Sistema Tutor Inteligente PAT2Math: Proposta de Arquitetura Multiagente

Fábio R Damasceno[1], **Anderson da Cruz**[1] **& Patrícia A. Jaques**[1]
[1]Programa Interdisciplinar de Pós Graduação em Computação Aplicada
Universidade do Vale do Rio dos Sinos
Av. Unisions, s/n - São Leopoldo - RS - BRAZIL
fabiorafaeldamasceno@gmail.com, anderson@a29.com.br, pjaques@unisinos.br

Abstract

As the development of an Intelligent Tutor System (ITS) is a quite complex, it should be modularized for its complexity reduction and better problem comprehension. A multi-agent approach has been applied in the conception of ITSs, as it allows to share specific roles in specialised agents. In this way, this article presents the analysis process steps of the ITS PAT2Math architecture, that was built under a multi-agent approach. This architecture was projected following the Prometheus Methodology.

Keywords: Multi-Agent System, Intelligent Tutoring Systems, Affective Computing

1. INTRODUÇÃO

Sistemas Tutores Inteligentes, STI (ou *Intelligent Tutoring Systems*) são sistemas computadorizados que possuem modelos instrucionais responsáveis por especificar o que lecionar, bem como estratégias de aprendizado para fazê-lo. Uma característica fortemente presente nestes sistemas é a capacidade de se adaptar dinamicamente o conteúdo apresentado ou o estilo de ensino. Tal ensino individualizado para o aluno fornece benefícios semelhantes aos do um professor particular, considerado o melhor caso para aprendizado [12].

Este trabalho apresenta a proposta de arquitetura multiagente para o STI PAT2Math *(Personal Affective Tutor To Math)*. Este sistema tem como objetivo o ensino de álgebra elementar a alunos do ensino fundamental em um ambiente capaz de considerar as emoções do aluno.

Métodos usados por uma arquitetura inteligente podem diferir daqueles usados em uma sala de aula convencional, causando conflitos de aprendizado por parte dos estudantes, tornando necessária uma análise criteriosa do projeto. O desenvolvimento do modelo cognitivo da arquitetura é considerado a tarefa mais importante e a que consome maior tempo do projeto, sendo necessário extenso trabalho e envolvendo um certo grau de dificuldade. Os tópicos descritos na seção 2 deste artigo retratam pontos importantes que devem ser considerados durante um projeto de um sistema tutor inteligente, abrangendo detalhes de conteúdo e arquitetura de tais sistemas.

O foco do presente artigo se concentra no projeto da arquitetura multiagente a ser utilizada para concretizar o sistema proposto. Maiores detalhes sobre a proposta do STI PAT2Math podem ser encontrados em [10].

Este artigo encontra-se organizado como segue. A seção 2 aborda os tópicos sobre modelagem de STI, área em que o PAT2Math se enquadra. A seção 3 apresenta uma introdução a ontologias, como podem ser representadas e seu benefícios. Na seção 4 são comentadas as características e aspectos positivos da metodologia para desenvolvimento do sistema multiagente (SMA) *Prometheus*. Na seção 5 é apresentada a arquitetura proposta para atender os objetivos do projeto, mostrando os agentes, suas respectivas bases de dados, mensagens e papéis desempenhados. Os protótipos desenvolvidos são apresentados na seção 6. Trabalhos relacionados são apresentados na seção 7. Conclui-se este estudo na seção 8.

2. MODELAGEM DE SISTEMAS TUTORES INTELIGENTES

Dentro de um STI, o aluno possui um ambiente interativo, onde pode melhor exercitar os passos das lições que acompanhou e exercitou. Com o objetivo de proporcionar a devida ajuda e esclarecimento de conteúdos aos seus usuários e organizar os dados do sistema, o conheci-

mento é dividido internamente nos STIs em três modelos: *Expert*, *Student* e *Instructional Model* [14].

No *Expert Model* são mapeados todos os conhecimentos relativos aos conteúdos propostos para aprendizado. Trata-se da representação do conhecimento de um especialista no assunto, assim permitindo que o STI compare as habilidades e saberes do estudante com este modelo, avaliando o que este sabe sobre o assunto.

Em casos onde o ambiente é mais realístico, o modelo pode avaliar cada uma das ações realizadas pelo aluno, usando, por exemplo, como comparação máquinas de estado que representam os estágios do aprendizado [14].

Por fim, o *Instructional Model* habilita o STI a ensinar o aluno, através de estratégias adequadas a cada situação, se baseando em dois elementos, informações sobre o aluno no seu respectivo modelo e o assunto sendo tratado no sistema. Por exemplo, avaliando o cenário onde o aluno é considerado iniciante em um assunto, o sistema poderia mostrar procedimentos passo-a-passo e explicações mais detalhadas dos conceitos básicos em um primeiro momento. Posteriormente, o sistema deixaria o estudante explorar simulações e exercícios e somente ofereceria ajuda quando requisitado.

Atualmente, muitos dos sistemas de ensino-aprendizagem têm usado a tecnologia de agentes na sua concepção. Nesta abordagem orientada a agentes, a arquitetura modular do STI é substituída por uma sociedade de agentes, denominada Sistema Multiagente (SMA), que trabalham de forma cooperativa e em *background* como parte da arquitetura do ambiente educacional. Esses agentes são usados para facilitar a construção modular da arquitetura de sistemas tutores e comunicação entre estes módulos. Um STI formado por agentes também é denominado de Agente Pedagógico ou Ambiente Inteligente de Aprendizagem.

Na literatura de Inteligência Artificial Distribuída (IAD) encontram-se diversas acepções para o termo agente. Uma compilação dessas acepções nos leva a definição proposta por Shoam [17]. Segundo o autor, um agente de software é uma entidade que funciona autônoma e continuamente em um ambiente particular sempre habitado por outros agentes e processos. O termo autonomia não é muito preciso e é empregado no sentido que o agente realiza as suas atividades sem a intervenção constante de uma pessoa. Uma sociedade de agentes que interagem para resolver um problema em comum é chamado de SMA.

Em um SMA, os agentes devem possuir algumas capacidades específicas para interagirem num mesmo ambiente [11] [18]. Eles devem ter conhecimento da sua própria existência, assim como da existência dos outros agentes. Eles devem ser capazes de se comunicar possuindo, para tanto, uma linguagem específica. Cada agente deverá possuir conhecimentos e habilidades para executar uma determinada tarefa e, portanto, devem cooperar para atingir um objetivo global. Todos esses aspectos fazem importante a utilização de uma metodologia para a concepção da arquitetura multiagente, tal como, a metodologia *Promotheus* (ver seção 5).

Uma área de estudo que está ganhando força e despertando interesse da comunidade acadêmica é a inserção de Ontologias como ferramenta para organização de conhecimento e tomadas de decisão em STIs. A análise desse recurso é feita na seção a seguir [5].

3. ONTOLOGIAS

Ontologia (com, "O"maiúsculo) é a área da Filosofia que estuda a natureza da existência e a estrutura da realidade, atuando como uma especificação explícita de uma conceitualização [4]. Já o termo ontologia (com, "o"minúsculo) se resume a categorização de tipos de elementos, relacionando estes com domínio estudado. Trata-se de um termo frequentemente utilizado para se referir a compreensão semântica, a estrutura conceitual do conhecimento, compartilhada por pessoas que participam em dado domínio. Em sistemas de Inteligência Artificial, o que existe de fato é o que pode ser representado [4]. Já em um ambiente da Web, no entanto, uma ontologia não é simplesmente um *framework* conceitual, mas sim uma estrutura concreta e sintática que modela a semântica de um domínio, em linguagem de máquina, compreensível por ela [5].

Atualmente a Web é estruturada para lidar com pessoas, termos de domínio e ainda *tags* HTML que são claramente compreensíveis por desenvolvedores e usuários, no entanto não possuem sentido algum para sistemas computadorizados, aplicações e/ou agentes. A estrutura XML vem mudando este conceito, porém seu *layout* lida primariamente com estrutura física dos documentos, além de sofrer com a falta de semântica presente em suas *tags*, que poderiam ser o meio de compreensão computacional [5].

É importante ressaltar que ontologias não são uma novidade no ambiente Web. Na realidade, um esquema *metadata* é na verdade uma ontologia especificando as características do conjunto de atributos físicos e/ou conceituais de determinados recursos [5]. O diferencial está na construção de linguagens e ferramentas de inferência de conhecimento, visto que uma ontologia pode especificar a semântica de recursos.

3.1. RESOURCE DESCRIPTION FRAMEWORK (RDF) E RDF VOCABULARY DESCRIPTION LANGUAGE SCHEMA (RDFS)

Embora a hierarquia não seja uma característica determinante e sempre presente em ontologias, se trata de

um importante componente no seu modelo de representação, formalmente definido pelo *Resource Description Framework* (RDF) e *RDF Vocabulary Description Language Schema* (RDFS). Tais modelos suportam reuso de elementos de qualquer ontologia ou esquema de metadados que pode ser identificado por um URI (*Uniform Resource Identifier*) [5].

O RDF define um modelo e conjunto de elementos para descrever recursos baseados em propriedades nominais e valores. Adicionalmente ele provê uma sintaxe que permite a qualquer comunidade de descrição de recursos criar uma representação específica do domínio com as suas respectivas semânticas, além da incorporação de elementos de distintos esquemas de metadados. Esse modelo, e sua sintaxe, pode ser usado para codificar informação em formato de linguagem de máquina, para troca de dados entre aplicações e processamento da semântica. O RDFS adicionalmente complementa e estende o RDF definindo uma linguagem de máquina declarativa e processável, que pode ser usado para formalmente definir uma ontologia ou conjunto de classes e suas respectivas propriedades. De forma cooperativa, fornecem um modelo sintático e estrutura semântica para definir ontologias processáveis por máquina e esquemas de metadados, além de permitir interoperabilidade de estruturas representativas entre conjuntos heterogêneos de recursos [5].

Uma ontologia RDFS é diferente das taxonomias e estruturas de classificação clássicas, no entanto, no topo dos níveis da hierarquia, a classe recurso e suas filhas classe e propriedade, não são determinadas pelo domínio da ontologia, e sim pelo esquema RDFS [5]. Cada elemento presente nesta ontologia é um tipo de classe ou propriedade. Relações entre estas são potencialmente multihierárquicas, ou seja, uma classe podendo ser filha de uma ou mais classes superiores.

Comparando tais estruturas com as ontologias RDFS percebe-se que a diferença mais significativa consiste em que estas últimas definem conjuntos de elementos que podem receber valores, para representar características físicas e conceituais de um determinado recurso, ao contrário das estruturas classificativas simples, que lidam com entidades que são atribuídas a classes hierárquicas do sistema, não podendo também lidar com regras de inferência sobre o conteúdo.

Uma ontologia não pode ser considerada como uma taxonomia, um esquema classificativo ou ainda um dicionário. Trata-se de um sistema representativo único, que integra em uma única estrutura as características de vários meios de representação, como os três últimos citados. A ontologia fornece a semântica básica para esquemas de metadados e facilita a comunicação entre sistemas e agentes através de um modelo conceitual de comunidade de usuários padronizado. Tais fatores descritos fornecem o fundamento conceitual que torna o objetivo da Web Semântica possível [5].

3.2. Benefícios do Uso de Ontologias

Uma ontologia no cenário Web pode oferecer um meio conciso e sistemático para definir a **semântica** dos recursos presentes na rede. A visão desta Web Semântica é voltada para a distribuição de dados e serviços, definidos e unidos de uma maneira que podem ser usados por máquinas e não apenas para propósitos de visualização, além de permitir a automação, integração e reusabilidade de dados e serviços entre várias aplicações [4]. Uma ontologia especifica conceitos do domínio relevantes, possíveis relações e propriedades destes. Dessa forma, proporciona a possibilidade de processar recursos baseando-se na interpretação do seu conteúdo, ao invés da estrutura física do mesmo.

As informações pedagógicas relevantes contidas em uma ontologia podem aprimorar ***Web Services*** em geral, aumentando sua precisão. Isso se deve à possibilidade de se prever durante o design dos mesmos, funcionalidades distintas dependendo do tipo de recurso sendo abordado, analisando-se as informações de uma ontologia. A composição de serviços também é beneficiada com estas informações, tendo em vista que o serviço *requester* pode necessitar de diferentes ações de um provedor, com base no tipo instrucional do recurso. Por fim, tem-se a interoperabilidade facilitada, pois na teoria cada sistema poderia proporcionar o seu próprio serviço especializado e fazer uso de serviços oferecidos por outros [20].

Em um sistema de ensino, um **Gerador de Cursos** une recursos de aprendizagem em um currículo, levando em consideração o estado e objetivos do aprendiz, suas preferências, entre outros. Se recursos extras são mapeados por uma função instrucional externa ao sistema, podem ser incluídos em cada um dos currículos disponíveis. A seleção de determinados conjuntos de recursos também pode ser mais precisa com o uso conjunto de tal ontologia [20].

Na arquitetura de um sistema de ensino, que armazena preferências pessoais e informação sobre o estudante em um **Modelo de Estudante**, tais ontologias podem utilizar a informação contida nelas para a atualização mais precisa dos dados referentes ao aluno [20].

4. Metodologia *Prometheus*

A metodologia *Prometheus* é uma abordagem consistente para construção de sistemas multiagente, consistida de três fases distintas: Especificação do Sistema, *Design* Arquitetural e *Design* Detalhado.

Na primeira fase, Especificação do Sistema, são analisados os fatores de entrada (*inputs*), saída (*outputs*), e repositórios de dados (*data sources*), sejam compartilha-

dos ou não. Posteriormente, na fase de *Design* Arquitetural, usam-se os *outputs* para determinar que agentes irão existir no sistema, bem como as suas interações. Tendo isto concluído, a fase de Design Detalhado pode ser iniciada, onde é avaliado internamente como cada agente irá concluir as suas tarefas [15].

O desenvolvedor ao utilizar esta metodologia deve considerar alguns fatores importantes na modelagem do seu respectivo SMA [15]:

- •Informações provenientes do ambiente do sistema são consideradas percepções do agente;
- •Mecanismos que podem afetar este ambiente são considerados ações que o agente pode tomar;
- •Um evento não é considerado igual a uma percepção, trata-se de um conceito de maior importância para um agente;
- •Nem tudo que ocorre no ambiente faz o agente alterar seus planos para alcançar um objetivo;
- •Percepções necessitam de processamento por parte do agente para se tornarem um evento; e
- •Tal processamento deve ocorrer no agente para ser considerada uma metodologia *Prometheus*.

A metodologia *Prometheus* foi selecionada para este projeto por fornecer uma modelagem concreta de fatores relevantes a um SMA. Adicionalmente, a *Prometheus* fornece uma interface sólida para a fase de *Design* da metodologia, tornando claro para uma equipe de desenvolvimento os avanços realizados em cada versão do sistema.

5. Trabalho Proposto

Existem cinco agentes que compõem atualmente a arquitetura multiagente deste projeto. Estes agentes correspondem aos módulos da arquitetura tradicional de um sistema tutor inteligente. São eles: Agente Interface (AI), Agente Domínio (AD), Agente Tutor (AT), Agente Modelo de Aluno (AMA) e Agente Modelo Cognitivo (AMC). O papel do Tutor é analisar os atributos do estudante e selecionar uma explicação apropriada para o mesmo. O Agente de Domínio (AD) varre a sua base, onde todo o conhecimento e tópicos estão mapeados, enviando estes dados para o Tutor quando requisitado. O Agente Modelo de Aluno monitora as informações sobre o usuário conforme este interage com o sistema. O Agente do Modelo Cognitivo (AMC) atua como um especialista, sabendo resolver equações algébricas, tendo como papel verificar a consistência dos elementos de uma equação e resolver ela. Por fim, o AI é responsável por unir os dados sobre ações do usuário dentro do sistema e, posteriormente, enviá-los para tomada de decisão do Tutor. Adicionalmente, este AI é encarregado de mostrar tópicos e informações na tela do sistema.

O diagrama, que pode ser visualizado na Figura 1, explica relação entre os agentes da arquitetura proposta, lembrando que o foco da implementação construída se concentra na interação do Agente Tutor (AT) com o AMC. Percebe-se com esta figura, a importância do AT, pois este atua como centro dos demais presentes na arquitetura, tomando decisões, repassando e processando dados.

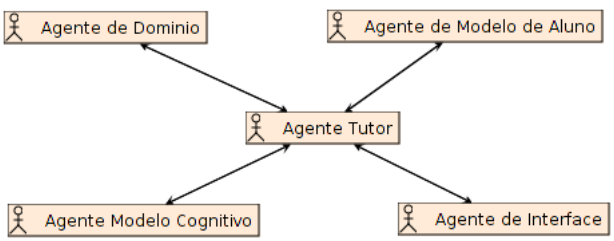

Figura 1. Diagrama geral dos agentes

O AD consulta na sua base de dados Objetos de Aprendizado conforme o AT solicita. O Agente de Modelo de Aluno pesquisa na sua respectiva base de informações dados sobre o aluno que atualmente utiliza o sistema, conforme o AT requisita informações, para criar um plano de ensino por exemplo. Já o AI recebe informações do Tutor para mostrar na tela, bem como informa a ele dados de uso do sistema, para que sejam encaminhados ao AMA, que atualiza a base de informações dos estudantes. Por fim o AMC retorna passos de solução de equações algébricas, analisando se determinados passos realizados pelo estudante fez estão corretos ou não.

5.1. Papéis dos Agentes

Esta seção apresenta a relação de papéis no sistema que cada um dos agentes presentes na arquitetura desempenha. De um total de cinco agentes, existem nove papéis que são trabalhados, sendo o AMA o que mais possui tais funções.

O diagrama que pode ser visualizado na figura 2, mostra a relação de papéis no sistema que cada um dos agentes presentes na arquitetura desempenha. De um total de cinco agentes, existem nove papéis que são trabalhados, sendo o AMA o que mais possui tais funções.

O Agente de Interface (AI) atua como:

- **Expositor de Dados para o Aluno:** Toda e qualquer informação que o AT julgue necessário mostrar ao aluno é repassado a este agente, que mostra tais informações no formato especificado, seja na tela do computador ou em um navegador de celular;

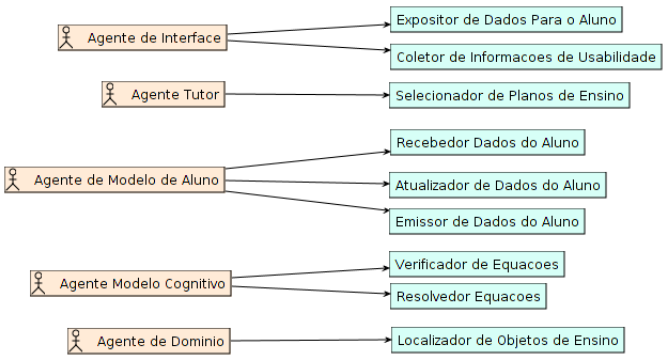

Figura 2. Diagrama de relação entre agentes com seus respectivos papéis

- **Coletor de Informações de Usabilidade:** Toda interação do usuário, que o Agente considere relevante, é mapeada e repassada ao Tutor, para que seja encaminhada à base de Modelo de Aluno. Outra característica é a conclusão de lições/exercícios, que devem ser atualizados como novos conhecimentos que o estudante em questão possui.

O Agente de Modelo de Aluno (AMA) atua como:

- **Recebedor de Dados do Aluno:** Quando o AT recebe uma informação relevante sobre o aluno através do AI, este encaminha ao AMA.
- **Atualizador de Dados do Aluno:** Uma alteração sobre o estado do aluno, como ter lido com sucesso um determinado conteúdo, é alterada na Base de Modelo do Aluno através deste papel.
- **Emissor de Dados do Aluno:** Quando o AT requisita uma informação sobre o aluno que atualmente está utilizando o sistema, o Agente de AMA é responsável por encaminhar tais dados.

O Agente de Modelo Cognitivo (AMC) atua como:

- **Verificador de Equações:** Quando um determinado aluno executa um passo durante a resolução de equação, o AT se encarrega de processar essa informação e enviar uma mensagem para o AMC para que este valide-a.
- **Resolvedor de Equações:** O AT pode julgar necessário mostrar a solução completa de uma equação-problema que o estudante esteja trabalhando, enviando uma mensagem para o AMC para que este envie o conjunto de passos a serem executados para tal.

O Agente de Domínio (AD) atua como:

- **Localizador de Objetos de Ensino:** O AT do sistema pode necessitar de um objeto de aprendizado com características distintas, seja com um texto mais aprofundado, ou imagens mais explicativas. Neste momento, ele envia uma mensagem ao AD para que faça a devida varredura na Base de Domínio e retorne-o para a devida construção do plano de ensino por parte do AT.

O Agente Tutor (AT) do Sistema atua como:

- **Selecionador de Planos de Ensino:** Informações do aluno, características dos Objetos de Aprendizado, disponibilidade de recursos, entre outros fatores são levados em consideração quando o AT vai criar um plano de ensino adequado ao aluno.

5.2. Relação das Bases de Dados com os Papéis dos Agentes

Esta seção mostra a relação das bases de conhecimento com os papéis dos agentes presentes na arquitetura do projeto proposto.

- **Base Cognitiva de Resolução de Equações:** Contém as regras para resolução de uma equação algébrica, organizadas em passos. É utilizada pelo AMC para resolução e validação de passos individuais no desenvolvimento de equações. É acionada sempre que este tenta resolver e validar passos realizados por alunos na resolução de uma equação.
- **Base de Modelo Cognitivo do Aluno:** Contém todos os dados relevantes sobre o aluno, tais como quais exercícios foram resolvidos e quais lições foram lidas dentro do sistema. Trata-se de uma base extremamente importante para o AT, pois esse a consulta para a criação de um plano de ensino mais eficiente para o aluno em questão. Tal base lida com o AMC, quando este executa os papéis de Atualizar Dados do Aluno, Emissor de Dados do Aluno e Recebedor de Dados do Aluno. Lida também com o Agente de Tutor, quando este recebe dados de usabilidade do sistema por parte do aluno, na execução do papel Coletor de Informações de Usabilidade.
- **Planos de Ensino e Estratégias Pedagógicas:** São bases com elementos comuns no projeto de um plano de ensino, possui estereótipos que o AT deve considerar ao derivar um plano personalizado para um determinado aluno. Interagem diretamente com o AT quando este executa seu papel de Selecionador de Planos de Ensino.
- **Base de Domínio:** Contém todas informações e explicações sobre os variados conteúdos contidos na arquitetura tutora do sistema proposta, sendo característica de tais objetos de aprendizado diversos níveis, para alunos distintos, com fatores cognitivos diferenciados em relação a tais conteúdos.

5.3. Troca de Mensagens entre os Agentes

No diagrama que pode ser visualizado na Figura 3 são retratadas as interações entre os agentes e suas respectivas bases de dados, através de mensagens.

O Agente Tutor (AT) pode enviar as seguintes mensagens:

- **Envia dados sobre Aluno:** O AI encaminha informações sobre mudanças no comportamento e características do aluno, como um exercício concluído com sucesso que deve ser atualizado na sua base de conhecimentos.
- **Consulta Base sobre Informações do Aluno:** O AT pode requisitar ao AMA algum dado pertinente sobre o estudante que estiver usando o sistema em determinado momento. Isso seria usado na criação de um plano de ensino adequado ao aluno, por exemplo.
- **Requisita resolução de Equação:** O AT pode requisitar, através de uma mensagem, a resolução, por completo ou em parte, de uma equação-problema que o aluno esteja tentando solucionar.
- **Requisita Busca de Conteúdo, baseado nos Dados do Aluno:** O AT pode requisitar um determinado conteúdo para criação de um plano de ensino, baseando-se em características do aluno que está usando o sistema.
- **Envia Conteúdo para Mostrar na Tela:** O AT, ao receber um determinado Objeto de Aprendizado que havia requisitado ao AD, pode enviar tais informações para a tela através de uma mensagem para o AI.

O Agente de Interface pode enviar as seguintes mensagens:

- **Envia dados de interação do aluno com interface:** Conforme o aluno interage dentro do sistema proposto, informações relevantes são repassadas ao Agente Tutor, para que este encaminhe ao Agente de Modelo do Aluno para a devida atualização, que reflete em maior precisão de dados no momento de criação de um plano de ensino, por exemplo.

O Agente de Modelo de Domínio pode enviar as seguintes mensagens:

- **Envia conteúdo solicitado pelo Agente Tutor:** Quando o Agente Tutor requisita um determinado conteúdo, é retornado um Objeto de Aprendizagem condizente com a requisição através desta mensagem. Tal informação é processada a agregada em um plano de ensino que o Agente Tutor esteja criando, por exemplo.

O Agente de Modelo Cognitivo (AMC) pode enviar as seguintes mensagens:

- **Envia informação de resolução da base cognitiva:** Quando o AT requisita a resolução ou validação de uma equação, o AMC envia os passos da solução/-validação através desta mensagem.

O Agente de Domínio (AD) pode enviar as seguintes mensagens:

- **Envia conteúdo solicitado pelo AT:** Quando o AT requisita um determinado conteúdo, é retornado um Objeto de Aprendizagem condizente com a requisição através desta mensagem.

O Agente de Modelo de Aluno (AMA) pode enviar as seguintes mensagens:

- **Retorna informações sobre o aluno:** Quando o AT requisita uma determinada informação sobre o aluno que está utilizando o sistema, o AMA retorna esta informação através desta mensagem. É utilizada na construção de planos de ensino por parte do AT.

Para a inferência das causas de erro durante o aprendizado de álgebra e a devida formalização em trocas de mensagens entre os agentes propostos, um estudo de ontologias foi realizado. Para a construção desta está sendo utilizada a plataforma *Protégé 4.0*, onde é possível especificar todos os componentes relevantes na comunicação entre os agentes da arquitetura, bem como fatores que levam ao erro durante resolução de exercícios algébricos e foi implementado no *framework* JADE (*Java Agent Development Framework*).

5.4. Estrutura das Mensagens

O Agente Tutor (AT), durante o processo de ensino ao estudante, pode requisitar ao Agente de Modelo Cognitivo (AMC) a resolução de uma determinada equação para verificar se um passo realizado pelo aluno está correto. Isso é realizado quando uma mensagem de sistema, contendo a *string* representando a equação, é encaminhada do AT para o AMC. É o próprio AMC que resolve as equações. Por fim, este último encaminha uma mensagem com a devida resolução. As mensagens seguirão o seguinte padrão:

•Descrição das siglas dos dados

- IA - Identificação do aluno
- IC - Identificação da ação
- EI - Equação inteira
- OP - Operadores
- OF - Operação feita

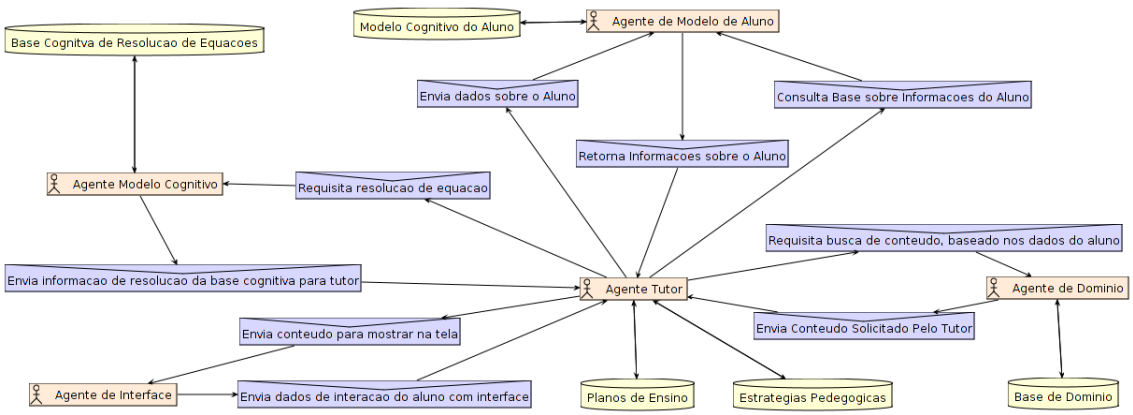

Figura 3. Arquitetura Multiagente do Pat2mathl

– RT - Resultado

•Operações possíveis:

– AD - Adição
– SB - Subtração
– DV - Divisão
– MT - Multiplicação
– MM - Mínimo Múltiplo Comum (MMC)
– DM - Distributiva em relação à multiplicação
– FC - Fator Comum (Colocar Termo em Evidência)
– QS - Produto Notável , Quadrado da Soma
– QD - Produto Notável ,Quadrado da Diferença
– PS - Produto Notável , Produto de uma soma indicada por uma diferença
– BK - Fórmula de *Bhaskara*
– OI - Operação Inversa / Oposta
– RZ - Raiz Quadrada (extrair)
– SP - Simplificação

6. IMPLEMENTAÇÃO

Os códigos apresentados nesta seção foram desenvolvidos utilizando a linguagem *java* e a plataforma JADE.

JADE (*Java Agent Development Framework*) é um *framework* implementado em sua totalidade na linguagem de programação Java, cujo objetivo é simplificar a construção de sistemas multiagente através de um *middleware* que está de acordo com as especificações da FIPA, além de um conjunto de ferramentas que permitem a depuração de código e desenvolvimento. Tal plataforma pode ser distribuída livremente entre máquinas, sem a necessidade de possuir o mesmo sistema operacional [6] .

A utilização deste *framework* se deu em decisões iniciais de projeto, e é portanto mantida até então.

Os protótipos de agentes desenvolvidos neste trabalho foram baseados em exemplos existentes no livro *Developing Multi-Agent Systems with JADE* [3].

O código 1 mostra a implementação da classe *RequestExplanationBehavior* que é referente a um comportamento do Agente Tutor. Esta classe simula o momento em que o agente solicita ao Agente Modelo Cognitivo a resposta de uma determinada equação. O método *action()* envia solicitações de resolução de equações para o Agente Modelo Cognitivo em intervalos de tempo aleatórios. Este intervalo foi utilizado para que o Agente Modelo Cognitivo pudesse carregar todas as regras necessárias para resolver as equações da base de regras.

```
1  public class RequestExplanationBehavior extends SimpleBehaviour {
2    public RequestExplanationBehavior(Agent a) {
3      super(a);
4      count = 0;
5    }
6
7    public void action() {
8      try {
9        Random r = new Random();
10       Thread.sleep(r.nextInt(10) * 1000);
11       ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
12       msg.addReceiver(new AID("CongtiveModelAgent", AID.ISLOCALNAME));
13       switch (count) {
14         case 0:
15           msg.setContent("IA:joao#IC:resolveEquacao#EI:x+3=0");
16           break;
17         case 1:
18           msg.setContent("IA:joao#IC:resolveEquacao#EI:7x+1=0");
19           break;
20         case 2:
21           msg.setContent("IA:joao#IC:resolveEquacao#EI:4x+2=0");
22           break;
23         case 3:
24           msg.setContent("IA:joao#IC:resolveEquacao#EI:3x+2x-5=0");
25           break;
26         case 4:
27           msg.setContent("IA:joao#IC:resolveEquacao#EI:x+3=0");
28           break;
29         case 5:
30           msg.setContent("IA:joao#IC:resolveEquacao#EI:8x-12=0");
31           break;
32         default: break;
33       }
34       count++;
35       System.out.println("Mensagem enviada:");
36       System.out.println(msg);
37       System.out.println();
38       myAgent.send(msg);
39     }
40     catch (Exception e) { e.printStackTrace(); }
41   }
42   private int count;
43   public  boolean done() {  return count == 5;  }
```

```
44 | }
```

Código 1. Classe RequestExplanationBehavior

O código 2 mostra a implementação da classe *SolveEquationBehavior* que é referente ao comportamento que o Agente do Modelo Cognitivo adota quando recebe um questionamento sobre uma equação. É neste momento que a equação matemática é resolvida e retornada ao Agente Tutor.

A linha 10 do código 2 mostra o momento em que o Agente Modelo Cognitivo recebe a mensagem solicitando a resolução de uma equação. Na linha vinte, o conteúdo da mensagem recebida é tratada. Na seqüência, o trecho da mensagem que é referente a equação a ser resolvida é passada como paramento ao resolvedor de equações, como pode ser visto na linha 28. Para resolver a equação, o resolvedor utiliza as regras que são carregadas no construtor da classe, linha 5. O resolvedor retorna um vetor com todos os passos necessários para resolver a equação. Este vetor é convertido para uma *string* no formato apresentado em 5.4 e remetido ao Agente Tutor, como pode ser visto na linha 38.

```
1  public class SolveEquationBehavior extends SimpleBehaviour {
2    private Resolvedor resolvedor;
3    public SolveEquationBehavior(Agent a) throws Exception {
4      super(a);
5      resolvedor = new Resolvedor();
6    }
7
8    public void action() {
9      try {
10       ACLMessage msg = myAgent.receive();
11       String[] messagePart;
12       String ia, ic, ei;
13       ia = ic = ei = "";
14       if (msg != null) {
15         this.resolvedor.clearResult();
16         System.out.println("Mensagem Recebida");
17         System.out.println(msg);
18         System.out.println();
19         messagePart = msg.getContent().split("#");
20         for (int i = 0; i < messagePart.length; i++) {
21           if(messagePart[i].split(":")[0].equals("IA"))
22             ia = messagePart[i].split(":")[1];
23           else if(messagePart[i].split(":")[0].equals("IC"))
24             ic = messagePart[i].split(":")[1];
25           else if(messagePart[i].split(":")[0].equals("EI"))
26             ei = messagePart[i].split(":")[1];
27         }
28         Vector<String> result = resolvedor.Solve(ei);
29         String fullResult = "";
30         for (String expression : result)
31           if(!expression.startsWith("#"))
32             fullResult += "OP:" + expression + "#";
33         ACLMessage reply = msg.createReply();
34         System.out.println("Mensagem de Resposta");
35         reply.setContent(fullResult);
36         System.out.println(reply);
37         System.out.println();
38         myAgent.send(reply);
39       }
40     }
41     catch (Exception e) { e.printStackTrace(); }
42   }
43 }
```

Código 2. Classe SolveEquationBehavior

O código 3 mostra a implementação do comportamento do Agente Tutor responsável por receber a solução da equação enviada pelo Agente do Modelo Cognitivo. Para este protótipo, este comportamento apenas envia para a tela do console a mensagem recebida.

```
1  public class SolutionBehavior extends SimpleBehaviour {
2    public SolutionBehavior(Agent a) {
3      super(a);
4    }
5    public void action() {
6      ACLMessage msg = myAgent.receive();
7      if (msg != null) {
8        System.out.println("Resposta");
9        String[] parts = msg.getContent().split("#");
10       for (int i = 0; i < parts.length; i++)
11         System.out.println(parts[i]);
12       System.out.println();
13     }
14   }
15   private int count;
16   public boolean done() { return count == 10; }
17 }
```

Código 3. Classe SolutionBehavior

O código 4 apresenta a implementação da classe que simula o Agente Tutor. Esta classe possui o método *setup()* que é responsável por configurar o Agente. Nesta implementação o método adiciona os dois comportamentos do Agente Tutor que estão sendo simulados, que são a requisição de resolução de equação mostrada no código 1 e o recebimento da solução, mostrada no código 3. A linha três pára a execução do processo por dez mil milissegundos para que o Agente do Modelo Cognitivo consiga carregar as regras para resolver as equações da sua base de regras.

```
1  public class TutorAgent extends Agent {
2    protected void setup() {
3      try {
4        Thread.sleep(10000);
5      } catch (InterruptedException e) {
6        e.printStackTrace();
7      }
8      super.addBehaviour(new RequestExplanationBehavior(this));
9      super.addBehaviour(new SolutionBehavior(this));
10   }
11 }
```

Código 4. Classe Tutor Agent

O código 5 mostra a implementação da classe que simula o Agente do Modelo Cognitivo. Assim como a implementação do Agente Tutor, esta classe possui um método *setup()* que adiciona ao agente a comportamento que recebe uma requisição para solucionar uma equação, resolve esta e remete ao Agente Tutor a solução. A implementação deste comportamento é apresentada no código 2.

```
1  public class CongtiveModelAgent extends Agent{
2    protected void setup() {
3      super.addBehaviour(new SolveEquationBehavior(this));
4    }
5  }
```

Código 5. Classe CongtiveModelAgent

7. TRABALHOS RELACIONADOS

Algebrain é um Sistema Tutor Inteligente cujo objetivo é resolver uma equação para uma determinada variável. O sistema não se preocupa em mostrar uma única resposta final, ao invés disso mostra o processo todo da resolução. Integrado ao sistema existe um sistema especialista baseado em regras que simula os passos de um expert da área de resolução de equações, o que ele faria em cada caso. Este modelo cognitivo de um estudante

ideal, que nunca erra, se encontra no servidor da aplicação web resolvendo equações junto com o usuário. Este está em contato freqüente com o Módulo Tutor que proporciona dicas de resolução assim como determina o grau de eficácia nos passos que o aluno faz para resolver uma equação e dar o *feedback* adequado a cada ocasião [1].

O trabalho proposto difere de outros STI Web, tal como *Algebrain* [1] por buscar uma melhoria na usabilidade da interface, enriquecendo a experiência do usuário através de páginas dinâmicas e execução de código em lado cliente. Além disso, PAT2Math será o primeiro STI capaz de considerar as emoções do usuário através de dispositivos próprios, capacitando o tutor a agir ativamente ao estado do aluno, incentivando-o através de fala, gestos, expressões faciais e corporais, facilitando a comunicação tutor-aluno. Além disso, embora o *Algebrain* também utilize um sistema especialista para auxiliar alunos na resolução de equações, ele não é capaz de detectar as concepções errôneas dos mesmos.

O *ActiveMath* é um sistema tutor inteligente cuja a abordagem pedagógica do sistema consiste na livre interação do aluno com as lições propostas, construindo o seu aprendizado. Visa o ensino individual, onde se pode optar por quais tarefas executar, bem como em qual ordem. O *ActiveMath* baseia-se em conteúdos que se adaptam ao aluno e suas necessidades.

A arquitetura do *ActiveMath* tem como princípio básico a separação de conteúdos e funcionalidades, bem como a devida diferenciação entre tipos de conhecimento. As regras pedagógicas de ensino são armazenadas na Base Pedagógica de Regras, não interferindo nas informações de aluno, armazenadas no Modelo de Estudante, por exemplo. Tal princípio facilita questões de reusabilidade e manutenção do modelo, além de estruturá-lo como um todo para a *web*. O conhecimento é representado na forma semântica em linguagem XML, que na realidade é uma extensão do *OpenMath*. Conhecimentos são tratados como objetos de aprendizado, não uma simples sintaxe como são abordados em geral. Isto traz alguns benefícios, tais como serem mais compreensivos e poderem ser interligados com outros saberes presentes no sistema [9].

Em relação a outros tutores que seguem os princípios gerais de arquiteturas de STI [2], existem vantagens que vão além da questão de um sistema Web capaz de considerar emoções. A abordagem utilizando um Arquitetura Multiagente é considerada uma técnica que contribui muito para a redução de complexidade de sistema, através da criação de componentes modulares que resolvem subtarefas que constituem um objetivo maior. Cada agente utiliza a sua técnica mais eficiente para resolver tais subtarefas, não seguindo a abordagem geral, mais genérica para a solução [19].

O presente projeto se compara aos demais da área pelo caráter educacional envolvido, bem como didática diferenciada na parte de tutoria, ensino de conteúdo. O PAT2Math visa traçar o aprendizado do aluno com técnicas de *Knowledge Tracing* [2] com o objetivo de saber como remediar um aprendizado errado por parte do aluno, saber inferir em que parte do processo cognitivo do aluno ocorreu um erro.

Um elemento que destaca o projeto dos demais é a consideração de fatores emocionais durante o ensino. Tal característica é proveniente do estudo de uma arquitetura de emoções a ser incluída no Módulo Tutor do Sistema, conforme descrito em [7] e da inferência de emoções por face, conforme descrito em [13].

8. CONCLUSÃO

A arquitetura proposta visa interação e relação entre múltiplos agentes, cada qual com um conjunto de papéis a serem representados no sistema. A execução de tais papéis torna o ensino-aprendizagem do aluno no proposto ambiente Web possível, se apoiando juntamente na metodologia *Prometheus*, que possui um forte referencial teórico e sistemas baseados na mesma. Além disso, a concepção do STI segundo uma abordagem multiagente diminui a complexidade no desenvolvimento do sistema e melhora a compreensão do projeto entre os membros da equipe devido a modularização dos componentes deste projeto.

Atualmente o projeto se encontra em fase intermediária de desenvolvimento. A arquitetura e o contexto do sistema já estão definidos, assim como os papéis dos agentes propostos [10]. Existe um grupo de pesquisa, composto por estudantes de graduação e mestrado na área das Ciências Exatas, trabalhando ativamente neste projeto. O esforço deste grupo resultou até o momento na especificação completa do sistema e no desenvolvimento dos seguintes agentes: AMC - já descrito em [16], AD, AI (parcial) e AT (parcial), bem como a documentação especialista em álgebra, que está servindo como base para as ontologias que serão embarcadas no AT. Adicionalmente, será integrado o recurso de reconhecimento de emoções através da face, baseado no trabalho de pesquisa de [13], assim como através de um modelo psicológico-cognitivo descrito em [7].

Juntamente no sistema será integrado o agente pedagógico animado e afetivo, cujo o referencial teórico encontra-se em [8].

Referências

[1]Sherman Alpert, Mark Singley, and Graeme Pye Fairweather. Deploying intelligent tutors on the web: An architecture and an example. *In IJAIED, 10(2)*, pages 183–197, 1999.

[2]John R. Anderson. General principles for an intelligent tutoring architecture. *Cognitive Approaches to Automated Instruction*, 1992.

[3]Fabio Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. wiley, 2007.

[4]Cui Guangzuo. Ontoedu:ontology-based eucation grid system for e-learning. *FIFTH AGRICULTURAL ONTOLOGY SERVICE (AOS) WORKSHOP*, 2004.

[5]Elie Jacob. Ontologies and the semantic web. *Bulletin of the American Society for Information Science and Technology*, 29:19, 22, 2003.

[6]JADE. Introduction to JADE. http://jade.tilab.com/, acessado em agosto de 2009, Setembro 2009.

[7]Patrícia Augustin Jaques. Avaliando um modelo afetivo de aluno baseado em uma abordagem cognitiva. *SBIE, 2008, Fortaleza - CDROM.*, 2008.

[8]Patrícia Augustin Jaques, Matheus Lehmann, and Sylvie Pesty. Evaluating the affective tactics of an emotional pedagogical agent. In *ACM-SAC '09*, pages 104–109, NY, USA, 2009. ACM.

[9]Erica Melis and Jörg Siekmann. Activemath: An intelligent tutor system for mathematics. *Seventh International Conference Artificial Intelligence and Soft Computing*, 2004.

[10]Gabriel Mello, Talvany Carlotto, Geiseane Rubi, Henrique M. Seffrin, and Patrícia A. Jaques. Implementando o agente de base de domínio so sistema tutor inteligente pat2math. *XIII Ciclo de Palestras sobre Novas Tecnologias na Educação, 2009, Porto Alegre.*, 2009.

[11]Jean-Peirre Müller and Yves Demazeu. Descentralized artificial intelligence. *European Workshop on Modelling Autonomous Agents in a Multi-Agent World, 1, 1989, Cambridge. North-Holland: Elselvier Science Publishers*, 1990.

[12]Tom Murray. Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, 1999.

[13]Eduardo Oliveira and Patrícia Augustin Jaques. Inferindo as emoções do usuário pela face através de um sistema psicológico de codificação facial. *IHC, 2008, POA : SBC/ACM*, 2008.

[14]James Ong and Sowmya Ramachandram. Intelligent tutoring systems: Using ai to improve training performance and roi. *Stottle Henke Publications*, 2003.

[15]Lin Padgham and Michael Winikoff. The prometheus metodology. April 2004.

[16]Henrique M. Seffrin, Geiseane Rubi, Talvany Carlotto, Gabriel Mello, and Patrícia A. Jaques. Um resolvedor de equações algébricas como ferramenta de apoio à sala de aula no ensino de equações algébricas. *WIE - CSBC, 2009, Bento Gonçalves.*, pages 1791–1800, 2009.

[17]Yoav Shoam. An overview of agent-oriented programming. *Software Agents. Menlo Park: AAAI Press/The IT Press*, pages 271–290, 1997.

[18]Jaime Simão Sichman, Yves Demazeu, and Olivier Boissier. How can knowledge-based systems be called agents? *SBIA, 9, 1992, Rio de Janeiro. Anais... RJ: PUC-RJ*, pages 173–185, 1992.

[19]Marina V. Sokolova and Antonio Fernández-Caballero. Modeling and implementing an agent-based environmental health impact decision support system. *Expert Syst. Appl.*, 36(2):2603–2614, 2009.

[20]Carsten Ullrich. C. ullrich. *Proceedings of Workshop on Applications of Semantic Web Technologies for E-learning, SW-EL'04*, pages 17,23, 2004.