outros. A escolha do tipo de ambiente tem influência direta na metodologia educacional escolhida.

Com isso, o Agente Mediador irá facilitar a utilização da abordagem de autoria, pois a partir do momento que diversos recursos são disponibilizados para o autor, a complexidade e usabilidade tornam-se inversamente proporcionais, ou seja, a complexidade aumenta e a capacidade de gerenciamento dos recursos diminui. Com isso, o agente mediador tem total

controle de todas as atividades que devem ser executadas pelos agentes e a quantos passos de distância o autor está da finalização da configuração do ambiente educacional.

6.2.2 Agentes Tutores Autônomos

Nos Agentes Tutores Autônomos, pretendem-se três melhorias:

1. Construir um módulo de configuração de distribuição dos agentes tutores autônomos;
2. Integração automática com outras bases de conhecimento para facilitar o reúso de conhecimentos através do mapeamento do conteúdo;
3. Abordar a necessidade de melhoramento dos aspectos motivacionais e afetivos.

6.2.3 Agente de Suporte

São três as perspectivas futuras dos agentes de suporte:

1. Melhoramento dos mecanismos de inferência de casos e regras;
2. Desenvolvimento de novos mecanismos de aprendizagem. Os próximos mecanismos que se pretende desenvolver são algoritmos de busca, aprendizagem por reforço e redes bayesianas;
3. Interfaces adaptativas para facilitar o reajuste das telas da plataforma. Isto é necessário, por exemplo, quando o autor decide integrar diversos mecanismos de inferência;

Outras técnicas podem ser adicionadas na plataforma através da configuração da ontologia e adição do agente específico da técnica.

6.2.4 Ferramentas Interativas

Pretende-se desenvolver novas ferramentas interativas, buscando maximizar o aprendizado do aluno.

Bibliografia

ACQUAVIVA, M. C. *Dicionário Enciclopédico do Direito*. São Paulo: Brasiliense, 1988.

AGUIAR, A. *A minimalist approach to framework documentation*. Tese (tese) — Faculdade de Engenharia da Universidade do Porto, 2003.

AIRES. *Laboratory for the Advanced Researches in Intelligent Educational Systems*. 1992. Disponível em: http://www.cs.usask.ca/research/research_groups/aries/. Acesso: 23 Ago. 2006.

ALEVEN, V. Using background knowledge in case-based legal reasoning: a computational model and an intelligent learning environment. *Artificial Intelligence*, Elsevier Science Publishers Ltd., Essex, UK, Vol. 150, n. 1-2, p. 183–237, 2003. ISSN 0004-3702.

ALEVEN, V. et al. The cognitive tutor authoring tools (ctat): Preliminary evaluation of efficiency gains. In: *Intelligent Tutoring Systems*. [S.l.: s.n.], 2006. p. 61–70.

ALEVEN, V. et al. Rapid authoring of intelligent tutors for real-world and experimental use. In: *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies (ICALT 2006)*. [S.l.: s.n.], 2006.

ALEXANDER, B. *Web 2.0: A New Wave Of Innovation for Teaching and Learning?* [S.l.], 2006.

ALHIR, S. S. *Learning UML*. [S.l.]: O'Reilly, 2003.

ALMEIDA, H. O. de; COSTA, E. de B.; PERKUSICH, A. Desenvolvimento de software para sistemas multiagentes. In: COMPUTAçãO, A. do XXV Congresso da Sociedade Brasileira de (Ed.). *XVIII Concurso de Teses e Dissertações (CTD/SBC'05), São Leopoldo, RS*. [S.l.: s.n.], 2005. v. 25, p. 80–85.

AROYO, L. et al. Ease: Evolutional authoring support environment. In: *Intelligent Tutoring Systems*. [S.l.: s.n.], 2004. p. 140–149.

AROYO, L.; MIZOGUCHI, R.; TZOLOV, C. Ontoaims: Ontological approach to courseware authoring. In: *Proceedings of the International Conference on Computers in Education (ICCE2003),Wanchai, HongKong*. [S.l.: s.n.], 2003. p. 1011–1014.

AZEVEDO, H. J. S. de. *Contribution à la modélisation des connaisssances à l'aide systèmes multi-agents.* Tese (Doutorado) — University of Technology of Compiègne, France, 1997.

AZEVEDO, H. J. S. de; SCALABRIN, E. E. Designing distributed learning environment with intelligent software agents. In: ____. [S.l.]: Idea Group, 2005. cap. A Human Collaborative Online Learning Enrironment Using Intelligent Agents, p. 1–32.

BARROS, B.; VERDEJO, M. F.; MIZOGUCHI, R. A platform for collaboration analysis in cscl. an ontological approach. In: *Artificial Intelligence in Education AIED, San Antonio, USA*. [S.l.]: IOS Press, 2001. p. 530–532.

BARROS, B. et al. Applications of a collaborative learning ontology. In: . [S.l.: s.n.], 2002. p. 301–310.

BARRY, D. K. *Web Services and Service-Oriented Architectures*. 1. ed. [S.l.]: Brochura, 2003. (ISBN 1558609067).

BEAL, C. R.; LEE, H. Creating a pedagogical model that uses student self reports of motivation and mood to adapt its instruction. In: *Workshop on motivation and affect in educational on motivation and affect in educational software, 2005, Amsterdam, Netherland*. [S.l.: s.n.], 2005. p. 18–22.

BECK, K. *Programação eXtrema (XP) eXplicada*. [S.l.]: Bookman, 2001.

BITTENCOURT, I.; BEZERRA, C.; COSTA, E. Ferramenta baseada na web semântica para geração de avaliações pedagógicas. In: *Simpósio Brasileiro de Sistemas Multimídia e Web, WebMedia, Natal*. [S.l.: s.n.], 2006.

BITTENCOURT, I. et al. Ontologia para construção de ambientes interativos de aprendizagem. In: *XVII Simpósio Brasileiro de Informática na Educação, SBIE, Brasília*. [S.l.: s.n.], 2006. p. 559–568.

BITTENCOURT, I.; COSTA, E. Construção de ambientes interativos de aprendizagem usando agentes inteligentes. In: *XVII Simpósio Brasileiro de Informática na Educação, SBIE, Brasília*. [S.l.: s.n.], 2006.

BITTENCOURT, I. et al. Um arcabouço para construção de sistemas baseados em agentes inteligentes. In: *XX Semana Paraense de Informática, SEPAI, Belém*. [S.l.: s.n.], 2006.

BITTENCOURT, I. et al. Um sistema de autoria para construção de ambientes interativos de aprendizagem baseado em agentes. In: *XVII Simpósio Brasileiro de Informática na Educação, SBIE, Brasília*. [S.l.: s.n.], 2006. p. 487–496.

BITTENCOURT, I. I.; COSTA, E. An agent based hybrid intelligent tutoring system for legal domain. In: FORA, U. F. de Juiz de (Ed.). *Proceedings do XVI Simpósio Brasileiro de Informática na Educação, Juiz de Fora, Brazil*. [S.l.: s.n.], 2005. v. 1, p. 180–189.

BITTENCOURT, I. I.; COSTA, E. The design and development of a tutoring system for legal domain. In: *The Fifth IASTED International Conference on Web-Based Education - WBE, Puerto Vallarta, México*. [S.l.: s.n.], 2006.

BITTENCOURT, I. I. et al. Sistemas de informação e apoio a decisão baseado em conhecimento. In: *Escola Regional de Informática de Minas Gerais*. [S.l.: s.n.], 2006.

BITTENCOURT, I. I. et al. Um sistema tutor inteligente multiagentes em anatomia Óssea do crânio. In: *Publicação aceita no Congresso Brasileiro de Infromática Biomédica*. [S.l.: s.n.], 2006.

BITTENCOURT, I. I. et al. An agent-based intelligent tutoring system: A case study in legal domain, accepted for publication. *Journal of International Transaction System Science And Applications*, 2006.

BITTENCOURT, I. I. et al. Combining ai techniques into a legal agent-based intelligent tutoring system. In: *Eighteenth International Conference on Software Engineering and Knowledge Engineering - SEKE, 2006, San Francisco*. [S.l.: s.n.], 2006. v. 18, p. 35–40.

BLOEDORN, E.; MANI, I.; MACMILLAN, T. R. Machine learning of user profiles: Representational issues. In: *AAAI/IAAI, Vol. 1*. [s.n.], 1996. p. 433–438. Disponível em: <citeseer.ist.psu.edu/bloedorn96machine.html>.

BOULAY, B. du; LUCKIN, R. Modelling human teaching tactics and strategies for tutoring systems. *International Journal of Artificial Intelligence in Education*, v. 12, p. 235–256, 2001.

BRAUBACH, L.; POKAHR, A.; WALCZAK, A. *Jadex*. 2002. Disponível em: http://vsis-www.informatik.uni-hamburg.de/projects/jadex/. Acesso em: 03 Ago. 2006.

BRITTAIN, J.; DARWIN, I. F. *Tomcat: The Definitive Guide*. 1. ed. [S.l.]: O'Reilly, 2003.

CENTINIA, F. et al. Statutor: Too intelligent by half? In: HAGE, J. et al. (Ed.). *Legal knowledge based systems, JURIX'95, Telecommunication and AI & Law, Koninklijke Vermande, Lelystad, Netherlands*. [S.l.: s.n.], 1995. p. 121–132.

CHEN, W.; MIZOGUCHI, R. Leaner model ontology and leaner model agent. *Cognitive Support for Learning - Imagining the Unknown*, p. 189–200, 2004.

CHEN, W.; WASSON, B. Designing distributed learning environment with intelligent software agents. In: ____. [S.l.]: Idea Group, 2005. cap. Intelligent Agents Supporting Distributed Collaborative Learning, p. 33–68.

CHEPEGIN, V. *UserModelling*. Http://smi-protege.stanford.edu:8080/ Knowledge-Zone/OntologyMetadata?ontologyid=22.

CLASSROOM, O. the. *Outside the Classroom*. 2006.

CMU. *Carnegie Mellon University*. 2006. Disponível em: http://www.cmu.edu/. Acesso: 23 Ago. 2006.

COSTA, E.; PERKUSICH, A.; FERNEDA, E. From a tridimensional view of domain knowledge to multi-agents tutoring systems. In: . [S.l.: s.n.], 1998. p. 61–72.

COSTA, E. B. *Um Modelo de Ambiente Interativo de Aprendizagem Baseado numa Arquitetura Multi-Agentes*. Tese (Tese de Doutorado) — Universidade Federal da Paraíba, Campina Grande, 1997.

COTTER, S.; TALIGENT, I. *Introduction to Taligent technology*. Addison-Wesley Publishing Company, 1995. Disponível em: <http://www.wildcrest.com/Potel/Portfolio/InsideTaligentTechnology/WW6.htm>.

CTAT. *Cognitive Tutor Authoring Tools*. 2003. Disponível em: http://ctat.pact.cs.cmu.edu. Acesso em: 23 Ago. 2006.

DACONTA, M. C.; OBRST, L. J.; SMITH, K. T. *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*. [S.l.]: Wiley Publishing, Inc.,, 2003.

DEERWESTER, S. et al. Indexing by latent semantic analysis. *Journal of the Society for Information Science*, v. 41, p. 391–407, 1990.

DILLENBOURG, P.; SELF, J. A framework for learner modelling. *Interactive Learning Environments*, v. 2, p. 111–137, 1992.

DOCTA-NSS. *Design and use of Collaborative Telelearning Artefacts*. 2006. Disponível em: http://www.intermedia.uib.no/docta/doctanss.html. Acesso: 31 Ago. 2006.

DOLONEN W. CHEN, A. M. J. Integrating software agents with fle3. In: *Proceedings of the International Conference on Computer Support for Collaborative Learning. Kluwer.* [S.l.: s.n.], 2003.

EINDHOVEN. *Eindhoven University of Technology*. Disponível em: http://www.tue.nl/. Acesso: 23 Ago. 2006.

FAYAD, M. E.; SCHMIDT, D. C.; JOHNSON, R. E. *Building application frameworks: object-oriented foundations of framework design*. New York, NY, USA: John Wiley & Sons, Inc., 1999. ISBN 0-471-24875-4.

FERGUSON, K. et al. Improving intelligent tutoring systems: Using expectation maximization to learn student skill levels. In: *Intelligent Tutoring Systems*. [S.l.: s.n.], 2006. p. 453–462.

FIPA. *Foundation for Intelligent Physical Agents*. 2006. Http://www.fipa.org.

GATECH. *Georgia Institute of Technology,*. 2006. Disponível em: http://www.gatech.edu/. Acesso: 24 Ago. 2006.

GEROSA, M. A. et al. Uma arquitetura para o desenvolvimento de ferramentas colaborativas para o ambiente de aprendizagem aulanet. In: *Anais do XV Simpósio Brasileiro de Informática na Educação - SBIE 2004*. [S.l.: s.n.], 2004.

GOODMAN, B. et al. A framework for asynchronous collaborative learning and problem solving. In: *In the Proceedings of the 10th International Conference on Artificial Intelligence in Education, AIED*. [S.l.: s.n.], 2001.

GREER, J. et al. Lessons learned in deploying a multi-agent learning support system: The i-help experience. In: AMSTERDAM, I. P. (Ed.). *Proceedings of AI in Education AIED'2001, San Antonio*. [S.l.: s.n.], 2001. p. 410–421.

GRUBER, T. R. A translation approach to portable ontology specifications'. *Knowledge Acquisition*, v. 5, n. 2, p. 199–220, 1993.

HARRER, A. et al. Collaboration and cognitive tutoring: Integration, empirical results, and future directions. In: *Proceedings of the 12th International Conference on Artificial Intelligence in Education, AIED*. [S.l.]: Amsterdam: IOS Press, 2005.

HARTMANN, C. et al. Linguagem e ferramenta de autoria para promover o desenvolvimento de perícias em xadrez. In: *Anais do XVI Simpósio Brasileiro de Informática na Educação, Juiz de Fora*. [S.l.: s.n.], 2005. v. 1, p. 656–665.

HORRIDGE, M. et al. *A Practical Guide to Build OWL Ontologies Using the Protégé-OWL Plugin and CO-ODE Tolls*. 1. ed. [S.l.], August 2004.

JENA. *Jena - A Semantic Web Framework for Java*. Disponível em: http://jena.sourceforge.net. Acesso em: 27 Set. 2006.

JESS. *JESS the Rule Engine for the Java Platform*. 2003. Disponível em: http://herzberg.ca.sandia.gov/jess/. Acesso: 23 Ago. 2006.

JESUS, D. de. *Código Penal*. 17. ed. [S.l.: s.n.], 2005. (850205225X).

JOHNSON, R. E. Documenting frameworks using patterns. In: *OOPSLA 92: conference proceedings on Object-oriented programming systems, languages, and applications*. New York, NY, USA: ACM Press, 1992. p. 63–76. ISBN 0-201-53372-3.

JOHNSON, R. E. Components, frameworks, patterns. In: *ACM SIGSOFT Symposium on Software Reusability*. [s.n.], 1997. p. 10–17. Disponível em: <citeseer.ist.psu.edu/johnson97components.html>.

JOHNSON, R. E.; FOOTE, B. Designing reusable classes. *Journal of Object-Oriented Programming*, v. 1, n. 2, p. 22–35, 1988. Disponível em: <ftp://st.cs.uiuc.edu/pub/papers/frameworks/designing-reusable-classes.ps ftp://p300.cpl.uiuc.edu/pub/foote/DRC.ps>.

JOHNSON, W. L. et al. Modeling motivational and social aspects of tutorial dialog. In: *Proceedings of the ITS'04 Workshop on Modelling Human Teaching Tactics and Strategies, Maceió, 2004*. [S.l.: s.n.], 2004. p. 35–44.

JR., A. P. L. *Uma Arquitetura Baseada em Agentes para um Sistema Tutor Inteligente em Cardiologia*. Dissertação (Mestrado) — Universidade Católica de Brasília (UCB), Junho 2005.

KRALINGEN, R. van. A conceptual frame-based ontology for the law. In: *Proceedings of the First International Workshop on Legal Ontologies, Melbourne, Australia*. [S.l.: s.n.], 1997.

KRUCHTEN, P. *The Rational Unifed Process: An Introduction*. [S.l.]: Addison Wesley, 2003.

KUMAR, V. et al. Toward an ontology of teaching strategies. In: *Proceedings of the ITS'04 Workshop on Modelling Human Teaching Tactics and Strategies, Maceió, 2004*. [S.l.: s.n.], 2004. p. 71–80.

LESSER, V. R. Cooperative multiagent systems: A personal view of the state of the art. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, v. 11, n. 1, p. 133–142, JANUARY/FEBRUARY 1999.

MAJOR, N.; AINSWORTH, S.; WOOD, D. Redeem: Exploiting symbiosis between psychology and authoring environments. *International Journal of Artificial Intelligence in Education*, v. 8, p. 317–340, 1997.

MATSUDA, N.; VANLEHN, K. Decision theoretic instructional planner for intelligent tutoring systemsmodeling motivational and social aspects of tutorial dialog. In: *Proceedings of the ITS'00 Workshop on Modelling Human Teaching Tactics and Strategies, Montreal, 2000*. [S.l.: s.n.], 2000. p. 72–83.

MATTSSON, S. E. On object-oriented modeling of relays and sliding mode behaviour. In: *Proceedings of the 1996 Triennal IFAC World Congress, IFAC'96*. "San Francisco, California, USA: Elsevier Science, 1996. F, p. 259–264. Disponível em: <citeseer.ist.psu.edu/article/mattsson96objectoriented.html>.

MCLAREN, B. M.; KOEDINGER, K. R.; SHNEIDER, M. Cognitive tutoring in a collaborative, web-based environment. In: *Engineering Advanced Web Applications: Proceedings of Workshops in Connection with the 4th International Conference on Web Engineering*. [S.l.]: Princeton: Rinton Press, 2005. p. 167–179.

MENEZES, C. S. de; CURY, D.; BERNARDINO, G. H. C. Amcora: um ambiente cooperativo para a aprendizagem construtivista utilizando a internet. In: *X Simpósio Brasileiro de Informática na Educação, Curitiva*. [S.l.: s.n.], 1999. p. 333–340.

MITRE. *Application Systems Engineering and Advanced Technology to Critical National Problems*. 1997. Disponível em: http://www.mitre.org/. Acesso: 24 Ago. 2006.

MIZOGUCHI, R.; BOURDEAU, J. Using ontological engineering to overcome common ai-ed problems. *International Journal of Artificial Intelligence in Education*, v. 11, n. 2, p. 107–121, 2000.

MOLAR, A. S. Computers in education: A brief history. *T.H.E. Journal, LLC*, v. 24, p. 63–69, June 1997.

MORABITO, M. G. *Online Distance Education*. [S.l.]: Dissertation.com, 1997.

MURRAY, T. Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence and Education, IJAIED*, v. 10, p. 98–129, 1999.

MURRAY, T. Authoring tools for advanced technologies learning environments: Toward cost-effective adaptive, interactive and intelligent educational software. In: ____. [S.l.]: Kluwer Academic Publishers, Netherlands, 2003. cap. An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art, p. 493–546.

MUUKKONEN, H.; HAKKARAINEN, K.; LAKKALA, M. Collaborative technology for facilitating progressive inquiry: future learning environment tools. In: *CSCL '99: Proceedings of the 1999 conference on Computer support for collaborative learning*. [S.l.]: International Society of the Learning Sciences, 1999. p. 51.

NASSEH, B. A briefy history of distance education. *Adult Education in the News*, 1997.

NETTO, J. F. de M.; MENEZES, C. S. de; JúNIOR, A. N. de C. Amcora : Uma arquitetura multiagente baseada em fipa. In: *XV Simpósio Brasileiro de Informática na Educação, Manaus*. [S.l.: s.n.], 2004.

NEWCOMER, E. *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. 1. ed. [S.l.]: Addison-Wesley Professional, 2002.

OKBC. *Open Knowledge Base Connectivity*. 1995. Disponível em: http://www.ai.sri.com/ okbc. Acesso em: 27 Set. 2006.

OSAKA. *Osaka University*. 2005. Disponível em: http://www.osaka-u.ac.jp/eng/. Acesso em: 24 Ago. 2006.

OTSUKA, J. L.; ROCHA, H. V. da. Um modelo de suporte À avaliaÇÃo formativa para ambientes de educaÇÃo a distÂncia: Dos conceitos À soluÇÃo tecnolÓgica. *Revista Novas Tecnologias na Educação (Renote)*, v. 3, n. 2, Novembro 2005.

OWL. *OWL Web Ontology Language Overview*. 2006. Disponível em: http://www.w3.org/TR/owl-features/. Acesso em: 22 de Setembro de 2006.

PAPERT, S. *LOGO: Computadores e Educação*. [S.l.]: Brasiliense, São Paulo (SP), 1985.

PAPERT, S. Microworlds: Transforming education. *Artificial Intelligence and Education - Learning Environments and Tutoring Systems*, v. 1, 1987.

PESSOA, J. M.; NETTO, H. V.; MENEZES, C. S. de. Famcora: um framework para a construção de ambientes cooperativos inteligentes de apoio a aprendizagem na internet baseado em web services e agentes. In: PINTO, S. C. C. S. (Ed.). *XIII Simpósio Brasileiro de Informática na Educação - SBIE 2002: Metodologias, tecnologias e aprendizagem dentro do cenário de informática na educação*. [S.l.]: UNISINOS, 2002.

PLEKHANOVA, V. *Intelligent Agent Software Engineering*. Hershey, PA, USA: Idea Group Publishing, 2002. ISBN 1591400465.

PREDICTIVENETWORKS. *Predictive Networks*. 2006. Disponível em:. Acesso: 24 Ago. 2006.

PROTéGé. *The Protégé Ontology Editor and Knowledge Acquisition System*. 2006. Disponível em: http://protege.stanford.edu/. Acesso em: 22 Set. de 2006.

PUC-PR. *Pontifícia Universidade Católica do Paraná*. Disponível em: http://www.pucpr.br/index_especializacao.php. Acesso: 30 Ago. 2006.

RODRIGUES, L. M. L.; CARVALHO, M. Its-i intelligent tutoring system with integrated cognition, emotion and motivation. In: *XV Simpósio Brasileiro de Informática na Educação, Manaus, Brazil*. [S.l.: s.n.], 2004.

RODRIGUES, P. N. M.; SANTOS, M. F. Future challenger in intelligent tutoring systems - a framework. In: *m-ICTE2005 3rd International Conference on Multimedia and Information and Communication Technologies in Education*. [S.l.]: FORMATEX, 2005.

RUSSEL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. [S.l.]: Elsevier, 2004.

SASK. *University of Saskatchewan*. 1994. Disponível em: http://www.usask.ca/. Acesso: 23 Ago. 2006.

SAVERY, J. R.; DUFFY, T. M. Problem based learning: An instructional model and its constructivist framework. *Educational Technology*, p. 31–38, 1995.

SCALABRIN, E. E. et al. A generic model of cognitive agent to develop open systems. In: *SBIA '96: Proceedings of the 13th Brazilian Symposium on Artificial Intelligence*. London, UK: Springer-Verlag, 1996. p. 61–70. ISBN 3-540-61859-7.

SCHANK, R. C. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. New York, EUA: Cambridge University Press, 1982.

SIBALDO, M.; LOREIRO, I. B. E.; COSTA, E. Infra-estrutura para acesso a comunidades virtuais na web através de dispositivos móveis. In: *XVII Simpósio Brasileiro de Informática na Educação, SBIE, Brasília*. [S.l.: s.n.], 2006.

SILVA, R. P. *Modelo de Apoio ao Diagnóstico no Domínio Médico aplicando Raciocínio Baseado em Casos*. Dissertação (Mestrado) — Universidade Católica de Brasília, Brasília (DF), 2005.

SIVA, R. pereira e; PRICE, R. T. Uso de técnicas de modelagem no projeto de frameworks orientado a objetos. p. 15, 1997. Acessado em dezembro de 2005.

SMI. *Stanford Medical Informatics*. 2006. Disponível em: http://www.smi.stanford.edu. Acesso em: 27 Set. 2006.

SPARQL. *SPARQL Query Language fir RDF*. October 2006. Disponível em: http://www. Acesso em: 12 Out. 2006.

STANFILL, C.; WALTZ, D. Toward memory-based reasoning. *Communications of the ACM*, v. 29, December 1986.

STOJANOVIC, L.; STAAB, S.; STUDER, R. elearning based on the semantic web. In: *DRTC Conference on ICT for facilitating Digital Learning Environment*. Bangalore, India: Documentation Research and Training Centre, 2006.

TALIGENT. *Building Object-Oriented Frameworks*. [S.l.], 1994. Disponível em: <http://www.ibm.com/java/education/oobuilding/index.html>.

THOMPSON, C. A.; GÖKER, M. H.; LANGLEY, P. A personalized system for conversational recommendation. *Journal of Artificial Intelligence Research*, v. 21, p. 393–428, 2004.

TILAB. *Java Agent Development Framework*. 2005. Disponível em: http://jade.tilab.com/. Acesso em: 03 Out. 2006.

TU/E. *Department of Mathematics and Computer Science of Eindhoven University of Technology*. Disponível em: http://w3.win.tue.nl/en/. Acesso em: 24 Ago. 2006.

TURANI, A.; CALVO, R. A.; GOODYEAR, P. An application framework for collaborative learning. In: *ICWE*. [S.l.: s.n.], 2005. p. 243–251.

UCB. *Universidade Católica de Brasília*. 2006. Disponível em: http://www.ucb.br. Acesso em: 13 Out. 2006.

UFES. *Universidade Federal do Espírito Santo*. Agosto 2006. Disponível em: http://www2.inf.ufes.br/. Acesso em: 23 Ago. 2006.

UIB. *University of Bergen*. 1999. Disponível em: http://www.uib.no/info/english/. Acesso: 31 Ago. 2006.

UTTPR. *Universidade Tecnológica Federal do Paraná*. 2005. Dicponível em: http://www.utfpr.edu.br/. Acesso: 30 Ago. 2006.

VASSILEVA, J.; DETERS, R. Lessons from deploying i-help. In: *of the Workshop on Agents and Internet Learning, AIL'2001 at the Autonomous Agents'2001 Conference, Montreal,*. [S.l.: s.n.], 2001.

VIRVOU, M.; MOUNDRIDOU, M. A web-based authoring tool for intelligent tutoring systems. *Educational Technology & Society*, v. 3, n. ISSN 1436-4522, 2000.

W3C. *World Wide Web Consortium*. 1994. Disponível em: http://www.w3c.com. Acesso em: 27 Set. 2006.

WEBBER, C.; PESTY, S. The baghera multiagent learning environment: An educational community of artificial and human agents. *Upgrade the European Journal for The Informatics Professional, Europe*, v. 4, p. 40–44, 2004.

WU, A. K.-W. Paradigms of its (intelligent tutoring system). In: *TENCON 93'. Proceendings. Computer, Communicatio, Control and Power Engineering. Beijing, China.* [S.l.: s.n.], 1993. p. 96–99.

ZUKOWSKI, J. *Java AWT Reference*. 1. ed. [S.l.]: O'Reilly, 1997. (Java Series).

ZUKOWSKI, J. *The Definitive Guide to Java Swing*. 3. ed. [S.l.]: Apress, 2005.

Apêndice A

Tecnologias Utilizadas

A.1 Jade

O JADE (*Java Agent Development Framework*) é uma plataforma *open source* para desenvolvimento de sistemas multiagentes distribuídos, possui licença LGPL (*Library General Public License*), é implementado totalmente em Java e segue as especificações propostas pela FIPA(FIPA, 2006)(*Foundation for Intelligent Physical Agents*)[1]. Por essas características, o JADE foi escolhido para o projeto, além de ser uma plataforma de fácil acesso e largamente utilizada no meio acadêmico.

A.1.1 Arquitetura

Um modelo abstrato para uma plataforma de agentes pode ser visto na Figura A.1, onde alguns ítens importantes, definidos pela FIPA, estão identificados:

- *Agent Management System (AMS)*: responsável por controlar todo o acesso e uso de uma instância da plataforma. Conhecido como "serviço de páginas brancas", existe apenas um AMS para cada instância da plataforma. Além disso, é esse agente que gerencia o ciclo de vida de todos os agentes existentes. No Serviço de páginas brancas, que está relacionado ao serviço de endereçamento, cada agente recebe uma identificação (*AID*) única no ambiente.

[1] Entidade responsável pela padronização em sistemas baseados em agentes.

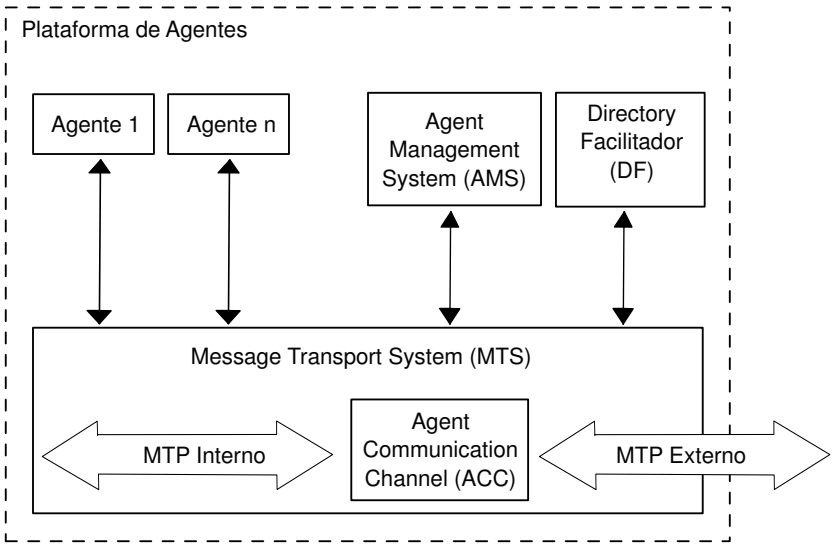

Figura A.1: Modelo de referência da FIPA para plataformas de agentes.

- *Directory Facilitator (DF)*: esse agente fornece um serviço de "páginas amarelas", que é a catalogação dos serviços oferecidos pelos agentes na plataforma.

- *Message Transport System (MTS)*: componente para o transporte das mensagens entre os agentes, onde se pode encontrar:

  - *Agent Communication Channel (ACC)*: provê a comunicação entre os agentes, e é responsável pelas as devidas traduções em trocas de mensagens entre agentes distribuídos e localizados, em ambientes heterogêneos.

  - *Message Transport Protocol (MTP)*: que utiliza-se do RMI[2] para toca de mensagens entre as máquinas virtuais Java (com IIOP[3] ou HTTP[4] quando entre plataformas diferentes)

O JADE utiliza-se dessa mesma arquitetura especificada pela FIPA, e um exemplo de um sistema multiagente em JADE pode ser visto na Figura A.2. Cada *container* é uma máquina virtual Java diferente, e agrupa um conjunto de agentes. As máquinas virtuais podem estar em hospedeiros (*hosts*) diferentes e heterogêneos, graças ao serviço do *ACC*.

[2] *Remote Method Invocation* (Java RMI)
[3] *Internet Inter ORB Protocol*
[4] *Hypertext Transfer Protocol*

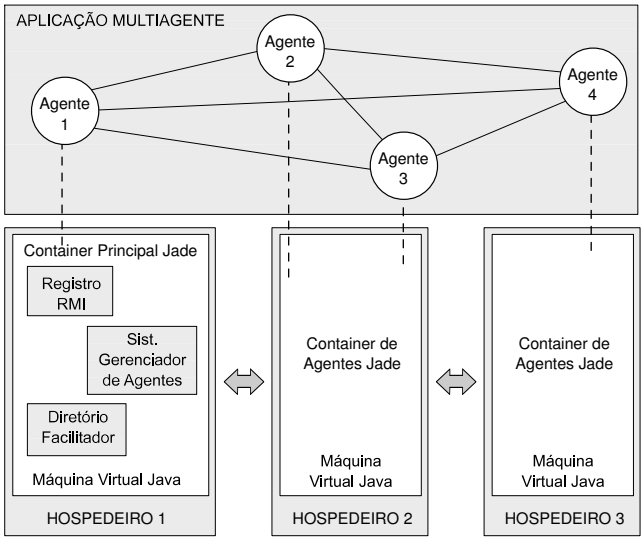

Figura A.2: Arquitetura JADE.

A.1.2 Agentes JADE

A Plataforma JADE oferece um arcabouço para construção de agentes, onde a tarefa de trocar mensagens, registro nas páginas amarelas, endereçamento dos agentes estão disponíveis para os usuários do arcabouço. Para que um agente JADE possa ser construído, basta que o usuário do arcabouço implemente a classe abstrata *jade.core.Agent*, que possui um único método abstrato chamado *setup()*, conforme pode mostrar a Figura A.3

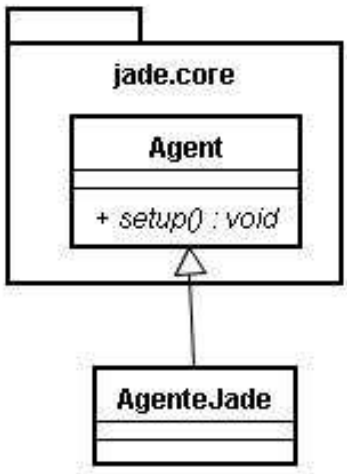

Figura A.3: Estendendo um agente JADE.

É importante frisar que é um agente JADE deve possuir conhecimento sobre três aspectos:

1. Comportamentos: cada agente pode executar diversos tipos de atividades diferentes através dos comportamentos;

2. Serviços: cada agente deve ter a capacidade de i) cadastrar os seus serviços e/ou ii) buscar serviços de outros agentes. O objetivo deste é devido a necessidade de comunicação entre os agentes;

3. Mensagens: para que um agente possa se comunicar com outro, ele deve ter a capacidade de receber e/ou enviar mensagens para outros agentes.

Nas subseções que seguem, comportamentos, serviços e mensagens são abordados.

A.1.3 Comportamentos

Cada agente JADE pode executar mais de uma tarefa, as quais podem ser mapeadas em comportamentos. O arcabouço faz todo o controle da execução dos comportamentos, que podem ser executados concorrentemente. JADE possui 9 comportamentos padrões que estendem da classe *jade.core.behaviours.Behaviour*, dentre os quais, destacam-se *TickerBehaviour* (sendo executado periodicamente, através de um tempo pré-estabelecido) e *OneShotBehaviour* (sendo executado apenas no momento que é chamado), como mostrado na Figura A.4. Quando um comportamento é atribuído a um agente, o mesmo solicita ao comportamento a execução do método *action()*, que se repete enquanto o método *done()* estiver retornando *false*.

No Código Anotado A.1 um comportamento é adicionado ao agente, chamado *BehaviourAgent*, através do método *addBehaviour(Behaviour behaviour)* da classe *jade.core.Agent*. A implementação desse comportamento segue em seguida.

```
1 public class AgenteJade extends Agent{
2     public void setup(){
3         this.addBehaviour(new BehaviourAgent(this));
4     }
5 }
```

Código Anotado A.1: Associando um comportamento a um agente.

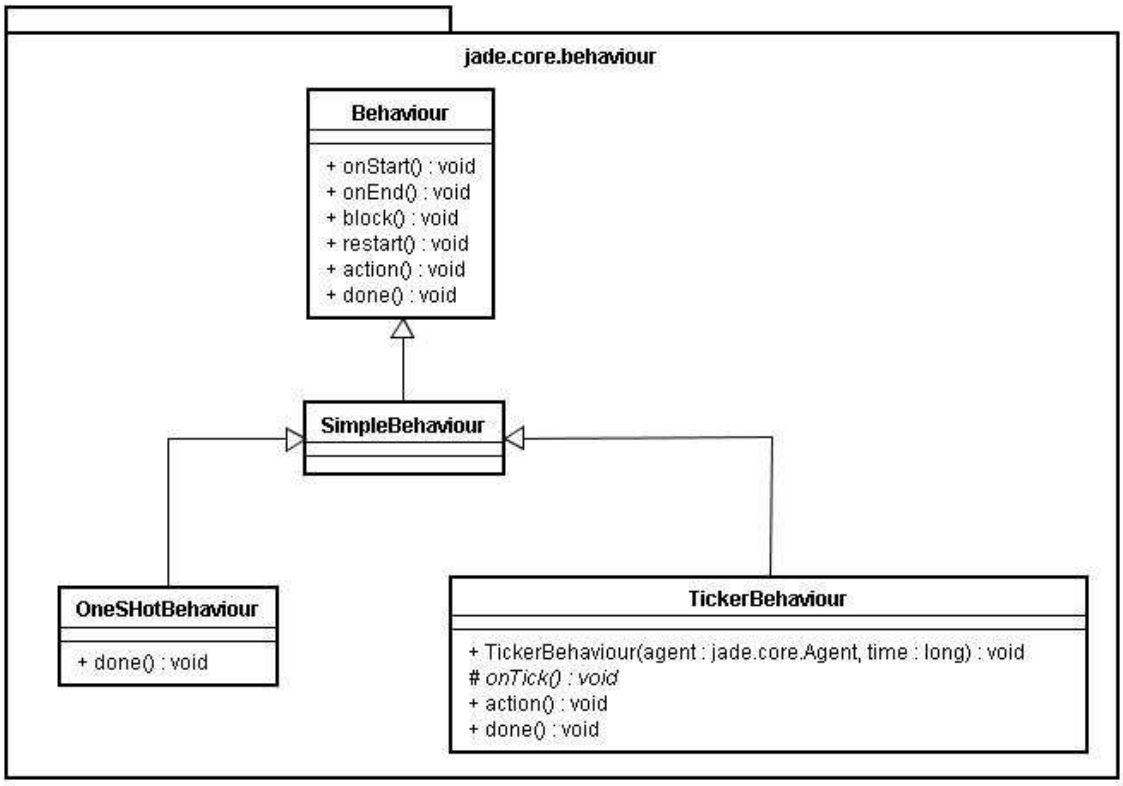

Figura A.4: Estendendo um comportamento Jade.

A.1.4 Serviços

Para que dois agentes possam trocar mensagens, é necessário que cada um deles conheça o endereço JADE um do outro, que está abstra´´ido na classe *AID*. Para que isso aconteça de forma dinâmica, pelo menos um dos agentes deverá ser cadastrado nas páginas amarelas (fornecido pelo agente *Directory Facilitator*), descrevendo quais seus serviços. No Código Anotado A.2, é criado um método que tem por objetivo cadastrar serviços nas páginas amarelas.

Esse cadastro é executado pelo método *insertIntoYellowPages(String serviceType, String serviceName)*, que executa os seguintes passos:

1. Na linha 3, é criado uma instância de *DFAgentDescription*. Essa classe representa uma descrição de um agente para as páginas amarelas;

2. Na linha 5, foi criado uma instância de *ServiceDescription*. Essa classe representa uma abstração de um serviço nas páginas amarelas. Nas linhas 7 e 8, foram atribuídas a essa instância, o nome e o tipo de serviço desejado;

```
1      public boolean insertIntoYellowPages
2                  (String serviceType, String serviceName){
3          DFAgentDescription dfAgentDescription =
4                                          new DFAgentDescription();
5          ServiceDescription serviceDescription =
6                                          new ServiceDescription();
7          serviceDescription.setName(serviceName);
8          serviceDescription.setType(serviceType);
9          dfAgentDescription.addServices(serviceDescription);
10         try{
11             DFService.register(this,dfAgentDescription);
12             return true;
13         }catch(FIPAException e){
14             return false;
15         }
16     }
```

Código Anotado A.2: Cadastro nas Páginas Amarelas.

3.Na linha 9, a descrição do serviço (*serviceDescription*) foi atribuída à instância da classe *DFAgentDescription*, chamada *dfAgentDescription*;

4.Na linha 11, o serviço descrito em *dfAgentDescription* foi registrado em *DFService*. A classe *DFService* é uma abstração do agente páginas amarelas (*Directory Facilitator*).

Além de cadastrar um serviço nas páginas amarelas, o agente pode fazer consultas para saber quais agentes são responsáveis por determinados serviços. Tal implementação é mostrada no Código A.3

Esse busca é executada pelo método *getAIDsFromYellowPages(String serviceType, String serviceName)*, que executa os seguintes passos:

1.Da linha 3 a 8, ocorre da mesma forma que o cadastro de um serviço nas páginas amarelas;

2.Nas linhas 11 e 12, o método *DFService.search* faz uma busca pelos agentes que fornecem um determinado serviço;

3.Das linhas 13 a 20, é feito o tratamento do método *DFService.search*, que retorna um Array de AID's, onde cada AID representa um endereço JADE;

```
1      public AID[] getAIDsFromYellowPages(String serviceType,
2                                                  String serviceName){
3          DFAgentDescription dfAgentDescription = new DFAgentDescription();
4          ServiceDescription serviceDescription = new ServiceDescription();
5
6          serviceDescription.setName(serviceName);
7          serviceDescription.setType(serviceType);
8          dfAgentDescription.addServices(serviceDescription);
9
10         try{
11             DFAgentDescription results[] =
12                     DFService.search(this,dfAgentDescription);
13             AID aid[] = new AID[results.length];
14             for (int i = 0; i < results.length; i++) {
15                 aid[i] = results[i].getName();
16             }
17             return aid;
18         }catch(FIPAException e){
19             return null;
20         }
21
22     }
```

Código Anotado A.3: Cadastro nas Páginas Amarelas.

A.1.5 Mensagens

A interação entre os agentes ocorre através da troca de mensagens. Para que um agente possa interagir, o mesmo deve criar um comportamento que tem a capacidade de receber e/ou enviar mensagens para outros agentes, através de uma linguagem de comunicação entre agentes[5], como mostrado no Código A.4.

```
1  public class MessageReceiver extends TickerBehaviour{
2      public MessageReceiver(Agent agent){
3              super(agent,1000);
4      }
5
6      public void onTick(){
7          ACLMessage receivedMessage = agent.receive();
8          if (receivedMessage != null){
9               //Treatment of the message
10         }
11     }
12 }
```

Código Anotado A.4: Comportamento (*Behaviour*) *MessageReceiver*.

[5]Esta linguagem é chamada Agent Communication Language (ACL)

O comportamento *MessageReceiver* foi estendido da classe *TickerBehaviour*, cuja qual possui um único método abstrato chamado *onTick()*, que deverá ser implementado. Esse método é executado a cada período de tempo definido em seu construtor. Neste caso específico (no código anterior), a cada um segundo (1000 milisegundos) o comportamento *MessageReceiver* executa as seguintes tarefas:

1. Na linha 7, o comportamento solicita ao seu agente que receba a última mensagem, através do método *agent.receive()*.

2. Na linha 8, o comportamento verifica se a mensagem recebida não é nula. Não sendo, ele tratará a mensagem a sua maneira.

Para que o tratamento seja feito, o agente precisa recuperar o conteúdo. O conteúdo de uma mensagem pode ser adquirido pelo agente de duas formas:

1. O agente recebe o conteúdo em uma String, através do método *getContent()*. Para enviar o conteúdo de uma String, deve-se usar mo método *setContent()*;

2. O agente pode receber o conteúdo em um Objeto, através do método *getContentObject()*. Tal objeto só irá funcionar se o mesmo implementar a Interface *Serializable*. Para enviar o conteúdo de uma String, deve-se usar mo método *setContentObject()*;

Por último, segue o código do agente *AgentMessageSender*, exibido no Código Anotado A.5, que está enviando uma mensagem para o agente *AgentMessageReceiver*, que executa essa tarefa segundo os seguintes passos:

1. Na linha 5, o agente *AgentMessageSender* procura nas páginas amarelas quem está recebendo mensagens com o nome *AgentMessageReceiver* e do tipo *ReceiveMensage*, através do método *getAIDsFromYellowPages*, mostrado no Código A.3.

2. Na linha 7, uma mensagem no formato JADE é representada por uma instância da classe *ACLMessage*, para ser posteriormente enviada ao agente *AgentMessageReceiver*.

3. Na linha 10, o agente *AgentMessageSender* atribui o endereço do remetente à mensagem, nesse caso, o endereço do agente *AgentMessageReceiver*.

4.Na linha 11, a mensagem recebe como conteúdo a String contendo *"Message Content"*.

5.Na linha 12, o agente *AgentMessageSender* envia a mensagem, através do método *send* disponibilizado pela classe *Agent*.

```
1  public class AgentMessageSender extends Agent{
2
3      public void setup(){
4
5          AID[] aid = this.getAIDsFromYellowPages
6                          ("ReceiveMessage","AgentMessageReceiver");
7          ACLMessage message = new ACLMessage();
8
9          if (aid[0] != null){
10             message.addReceiver(aid[0]);
11             message.setContent("Message Content");
12             this.send(message);
13         }
14
15     }
16
17 }
```

Código Anotado A.5: Agente que envia mensagens.

A.1.6 Pacote *jade.tools*

O pacote jade.tools contém algumas ferramentas úteis que facilitam a administração e implementação dos agentes na plataforma.

Remote Monitoring Agent (RMA):console responsável pela administração e controle da plataforma. O JADE mantém coerência entre os RMAs através de envio de *multicasting*[6] entre eles.

Dummy Agent:ferramenta de monitoramento e *debug* que possui funções típicas tais como enviar, receber e armazenar mensagens ACL (*Agent Communication Language*).

Sniffer Agent:ferramenta utilizada para *debug* que rastreia e salva em arquivo a comunicação entre agentes. É capaz de interceptar as mensagens ACL em trânsito e exibir uma

[6]Transmissão de dados de um computador central a vários outros numa rede.

anotação gráfica muito semelhante ao UML[7], que é de grande utilidade para depuração da sociedade de agentes.

Introspector Agent:ferramenta que permite monitorar o ciclo de vida dos agentes, suas mensagens ACL trocadas e os behaviours em execução.

SocketProxy Agent:agente simples que age como uma porta bidirecional entre a plataforma e uma conexão TCP/IP. Este agente é útil para manipular *firewalls* ou prover interações entre a plataforma e *applets java*.

DF GUI: Modo gráfico para gerenciar a base de conhecimento do serviço de "páginas amarelas".

A.2 OWL

OWL[8] (OWL, 2006) é a linguagem de ontologias utilizada para Web Semântica, a qual foi padronizada pela W3C (W3C, 1994). OWL possui um alto poder de expressividade e capacidade de inferência, sendo garantido por três dialetos diferentes (HORRIDGE et al., 2004):

- *OWL Lite*: equivale a uma sub-linguagem bastante simples, podendo ser usada quando somente uma simples hierarquia ou restrições são necessárias;
- *OWL-DL*[9]: possui maior expressividade que *OWL-Lite*, sendo baseado na Lógica de Descrição [10]. Com *OWL-DL* é possível verificar inconsistências na ontologia e inferir a classificação da hierarquia;
- *OWL FULL*: representa a mais expressiva das sub-linguagens OWL, devendo ser utilizada em situações onde a necessidade de expressividade é maior que a garantia de computabilidade.

Os componentes presentes em OWL são três, sendo eles: *i) Classes*: equivalendo a uma representação concreta de termos. Além disso, são representadas como um conjunto de

[7] *Unified Modeling Language*
[8] Do inglês: Ontology Web Language
[9] DL significa *Description Language*, ou seja, lógica de descrição.
[10] Lógica de Descrição representa uma parte da lógica de primeira ordem.

contém indivíduos; *ii) Propriedades*: equivalem a relações binárias de indivíduos; e *iii) Indivíduos*: representam objetos em um determinado domínio que foi formalizado, sendo referenciados como instâncias de classes.

A.2.1 Propriedades

Um destaque maior será dado às propriedades, pois possuem características fundamentais para a construção de qualquer ontologia em OWL. Uma propriedade pode ter diversas características, dentre as quais citam-se:

Funcional:uma propriedade é funcional quando o seu indivíduo pode possuir no máximo um valor;

Funcional inversa:significa que a propriedade inversa é funcional, ou seja, significa que esta propriedade pode ter no máximo um indivíduo através da propriedade inversa;

Transitiva:se uma propriedade é transitiva, isto quer dizer que em situações onde um indivíduo **a** tem relação com um indivíduo **b**, e **b** tem relação com **c**, então, pode-se inferir que o indivíduo **a** tem relação com o indivíduo **c**;

Simétrica:propriedades simétricas significam que, quando um indivíduo **b** está relacionado com **a**, quer dizer que o indivíduo **a** também está relacionado com o indivíduo **b**.

Além disso, as propriedades possuem possuir restrições, especificadas em lógica de descrição. Os tipos de restrições são:

Quatificadores:as restrições deste tipo podem ser duas: i) quantificador existencial ($\exists$), tendo que possuir ”pelo menos um“ indivíduo e ii) quantificador universal ($\forall$), sendo lido como ”somente“ estes tipos de indivíduos;

Cardinalidade:as restrições deste tipo são de três tipos: i) Cardinalidade mínima, sendo dito qual é a mínima quantidade de indivíduos da propriedade; ii) Cardinalidade Máxima, sendo dito qual é a máxima quantidade de indivíduos da propriedade e iii) Cardinalidade Exata, informando quantos indivíduos exatamente a propriedade deve ter.

***hasValue*:**esta restrição, denotada pelo símbolo ($\ni$), descrevendo que um indivíduo, através de uma propriedade, tem pelo menos um relacionamento com outro indivíduo.

A.3 Protégé

Protégé (PROTéGé, 2006) é um ambiente integrado para a modelagem e a aquisição de conhecimento voltado para o desenvolvimento de ontologias (Vide Figura A.5), desenvolvido pelos pesquisadores do *Stanford Medical Informatics* (SMI, 2006). O desenvolvimento de ontologias no protégé pode ser feito de duas maneiras, utilizando o i) *Protégé-Frames*: o usuário pode construir e povoar ontologias que são baseadas em *frames*, em conformidade com *Open Knowledge Base Connectivity* (OKBC, 1995) criar frames ou ii) *Protégé-OWL*: o usuário pode criar ontologias para a Web Semântica, em particular em W3C's *Ontology Web Language*.

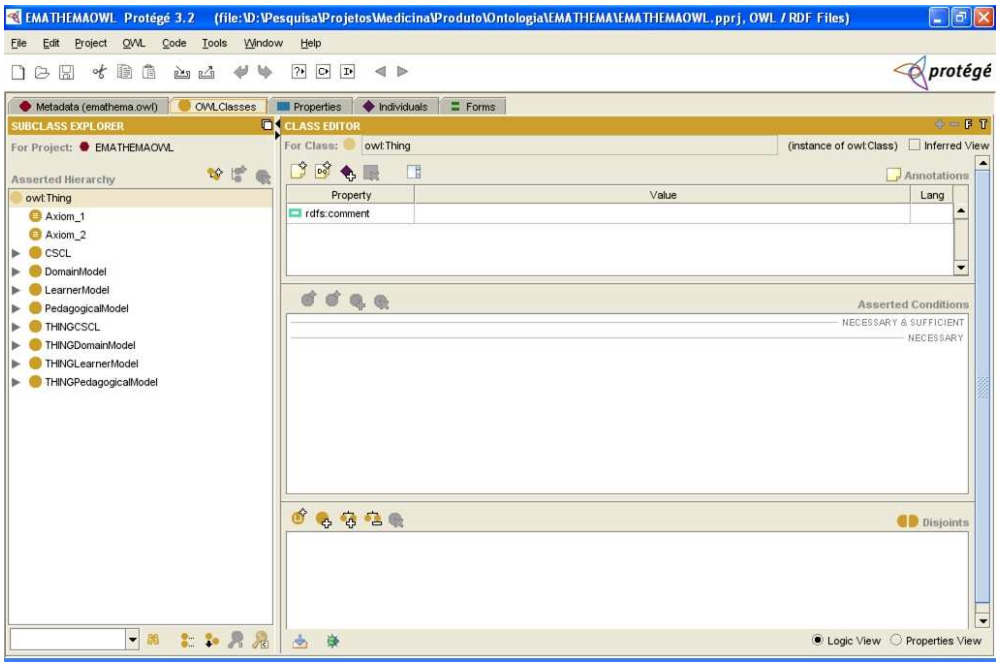

Figura A.5: Tela Inicial do Protégé.

Esta ferramenta é usada por desenvolvedores de sistemas e especialistas do domínio para construir sistemas baseados em conhecimento, pois oferece vantagens como:

- Livre;
- Código aberto;
- Pode ser exportado em uma variedade de formatos, como RDF(S), OWL, XML Schema, outros;
- Baseado em Java;

- É extensível e disponibiliza um ambiente baseado em plug-ins, tornando-o flexível para o desenvolvimento de aplicação e rápida prototipação;
- Disponibiliza diversos plug-ins para uso;
- Possui documentação abordando o uso da ferramenta;
- Possui listas de discussões e uma equipe de desenvolvimento dando continuidade na geração de novas versões;
- É suportado por uma comunidade de mais de 62.000 (sessenta e dois mil) usuários.

Atualmente o protégé está na versão 3.2.1., onde o principal foco desta versão é o desenvolvimento de ontologias em *OWL*. Além de criar ontologias em OWL, o desenvolvedor poderá fazer o gerenciamento de mudanças, depurarem ontologias em OWL-DL, dentre outras características. O grupo de desenvolvedores responsável pelo protégé já está desenvolvendo a versão 4.x.

A.3.1 Protégé-OWL

O Protégé-OWL é uma extensão do Protégé para dar suporte a Web Ontology Language (OWL). Dentre diversas funcionalidades, o editor para o Protégé-OWL permite ao desenvolvedor:

- Carregar e salvar ontologias nos formatos OWL e RDF, como mostrado na Figura A.6;

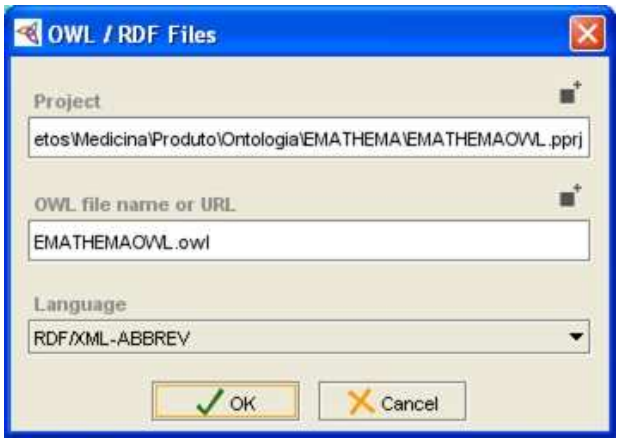

Figura A.6: Tela de Salvar do Protégé.

•Editar e visualizar classes, propriedades (Vide Figura A.7) e regras (SWRL[11]);

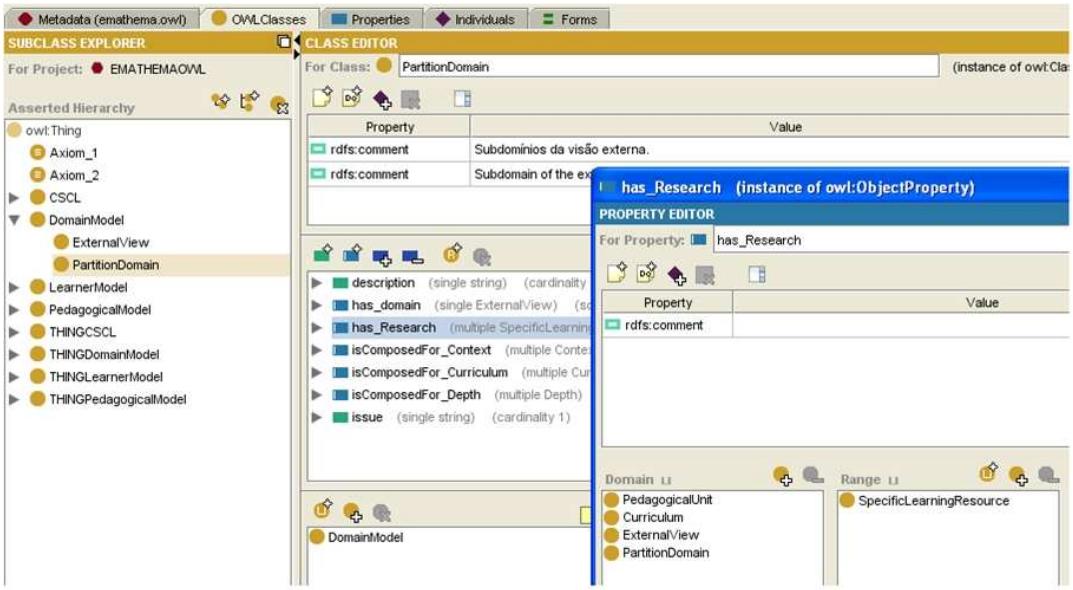

Figura A.7: Editor de Classes e Propriedades do Protégé.

•Definir características lógicas das classes através de expressões OWL-DL, como mostrado na Figura A.8;

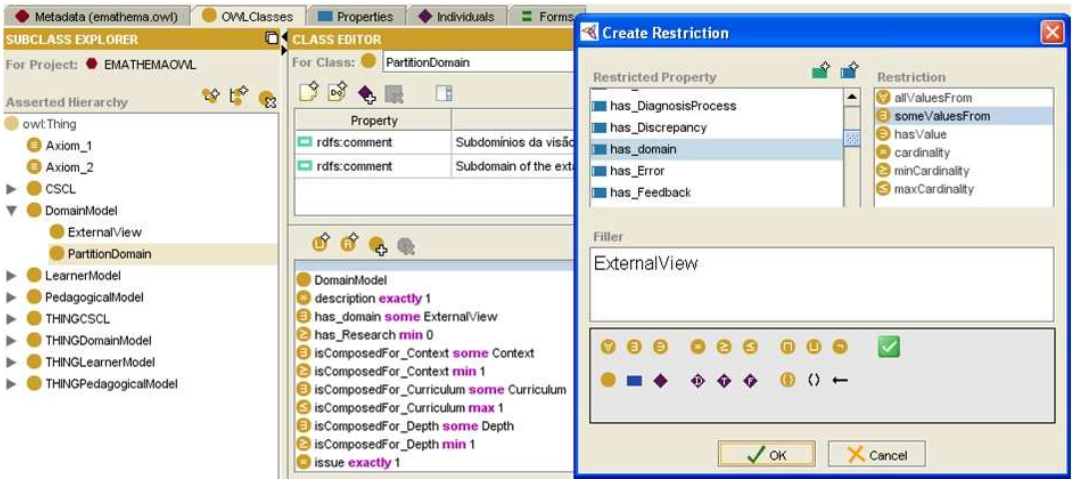

Figura A.8: Tela do Protégé com especificação em OWL-DL.

[11]Do Inglês: Semantic Web Rule Language.

•Editar indivíduos OWL (Vide Figura A.9).

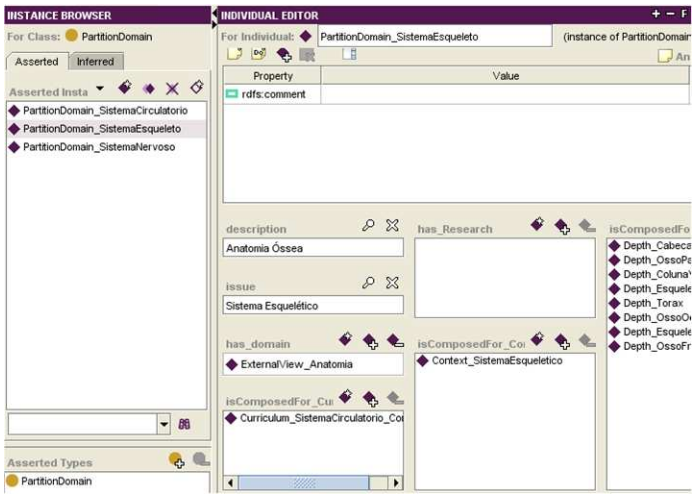

Figura A.9: Tela de Indivíduos do Protégé.

A.3.2 ProtégéOWL-API

O Protégé disponibiliza uma API para o desenvolvedor poder trabalhar diretamente da linguagem Java. Esta API provê classes e métodos para carregar e salvar arquivos OWL, consultar e manipular modelos de dados OWL e executar inferências. Uma das grandes vantagens em se utilizar esta API é devido a abstração de Jena[12], ou seja, o desenvolvedor não precisará implementar diretamente utilizando Jena, pois a API gera uma camada que facilita a manipulação de arquivos OWL. O Código A.6 aborda o carregamento e salvamento de um arquivo OWL utilizando a API do Protégé.

```
1       public void load(final String owlFile) {
2           this.errors = new ArrayList();
3           this.project = Project.loadProjectFromFile(owlFile, getErrors());
4           this.owlModel = (OWLModel) getProject().getKnowledgeBase();
5       }
6
7       public void save(){
8           this.project.save(getErrors());
9       }
```

Código Anotado A.6: Carregamento e Salvamento de um arquivo OWL.

O método *load(final String owlFile)* é responsável pelo carregamento de um arquivo

[12] Jena equivale a um *framework* para a Web Semântica (JENA,).

OWL. Os passos para o carregamento são:

1. Na linha 3, é feito o carregamento do arquivo através do método *loadProjectFrom-File(owlFile, getErrors())* da classe Project. Esta classe permite manipular projetos criados em *Protégé* [13]. Já o método possui dois atributos, sendo o primeiro o caminho do arquivo do projeto OWL e o segundo uma *Collection* que será preenchida caso algum erro ocorra no carregamento do arquivo;

2. Na linha 4, a base de conhecimento é passada para uma propriedade do tipo *OWL-Model*. *OWLModel* é uma interface equivalendo a uma base de conhecimento com métodos para manipular o arquivo OWL.

Além disso, o desenvolvedor pode fazer todo o mapeamento da ontologia em código Java. Para isso, o desenvolvedor deve seguir os seguintes passos:

1. Gerar a classe com o ”esquema“ de toda a ontologia OWL, ou seja, toda a estrutura do OWL será mapeada nesta classe. A Figura A.10 aborda onde ocorre o processo. Paro o mapeamento, o desenvolvedor deve informar o local e o pacote que a classe será salva;

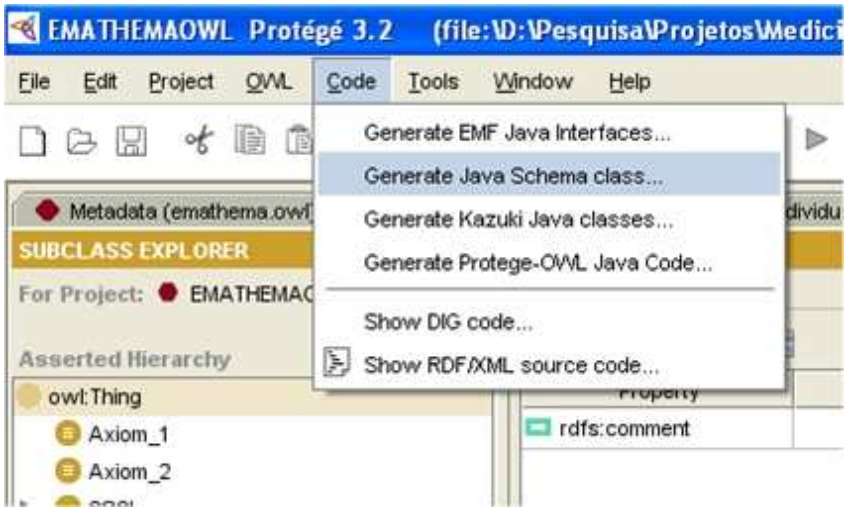

Figura A.10: Gerar a classe com o ”esquema“ de toda a ontologia OWL.

[13] Observe que a classe *Project* faz com que haja necessidade do arquivo de projeto do Protégé (*.pprj) para o carregamento do arquivo OWL.

2.Gerar as classes e interfaces. O Protégé possui um *plugin* (Vide Figura A.11) que faz o mapeamento da ontologia em classes Java.

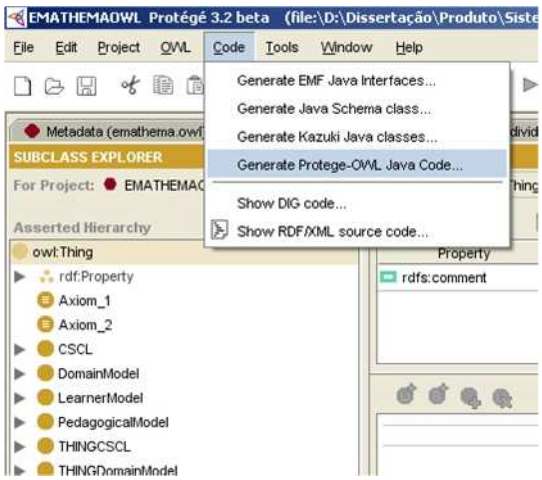

Figura A.11: Mapeamento Ontologia-Objeto através do Protégé.

Para a geração do código (Vide Figura A.12), é necessário configurar *i)* a pasta de geração do código, *ii)* o pacote e *iii)* o nome da classe de fábrica.

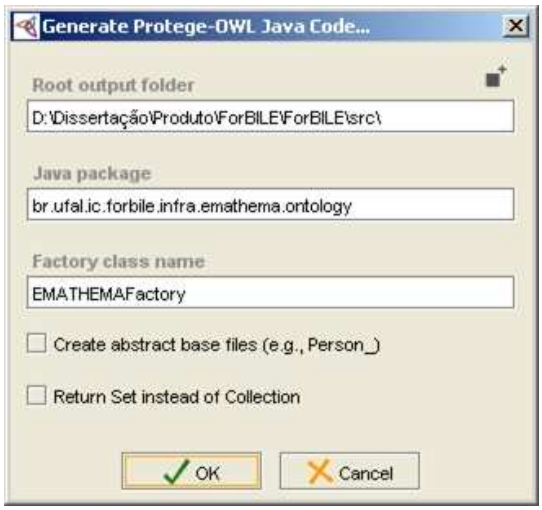

Figura A.12: Tela do Protégé com o Mapeamento Ontologia-Objeto.

É gerada, para cada classe da ontologia, uma interface[14] e uma classe concreta. A interface possui a assinatura dos métodos da classe, onde estes métodos equivalem aos *getters* e *setters*[15] dos *slots*. A classe possui as regras da classe definidas em lógica de descrição e a implementação dos métodos da interface, é mostrada na figura A.13.

[14]É importante frisar que toda interface gerada pela API herda da Interface *OWLIndividual*.

[15]São métodos padrões utilizados no desenvolvimento de software orientados a objetos, onde objetiva-se

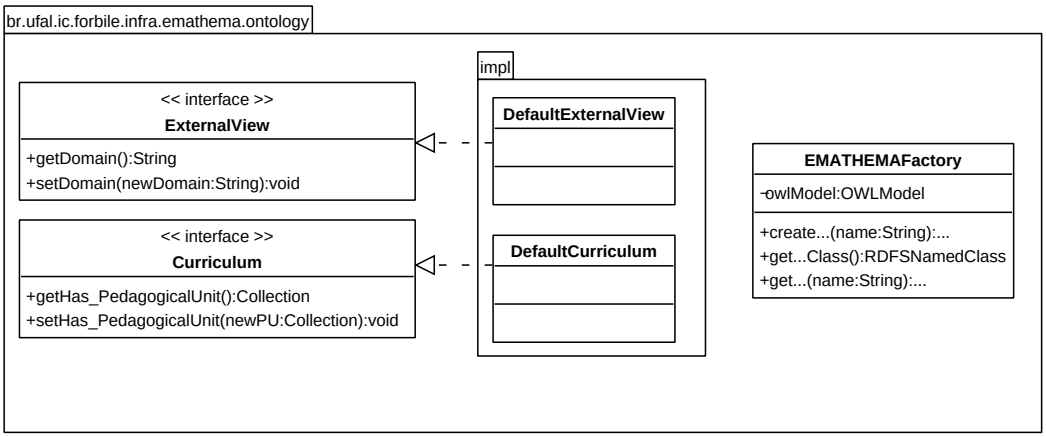

Figura A.13: Pacote com as classes do Mapeamento Ontologia-Objeto.

A classe *EMATHEMAFactory* é responsável pela construção dos objetos da ontologia. Além disso, esta classe possui o modelo OWL da ontologia (*OWLModel*). A propriedade *OWLModel* equivale a instância da ontologia. O Código A.7 mostra um trecho da Classe *EMATHEMAFactory*.

```
1  public class EMATHEMAFactory {
2
3      public EMATHEMAFactory(OWLModel owlModel) {
4          this.owlModel = owlModel;
5      }
6
7      public ExternalView createExternalView(String name) {
8          final RDFSNamedClass cls = getExternalViewClass();
9          return (ExternalView) cls.createInstance(name).as(ExternalView.class);
10     }
11
12     public ExternalView getExternalView(String name) {
13         return (ExternalView) owlModel.getRDFResource(name)
14             .as(ExternalView.class);
15     }
16 }
```

Código Anotado A.7: Classe EMATHEMAFactory gerado através da API do Protégé-OWL.

Além disso, pode-se fazer consultas através da implementação de forma bastante simples[16], sendo estas de duas formas:

1.Recuperando indivíduos de alguma classe da ontologia, como mostra o Código A.8;

atribuir e recuperar valores de propriedades.

[16]Tal simplicidade é garantida devido a camada de abstração de Jena, que não foi diretamente utilizada.

```
1       public retrievePartitionDomain(String owlFile, String individual) {
2           this.errors = new ArrayList();
3
4           this.project = Project.loadProjectFromFile(owlFile, this.getErrors());
5           this.owlModel = (OWLModel) this.getProject().getKnowledgeBase();
6
7           this.emathema = new EMATHEMAFactory(getOwlModel());
8
9           PartitionDomain c = this.emathema.getCurriculum(individual);
10      }
11
```

Código Anotado A.8: Recuperação de indivíduo da classe *PartiionDomain*.

2.Através de SPARQL[17], como mostrado no Código A.9.

```
1       public void retrieve(String querySPARQL) {
2
3           QueryResults qr = null;
4
5           qr = this.owlModel.executeSPARQLQuery(querySPARQL);
6
7       }
```

Código Anotado A.9: Recuperação de indivíduos através de SPARQL.

[17]SPARQL equivale a uma linguagem de consulta para Web Semântica (SPARQL, 2006)