LUCIANA BOLAN FRIGO

Um modelo de Autoria para Sistemas Tutores Adaptativos

FLORIANÓPOLIS
2007

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Um modelo de Autoria para Sistemas Tutores Adaptativos

Tese submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Doutor em Engenharia Elétrica.

LUCIANA BOLAN FRIGO

Florianópolis, janeiro de 2007.

UM MODELO DE AUTORIA PARA SISTEMAS TUTORES ADAPTATIVOS

Luciana Bolan Frigo

Esta Tese foi julgada adequada para a obtenção do título de Doutor em Engenharia Elétrica, Área de Concentração em *Sistemas de Informação*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.

Prof. Guilherme Bittencourt, Dr.
Orientador

Prof. Nelson Sadowski, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora :

Prof. Guilherme BITTENCOURT, Dr.
Presidente

Prof. Janette CARDOSO , Dr.

Prof. Bernard ESPINASSE, Dr.

Prof. Evandro de Barros COSTA , Dr.

Prof. Clara Amélia de OLIVEIRA, Dr.

Prof. Roberto WILLRICH, Dr.

Ao Cleicio

AGRADECIMENTOS

Meus agradecimentos aos membros do júri, em especial aos relatores Bernard Espinasse e Evandro de Barros Costa, que fizeram a avaliação desta tese com relevantes sugestões.

Aos meus orientadores, Guilherme Bittencourt e Christophe Sibertin-Blanc pela oportunidade de crescimento pessoal e profissional, que me acolheram em suas equipes.

À minha co-orientadora Janette Cardoso que me apoiou, principalmente quando de minha chegada à Toulouse, acolhendo-me e apoiando-me em tudo que precisei.

Ao CNPq pela bolsa de doutorado aqui no Brasil, fazendo-me acreditar que o País está investindo em educação e tecnologia.

À Capes - COFECUB pela oportunidade de crescimento profissional e também pessoal proporcionado pela vivência em um outro país.

À Universidade Federal de Santa Catarina e ao Departamento de Automação e Sistemas (DAS), onde passei a maior parte dos dias nestes últimos 12 anos e o qual devo todo conhecimento adquirido neste período.

A toda equipe da Universidade Toulouse 1, na França, que me recebeu muito bem.

Aos participantes do projeto MathNet que colaboraram para este trabalho, em especial ao Emílio Eyamane com todo o seu conhecimento de programação.

Ao meu marido, Cleicio, um agradecimento especial, por ter estado ao meu lado, apesar da distância durante parte do meu trabalho realizado na França, por todo o seu apoio e sacrifício.

Aos meus pais, Luiz e Cirlene, por toda dedicação e esforço para que tudo isto se tornasse possível.

Ao meu irmão, Tiago e à todos os familiares que vibram com minhas conquistas.

A todos os amigos e colegas que direta ou indiretamente ajudaram na realização deste projeto, em especial à Tatiana Garcia, Rafael Obelheiro, Magnus Martinello, Roberta Gomes, José Valentin, Alessandra Cristina, Ana Lúcia Franco, Cássia Yuri Tatibana, Pascaline Tchienehom, Matthias Maillard e Omar Tahir que compartilharam os dias mais difíceis e também os mais alegres na minha estada na França.

Resumo da Tese apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Doutor em Engenharia Elétrica.

Um Modelo de Autoria para Sistemas Tutores Adaptativos

Luciana Bolan Frigo

Janeiro/2007

Orientadores : Prof. Guilherme Bittencourt, Dr. e Prof. Christophe Sibertin-Blanc, Dr.
Co-orientadora : Profa. Janette Cardoso, Dra.
Área de Concentração : Sistemas de Informação Industrial
Palavras-chave : sistemas tutores inteligentes, adaptação, ferramenta de autoria
Número de Páginas : xii + 126

Um dos grandes desafios dos pesquisadores da área de Sistemas Tutores Inteligentes (STI) tem sido definir maneiras de construir STI adaptativos sem que isso acarrete a elevação dos custos de tempo e esforço do professor. A principal contribuição desta tese é a proposta de um modelo formal de suporte à adaptação para STI, implementado na Ferramenta de Autoria para Sistemas Tutores (FAST). A definição do modelo é baseada em formalismos como Ontologias, para a representação do conhecimento envolvido nos modelos do domínio e do estudante, e Redes de Petri a Objetos, para definir o modelo pedagógico. As transições destas Redes de Petri controlam as interações com o estudante, escolhendo os conteúdos do domínio a serem apresentados a cada momento de acordo com as informações armazenadas no modelo do estudante. Todo o gerenciamento e controle do modelo do estudante, bem como a determinação dos conteúdos mais apropriados para um determinado estudante, são feitos automaticamente pelo modelo pedagógico. O professor especifica somente o conteúdo referente ao domínio do curso, estabelecendo pré-requisitos e níveis de dificuldade de acordo com a estrutura do modelo de autoria.

Abstract of Thesis presented to UFSC as a partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering.

An Authoring Model for Adaptive Intelligent Tutoring System

Luciana Bolan Frigo

January/2007

Advisors: Prof. Guilherme Bittencourt, Dr. and Prof. Christophe Sibertin-Blanc, Dr.
Co-advisor: Profa. Janette Cardoso, Dra.

Area of Concentration: Information Systems
Key words: Intelligent Tutoring Systems, adaptive, authoring tool
Number of Pages: xii + 126

This thesis proposes a method for building ITSs that uses the information about the student interactions to adapt itself to the student. The method allows the teacher to specify the course contents without burdening on how the adaptive mechanism work. It also allows an automatic integration of the contents provided by the teacher with the adaptive mechanism that takes into account student profiles and their performances in the course. This method is implemented in a tool called FAST (Ferramenta de Autoria para Sistemas Tutores in Portuguese). FAST generates the description of a specific ITS using Objects Petri Nets formalism. The Petri Net formalism, besides being a graphic tool, allows to verify some systems properties like deadlocks in a course sequence. Nevertheless, the use of this formalism is transparent for both, the teacher and the student. The teacher has only to specify the domain contents establishing prerequisite and difficulty level. The course description provided by FAST is executed by an ITS shell, named ASTI. FAST and ASTI are multi-agent systems that use ontologies to represent student and domain knowledge.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Sempre há o que aprender, ouvindo, vivendo e sobretudo, trabalhando, mas só aprende quem se dispõe a rever as suas certezas.
Darcy Ribeiro

A Inteligência Artificial na Educação (IA-ED) [Kearsley, 1987; Wenger, 1987] aplica técnicas de Inteligência Artificial (IA) [Klassner, 1996; Russell e Norvig, 1995; Pereira, 2002; Bittencourt, 1998b] a programas computacionais educacionais, com o objetivo de construir modelos e arquiteturas que gerenciem, com algum grau de autonomia, as interações de um tutor artificial com os estudantes na busca de um aprendizado mais eficaz. A presente tese aborda um dos ramos da pesquisa em Inteligência Artificial aplicada à Educação, conhecido como Sistemas Tutores Inteligentes (STI's) [Self, 1990; Knezek, 1988; Nakabayashi et al., 1997; Vicari e Giraffa, 2003].

Esta tese faz parte de um projeto de desenvolvimento de um arcabouço para sistemas tutores inteligentes distribuídos que utilizam técnicas de Inteligência Artificial Distribuída (IAD) [Demazeau e Müller, 1989; Gasser, 1992], seguindo a abordagem de Sistemas Multiagentes (SMA) [Sichman et al., 1992; Franklin e Graesse, 1996; Jennings et al., 1999; Ferber, 1999]. Este arcabouço é baseado no MATHEMA [Costa, 1997] que é um modelo para concepção e desenvolvimento de ambientes de aprendizagem assistidos por computador [Park, 1998; Self, 1988].

Neste capítulo introdutório faz-se um breve relato sobre a motivação de se investir na referida área, uma discussão sobre o contexto, os objetivos, as contribuições e por fim, apresenta-se a organização do documento em questão.

1.1 Motivação

Os esforços destinados a conceber e desenvolver sistemas para educação e treinamento fazem parte da área denominada, entre a comunidade acadêmica brasileira, de Informática na Educação (IE) [Vicari e Giraffa, 2003]. O termo Informática na Educação possui diversos significados dependendo da visão educacional e pedagógica em que o computador é utilizado. Definiu-se, neste projeto de pesquisa, a Informática na Educação como sendo o uso do computador no processo de aprendizagem dos conteúdos curriculares.

Existem duas maneiras de se utilizar o computador na educação. Uma seria através da extensão do processo de ensino, onde o computador é utilizado apenas para transmitir a informação para o aluno. A outra, cria condições para que o aluno construa seu próprio conhecimento através de ambientes de aprendizagem que façam uso do computador.

A primeira abordagem é mais simples de ser implementada e executada, sendo aplicada no ensino tradicional, onde o computador é utilizado apenas para informatizar os processos de ensino existentes. Esta abordagem, apesar de ser a mais utilizada, é bastante questionável no mundo moderno, onde o mercado de trabalho busca profissionais dinâmicos, criativos e capazes de construírem seu próprio conhecimento.

Já a segunda abordagem, o uso do computador na criação de ambientes de aprendizagem que enfatizam a construção do conhecimento, apresenta enormes desafios, redimensionando conceitos, idéias e valores já conhecidos. O processo de formação deve oferecer condições para o professor/aluno construir conhecimento sobre técnicas computacionais e entender porque e como integrar o computador na sua prática pedagógica.

Uma das ramificações da Informática na Educação é a modalidade denominada de Educação à Distância (EAD) mediada por computador [Sherry, 1996; ABED; MEC], onde as atividades de ensino-aprendizagem são desenvolvidas majoritariamente sem que professores e alunos estejam presentes espacial e/ou temporalmente. A superação da distância (física) se dá através da utilização de tecnologias de informação e comunicação que podem ser usadas isoladamente ou podem ser combinadas. As tecnologias mais comumente utilizadas são: televisão, correspondência, vídeos, áudios, computadores e, mais recentemente, a Internet.

Dentre as modalidades de Ensino à Distância destacam-se os Ambientes de Aprendizagem Assistidos por Computador, que fazem uso das redes de computadores. A união dos Ambientes de Aprendizagem Assistidos por Computador com algumas técnicas de Inteligência Artificial (IA) tem apresentado resultados encorajadores. Dentre estas técnicas a Inteligência Artificial Distribuída (IAD), que estuda o conhecimento e as técnicas de raciocínio que podem ser necessárias ou úteis para que agentes computacionais participem de sociedades de agentes, originou a área de pesquisa denominada Inteligência Artificial na Educação (IA-ED) (ver figura 1.1) na qual encontram-se os Sistemas Tutores Inteligentes (STIs).

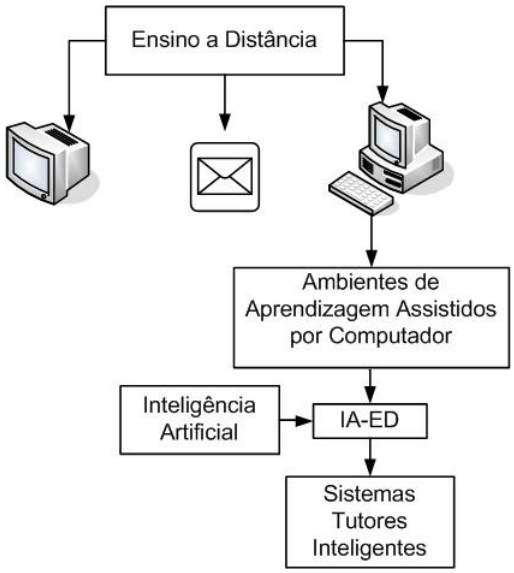


Figura 1.1: EAD e STI

Apesar do Ensino à Distância não fazer parte diretamente do escopo deste projeto, muitas das técnicas desenvolvidas para os STIs podem ser aplicadas em ambientes computacionais para o Ensino à Distância.

Os STIs raramente são usados em ambientes educacionais reais, enquanto que o uso dos ambientes para ensino à distância estão sendo difundidos a uma velocidade crescente. A primeira razão disto é devido ao fato dos STIs serem tipicamente difíceis de serem utilizados sob a ótica do professor. Estima-se que o tempo de desenvolvimento de uma hora de instrução exija do autor algo entre 200 a 300 horas de autoria [Aleven et al., 2006]. Uma maneira de expandir o uso dos STIs é pela redução deste tempo, através das chamadas ferramentas de autoria.

1.2 Contexto e Definição do Problema

Um Sistema Tutor Inteligente (STI) envolve três entidades principais: o aprendiz[1], o professor e a máquina. Do ponto de vista da implementação computacional encontram-se diversas configurações de acordo com o enfoque de cada sistema, contudo tem-se sempre as três entidades principais.

Construir um Sistema Tutor Inteligente é sobretudo um esforço em adequar as dinâmicas de aprendizagem do curso presencial para um sistema computacional, criando um ambiente motivador, onde o estudante sinta que a distância entre o tutor e o estudante é apenas física e não pessoal.

[1]Os termos aprendiz, aluno e estudante são usados como sinônimos neste documento.

Os STIs são ambientes computacionais que utilizam essencialmente informações das interações entre o sistema e o estudante para fornecer um sistema adaptado as necessidades do estudante.

Fazer com que o comportamento de um STI se assemelhe ao de um tutor real é um desafio que se pretende superar através do uso de técnicas computacionais. Entretanto, é possível relacionar elementos limitantes no desenvolvimento de STIs como: a falta de teorias psicológicas e pedagógicas adequadas, além da falta de recursos capazes de reconhecer e diagnosticar o nível de conhecimento e o grau de entendimento de um estudante através da visualização das expressões faciais e corporais. Durante o projeto de um programa educacional, de acordo com o tipo de modalidade escolhida, faz-se necessário tomar uma série de decisões com o intuito de superar algumas limitações no período de desenvolvimento.

Os Sistemas Tutores Inteligentes direcionam as pesquisas de sistemas computacionais educacionais que objetivam proporcionar um ambiente adaptado ao aluno. A questão da adaptação ou da personalização tem sido objeto de estudo de muitos grupos de pesquisa [Brusilovsky e Peylo, 2003; Palazzo, 2002; Jonassen e Wang, 1993; D'Amico et al., 1998; Cristea, 2004; Bica et al., 1998; Bra et al., 2003; Lin et al., 2004; Aleven et al., 2006] e até o momento este objetivo não foi alcançado por completo e continua a ser o gargalo para a evolução deste tipo de sistema educacional.

O problema da adaptação é bastante complexo e trata basicamente das interações que envolvem a navegação e a apresentação de um domínio do conhecimento de acordo com as estratégias e táticas pedagógicas determinadas pelo professor e que sejam ao mesmo tempo adequadas ao estilo de aprendizado do estudante.

O termo adaptação pode ser empregado em diferentes situações, para evitar interpretações equivocadas se faz necessário explicitar o contexto usado nesta tese. Enfoca-se a adaptação como o modo de interação entre o ambiente de aprendizagem computacional e o estudante, ou seja, deseja-se obter um Sistema Tutor Inteligente com interações adaptativas. De acordo com as respostas do estudante o sistema traça dinamicamente um ou mais percursos de navegação respeitando as necessidades individuais do estudante.

1.3 Objetivos

O desafio proposto é o de contribuir efetivamente para a área de Sistemas Tutores Inteligentes (STIs) enfocando a modelagem das interações entre o sistema tutor e o estudante, de modo a facilitar a construção de cursos através destes sistemas.

Geral Propor um modelo formal para o desenvolvimento de STI que, por um lado, permita a geração de sistemas tutores inteligentes que permita uma escolha pelo tutor do melhor per-

curso para o estudante e, por outro lado, defina uma metodologia para que o professor especifique o conteúdo do curso sem envolver-se com os mecanismos de adaptação tutor/estudante.

Específicos

1. Especificação de uma metodologia de criação de cursos que permita a integração automática do conteúdo fornecido pelo professor com o mecanismo de adaptação do tutor ao estudante.
2. Especificação de um modelo do estudante que armazene tanto suas características individuais como os resultados de seu desempenho durante a interação com o sistema.
3. Definição de um mecanismo que permita a especificação de protocolos complexos de interação, cujo controle leve em conta as características individuais e o desempenho dos estudantes.
4. Desenvolvimento de uma ferramenta de autoria para STI que implemente a metodologia proposta em 1, para gerar STI que integrem o modelo do estudante proposto em 2 com o mecanismo de controle proposto em 3.

1.4 Contribuições

A principal contribuição desta tese é a proposta de um modelo formal que forneça aos STI um mecanismo de adaptação do ambiente computacional ao estudante de forma a diminuir o número de tarefas a serem executadas pelo professor na construção de um curso, tal mecanismo é implementado na Ferramenta de Autoria para Sistemas Tutores (FAST). A separação do planejamento instrucional (inseridas pelo professor) das estratégias de ensino-aprendizagem (implementadas pelo desenvolvedor), é também uma contribuição pois, a FAST permite gerar um STI onde o professor especifica somente o conteúdo referente ao domínio do curso, estabelecendo pré-requisitos e níveis de dificuldade de acordo com a estrutura do modelo de autoria. Outra vantagem deste modelo é a sua capacidade de se adicionar novas informações sobre o estudante na medida em que o modelo do estudante evolui.

1.5 Organização

Esta tese está estruturada em 6 capítulos e 5 apêndices, sendo este o primeiro capítulo e os demais são organizados conforme a descrição a seguir.

Os Sistemas Tutores Inteligentes englobam uma área altamente interdisciplinar, fazendo-se necessário abordar conceitos e tecnologias relacionadas com as diferentes áreas do conhecimento, apresentadas no Capítulo 2. O estado da arte sobre os arcabouços e ferramentas de autoria estão descritos neste capítulo.

O Capítulo 3 apresenta o projeto que deu origem a diversos trabalhos na busca de se obter um arcabouço para sistemas tutores inteligentes no qual esta tese deixa sua contribuição. Este capítulo traz as principais características do modelo conceitual para concepção e desenvolvimento de ambientes de aprendizagem assistidos por computador que fornece o embasamento teórico para este projeto de pesquisa. Para verificar a evolução dos trabalhos realizados um histórico de desenvolvimento do arcabouço para Sistemas Tutores Inteligentes é mostrado.

O Capítulo 4 é o capítulo central e apresenta a descrição dos elementos e do mecanismo de integração do modelo de autoria proposto, que deu origem a Ferramenta de Autoria para Sistemas Tutores (FAST), juntamente com alguns aspectos de implementação.

O Capítulo 5 aborda as características requeridas no Arcabouço para Sistemas Tutores Inteligentes (ASTI) em desenvolvimento e apresenta dois estudos de caso de forma a ressaltar os aspectos de modelagem a serem realizados pelo professor durante a criação de um curso utilizando a FAST e a ASTI.

O Capítulo 6 aborda as principais conclusões desta tese, juntamente com as perspectivas para futuras investigações.

O Apêndice A descreve as ferramentas que deram suporte para a implementação da FAST.

O Apêndice B mostra noções sobre o funcionamento das Redes de Petri e das Redes de Petri a Objetos.

O Apêndice C apresenta os principais passos para a construção do compilador que transforma as redes de Petri em regras JESS.

O Apêndice D mostra o código JESS usados para inferir o que apresentar ao estudante.

O Apêndice E indica o histórico da área de Informática na Educação no Brasil.

Capítulo 2

Revisão da Literatura

> *Todo o programa pode ser considerado um programa educacional, desde que utilize uma metodologia que o contextualize no processo de ensino-aprendizagem*
> Rosa Vicari e Lucia Giraffa [Vicari e Giraffa, 2003].

A área dos Sistemas Tutores Inteligentes, por tratar da interação homem-máquina busca, através de técnicas e teorias, atenuar a separação física entre professor e aluno.

Neste capítulo é apresentada uma visão de como o ensino por computador está inserido na Inteligência Artificial além dos recursos tecnológicos que servem de base para a compreensão deste documento, sem a pretensão de esgotar os assuntos aqui abordados.

2.1 Inteligência Artificial

Em 1950, Allan Turing [Turing, 1950] sustentava: "é inteligente uma máquina que é capaz de iludir e passar por inteligente aos olhos dos homens". Ele propôs a construção de máquinas "inteligentes", que fossem capazes de imitar comportamentos humanos. As máquinas propostas por Turing são predecessoras dos sistemas especialistas atuais, que também procuram ser cópias tão perfeitas quanto possível dos processos cognitivos humanos.

A Inteligência Artificial é, por um lado, uma ciência, que procura estudar e compreender o fenômeno da inteligência, e, por outro, uma área da engenharia que procura construir instrumentos para apoiar a inteligência humana. Um exemplo clássico de Inteligência Artificial é o jogo de xadrez virtual, onde a máquina tentaria antecipar o movimento do adversário, para obter uma vantagem que lhe permitisse vencer a partida.

A partir de meados do século XX, o desenvolvimento da Inteligência Artificial esteve profundamente ligado à evolução dos computadores. Através deles, tornou-se possível simular vários aspectos da inteligência humana, levando o homem a questionar se as máquinas

poderiam ser programadas de maneira a tornarem-se inteligentes (como os seres humanos) e capazes de aprender. Algumas máquinas desempenham tarefas que, quando realizadas pelo homem, implicam no uso da inteligência. Existe uma distância entre a inteligência humana (considerada como o referencial de inteligência) e a *inteligência* da máquina. De uma maneira simplificada, pode-se dizer que a *inteligência* da máquina não possui todas as *habilidades* da inteligência do homem [Abrantes, 1993].

A Inteligência Artificial nasceu oficialmente em 1956 com um seminário em Dartmouth, nos Estados Unidos, onde os primeiros líderes da IA se encontraram. Entre eles estavam John McCarthy, Marvin Minsky, Allen Newell e Herbert Simon [Klassner, 1996]. No entanto, o objeto de estudo da IA continua nebuloso, pois ainda não se tem uma definição suficientemente satisfatória de inteligência e, para se compreenderem os processos da inteligência artificial e da representação do conhecimento é necessário dominar os conceitos de inteligência humana e conhecimento.

Os estudos em IA dividem-se em quatro ramos fundamentais [Vignaux, 1995]:

i. o ramo clássico da Inteligência Artificial que tenta representar na máquina os mecanismos de raciocínio, ligando-se desde o início à Psicologia, desde os anos 70 à Epistemologia, e desde os anos 80 à Sociologia;

ii. o ramo ligado ao estudo das redes neurais e ao conexionismo, que se relaciona com a capacidade dos computadores de aprenderem e reconhecerem padrões;

iii. o ramo ligado à biologia molecular, na tentativa de construir vida artificial;

iv. o ramo relacionado com a robótica, procurando construir máquinas que alojem vida artificial.

2.1.1 Inteligência Artificial Distribuída

A *Inteligência Artificial Distribuída (IAD)* é uma das áreas da *Inteligência Artificial (IA)* que mais se desenvolveram nos últimos anos e apresenta um enorme potencial de aplicações, relaciona-se com a solução cooperativa de problemas dentro de um certo ambiente por intermédio de agentes distribuídos. A IAD estuda o conhecimento e as técnicas de raciocínio que podem ser necessárias ou úteis para que agentes computacionais participem de sociedades de agentes.

A aplicação da IAD baseia-se em idéias de que a agilidade, flexibilidade, inteligência e desempenho de um sistema podem ser sensivelmente melhorados a medida que ela permite alcançar os seguintes objetivos [Parunak, 1995]:

- Construção de sistemas descentralizados ao invés de centralizados;

- Soluções emergentes (resultado da interações entre agentes e/ou humanos) ao invés de totalmente planejadas;
- Execução concorrente ao invés de seqüencial.

Muitos problemas reais são naturalmente e fisicamente distribuídos, além de serem funcionalmente distribuídos e heterogêneos e a difusão das redes de computadores são alguns dos motivos estimuladores da distribuição de Sistemas Inteligentes [Ferber, 1999].

A IAD se dividiu basicamente em duas áreas, sendo que ambas trabalham com o conceito de agentes:

- Solução Distribuída de Problemas (SDP), e
- Sistemas Multiagentes (SMA).

A Solução Distribuída de Problemas possui como foco principal o problema, sendo que este enfoque é derivado da IA simbólica. O principal objetivo da SDP é a utilização da capacidade de processamento e da robustez oferecidas pela tecnologia de redes para atacar problemas de natureza distribuída ou muito complexos [Bittencourt, 1998b], como, por exemplo, o problema de controle de tráfego aéreo. Os agentes envolvidos na Solução Distribuída de Problemas são programados para cooperar, dividir tarefas e comunicar-se de maneira confiável, entretanto não é simples estabelecer estas propriedades.

Os Sistemas Multiagentes enfocam o estudo das pressuposições básicas sobre agentes que garantam a possibilidade de ação cooperativa em sociedade [Bittencourt, 1998b].

Tanto no SMA como na SDP os agentes compartilham um mesmo ambiente. A diferença está mais na forma de interação entre os agentes. Nos modelos SDP, os agentes são projetados de maneira que haja o máximo de cooperação. Nos modelos SMA, os agentes competem por recursos e precisam muitas vezes resolver conflitos [Sherer e Schlageter, 1995].

2.2 Sistemas Multiagentes

Do ponto de vista da Inteligência Artificial Distribuída, um Sistema Multiagente é uma rede acoplada de entidades de resolução de problemas que trabalham juntas para achar respostas aos problemas que estão além das capacidades individuais ou do conhecimento de cada entidade [Durfee e Rosenschein, 1994].

Para a formação de uma sociedade de agentes visando solucionar um problema torna-se necessária a consideração de alguns conceitos como:

- Organização: um grupo de agentes, objetivando resolver um determinado problema, onde a idéia é realizar uma decomposição em subproblemas, repartidos entre os agentes envolvidos.
- Controle: mecanismo que controlam o sistema podendo se centralizado ou distribuído.
- Cooperação: mecanismo que viabiliza as atividades cooperativas entre agentes.
- Comunicação: mecanismo responsável pela comunicação entre agentes, normalmente baseados na troca de mensagens ou compartilhamento de informações.

Sistemas Multiagentes é a parte da IAD onde o foco principal das pesquisas são os agentes, as características básicas deste tipo de sistema são [Ferber, 1999; Jennings, 1994]:

- A comunidade é concebida para cooperar em um ambiente aberto, onde cada agente tem autonomia e pode participar da resolução de problemas;
- Eventualmente, um agente pode resolver sozinho um problema;
- Os agentes competem entre si pelos recursos, precisando saber lidar com conflitos e coordenar atividades para aumentar a eficiência na solução de problemas;
- Os agentes não precisam utilizar a mesma linguagem, implicando a necessidade de traduções e mapeamentos para as representações individuais;
- Os agentes possuem apenas uma visão parcial do ambiente, não possuindo informações completas;
- O controle do sistema é descentralizado, as informações distribuídas e o processamento é assíncrono.

Para Ferber [1999], o termo SMA é aplicado a sistema que compreende os seguintes elementos:

- Um ambiente E, representado por um espaço que possui um volume;
- Um conjunto de entidades O. É possível, em um dado momento, associar qualquer entidade com uma posição E. Essas entidades são passivas, ou seja, elas podem ser percebidas, criadas, destruídas e modificadas pelos agentes;
- Um conjunto de agentes A, de entidades específicas representando as entidades ativas do sistema;
- Um conjunto de relações, R, que liga objetos (e portanto agentes) entre si;
- Um conjunto de operações, Op, que permite aos agentes de A perceberem, produzirem, consumirem, transformarem e manipularem entidades de O;

- Operações com a tarefa de representar a aplicação dessas operações e a reação do mundo para suas tentativas de modificação.

2.2.1 Definições de Agentes

Um agente pode ser definido em termos de suas propriedades fundamentais e deve possuir um certo grau de autonomia para raciocinar e tomar decisões por sua própria vontade. Além disso, deve ter a capacidade de interagir com outros agentes e apresentar um certo grau de independência para resolver um problema ou parte dele.

Não existe uma definição única para os agentes por isso, os autores normalmente ligam a definição ao domínio da aplicação, às formas de cooperação e dos seus níveis de autonomia. Diversas definições podem ser encontradas na literatura:

> *Chama-se um agente uma entidade real ou abstrata que é capaz de agir sobre ela mesma e sobre seu ambiente, que dispõe de uma representação parcial deste ambiente, que, em um universo multiagente, pode comunicar-se com outros agentes, e cujo comportamento é conseqüência de suas observações, de seu conhecimento e das interações com outros agentes.* Ferber e Gasser [1991]

> *Um agente é uma entidade à qual se pode associar uma identidade única, e que é capaz de realizar cálculos formais. Um agente pode ser considerado como um meio que produz um certo número de ações a partir dos conhecimentos e mecanismos internos que lhe são próprios.* Gasser [1992]

> *Agentes são programas que estão envolvidos em atividades de diálogo e que podem negociar e coordenar transferências de informação.* Franklin e Graesse [1996]

A definição de N. Jennings [Jennings, 1994] foi adotada neste projeto de pesquisa:

> *Um agente é um sistema computacional, posicionado em algum ambiente, que é capaz de agir com autonomia flexível visando atingir os objetivos para o qual foi projetado.*

Existem algumas propriedades que devem ser observadas em um agente [Franklin e Graesse, 1996]:

1. Posicionamento (*situatedness*): o agente recebe sinais de entrada dos seus sensores vindos do ambiente no qual está localizado e pode executar ações contextualizadas que modifiquem o ambiente de alguma forma;

2. Autonomia (*autonomy*): o agente deve ter a possibilidade de agir sem a intervenção direta de usuários ou de outros agentes, além de poder controlar totalmente suas ações e seu estado interno;

3. Pró-atividade (*pro-activeness*): os agentes não devem apenas agir em resposta ao seu ambiente, mas devem agir de maneira oportuna por iniciativa própria de acordo com seus objetivos;

4. Sociabilidade (*sociability*): os agentes devem poder interagir, quando apropriado, com outras entidades do ambiente de forma a solucionar seus problemas e ajudá-las nas suas atividades;

5. Adaptabilidade (*adaptiveness*): os agentes devem poder mudar o seu comportamento devido a uma experiência anterior;

6. Receptividade (*responsiveness*): os agentes devem poder perceber o seu ambiente e responder adequadamente a mudanças que ocorram nele;

7. Mobilidade (*mobility*): os agentes podem estar aptos a transportar-se de uma máquina para outra.

2.2.2 Classificação dos Agentes

A noção de agente autônomo tem um papel central na pesquisa contemporânea em Inteligência Artificial [Demazeau e Müller, 1989]. Um agente autônomo é um sistema com capacidade de decisão sobre as metas que orientarão sua atividade em um dado ambiente. Ele deve ser capaz de aprender com a experiência através de sua interação com o ambiente que o cerca, e generalizar esta aprendizagem para aprender melhor no futuro. Pode-se distinguir duas classes principais de agentes autônomos: os assim chamados cognitivos (ou deliberativos, ou ainda simbólicos), e os reativos.

Os **reativos** são baseados em modelos de organização biológica ou etológica, como por exemplo, as sociedades de formigas ou cupins. Uma formiga sozinha não é considerada uma entidade inteligente, mas o formigueiro sim, pois apresenta comportamento inteligente relacionado à busca e armazenamento de alimentos, além da defesa e organização dos berçários. O modelo de funcionamento de um agente reativo é o de estímulo-resposta. Em geral, esses agentes não apresentam memória, não planejam suas ações futuras e não se comunicam com outros agentes, tomando conhecimento das ações dos outros agentes pelas mudanças ocorridas no ambiente. Normalmente existem em grande quantidade no sistema e possuem baixa complexidade [Bittencourt, 1998b].

Os **cognitivos** são baseados em organizações sociais humanas como grupos, hierarquias e mercados. Os agentes possuem uma representação explícita do ambiente e dos outros agentes, dispõem de memória, e por isto são capazes de planejar ações futuras. Agentes

cognitivos podem comunicar-se entre si diretamente, isto é, seus sistemas de percepção e de comunicação são distintos, o que não acontece nos reativos. Normalmente existem em pequena quantidade no sistema e são de média ou alta complexidade. Além disso, requerem sofisticados mecanismos de coordenação e protocolos de suporte à interação de alto nível [Ferber, 1999].

A utilização de estados mentais para a modelagem de agentes cognitivos é chamada de abordagem mentalista [da C. Mora et al., 1999], onde o agente cognitivo possui estado interno que se relaciona com o estado do ambiente com o qual interage. Estes estados seriam correspondentes aos estados mentais humanos, que apresentam um vínculo com o mundo em termos da sua existência e significância. Crença, desejo, expectativa e intenção são exemplos de estados intencionais [Giraffa, 1999].

Uma crença de um agente corresponde às informações que o agente tem sobre o mundo (é o conhecimento do ambiente de forma explícita). Crenças podem ser vistas como simples variáveis (como por exemplo, na linguagem PASCAL), mas agentes modelados na arquitetura BDI representam crenças de forma simbólica (como por exemplo, fatos em PROLOG) [Zamberlam et al., 2000].

Um desejo de agente (ou objetivo em um sistema) intuitivamente corresponde a tarefas estabelecidas pelo próprio agente. Agentes BDI, assim como agentes humanos, não exigem que desejos sejam logicamente consistentes. Um desejo é um estado mental intencional e com potencial motivador das ações do agente [Zamberlam et al., 2000].

A intuição em sistemas BDI é que o agente não é capaz, geralmente, de realizar todos os seus desejos, mesmo sendo consistentes. Esses desejos escolhidos são adotados como intenções e um agente continuará a tentar realizar uma intenção até que acredite que a intenção foi satisfeita, ou acredite que a intenção não poderá ser realizada [Zamberlam et al., 2000].

A característica essencial da abordagem SMA é a filosofia de resolução distribuída de problemas, na qual adota-se uma estratégia de dividir para conquistar [Jennings, 1994]. A resolução cooperativa distribuída de problemas diz que um problema é dividido em subproblemas e cada um é executado separadamente por um agente, cada um destes comunicando ou cooperando entre si quando necessário, com a idéia básica de que a soma dos resultados locais corresponderá à solução do problema geral.

Existem dois tipos de cooperação [Durfee, 1987]: partilha de informação ou partilha de tarefas.

A **partilha de informação** se dá quando um dado agente dispõe ou produz informações parciais que julga serem úteis partilhar e as envia para os outros agentes.

A **partilha de tarefa** é efetuada quando um dado agente, ao decompor uma dada tarefa, detecta subtarefas que não pode ou não quer realizar, sendo necessário procurar outros

agentes que possam auxiliá-lo.

A área de sistemas multiagentes é vasta e por este motivo optou-se por apresentar apenas os conceitos essenciais para o entendimento dos conceitos utilizados nesta tese.

2.3 Ontologias

O termo ontologia surgiu, originalmente, como um ramo da filosofia que trata da existência de seres no mundo. Entretanto, atribuem-se diferentes definições para o termo ontologia, de acordo com a aplicação. Em ciências da computação, uma ontologia é um modelo de dados que representa um domínio e é usado para explicitar os objetos pertencentes a um domínio e as relações entre eles, ou seja, descreve o que pode ser computacionalmente representado sobre o mundo. As ontologias são utilizadas em Inteligência Artificial, Engenharia de Software e Web Semântica como uma forma de representação do conhecimento, ou seja, uma modelagem do mundo ou algo que faça parte deste. A definição mais citada na literatura e na comunidade de ontologias é a que segue: *Ontologia é uma especificação formal e explícita de uma conceituação compartilhada* [Gruber, 1993].

Uma ontologia geralmente especifica um vocabulário comum e é definida por conjuntos de:

- Classes de objetos (Conceitos): conjuntos, coleções ou tipos de objetos;
- Atributos: propriedades, características ou parâmetros que os objetos podem ter e compartilhar;
- Relações: como os objetos podem relacionar-se;
- Indivíduos (Instâncias).

A definição de uma ontologia busca solucionar problemas relacionados à falta de conhecimento compartilhado facilitando a comunicação entre as partes envolvidas. Ontologias objetivam capturar o conhecimento consensual de um modo genérico, podem ser reusáveis e compartilhadas entre aplicações (*software*) e por grupos de pessoas. Ontologias são normalmente construídas por um grupo de pessoas em diferentes locais [Gómez-Pérez, 1999].

Existem diferentes **tipos** de ontologias, de acordo com seu grau de genericidade[Gómez-Pérez, 1999]:

- Ontologias de representação definem as primitivas de representação - como frames, axiomas, atributos e outros - de forma declarativa.

- Ontologias gerais (ou de topo) trazem definições abstratas necessárias para a compreensão de aspectos do mundo, como tempo, processos, papéis, espaço, seres, coisas, etc.
- 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 tratam de um domínio mais específico de uma área genérica de conhecimento, como direito tributário, microbiologia, etc.
- Ontologias de aplicação referem-se ao conhecimento necessário à solução de um problema específico de um domínio, como identificar doenças do coração, a partir de uma ontologia de domínio de cardiologia. Normalmente, ela referencia termos de uma ontologia de domínio.

Existem diversas metodologias para o desenvolvimento de ontologias, como por exemplo: METHONTOLOGY [Corcho et al., 2003], On-To-Knowledge [Staab et al., 2001], Activity-First Method [Mizoguchi, 2003b]). Entre as metodologias propostas, existem algumas etapas que aparecem em boa parte delas Gómez-Pérez [1999]: especificação, conceitualização, formalização, implementação e manutenção.

Ontolingua [Stanford, a], RDF [W3C, b] e OWL (DAML+OIL) [W3C, a] figuram entre as linguagens de representação de ontologias mais difundidas. Assim como, WebODE [Corcho et al., 2003] e Protégé [Stanford, b] são algumas das ferramentas mais usadas para a construção de ontologias. Mizoguchi [2003a,b] faz uma análise das principais ferramentas, linguagens e metodologias existentes para o desenvolvimento de ontologias.

Ontologias e Agentes

O uso das ontologias em Sistemas Multiagentes é bastante desejável, pois as ontologias servem como ferramenta para organização, reuso e disseminação de conhecimento já especificado, facilitando a construção de novos agentes. Além disso, uma ontologia comum define um vocabulário com o qual os agentes trocarão mensagens (informações), este vocabulário nada mais é do que uma descrição dos objetos que pertencem ao domínio em questão. Ainda que os agentes compartilhem um mesmo vocabulário isso não implica que eles possuam o mesmo conhecimento. Também não se espera que um agente que se comprometa com uma ontologia seja capaz de responder a todas as perguntas que possam ser formuladas com o vocabulário compartilhado.

2.4 Da Instrução Assistida por Computador aos Sistemas Tutores Inteligentes

Os sistemas de Instrução Assistida por Computador (CAI - Computer Aided Instruction) surgiram na década de 50 do século *XX* em projetos nas áreas de Educação [Vicari e Giraffa, 2003]. Nas versões iniciais apresentavam instruções programadas que repetiam os métodos utilizados pelo paradigma behaviorista, vigente na época. Foram criados para oferecer suporte ao ensino de habilidades específicas, sem a utilização do modelo do aprendiz para orientar a forma da interação. O conteúdo é pré-programado pelo professor, baseado num currículo de referência e elaborado proceduralmente.

Na década de 70 buscou-se desenvolver um programa educacional que deixasse de ser um mero virador de páginas eletrônico. Com este objetivo introduziu-se, nos sistemas CAI existentes, técnicas de Inteligência Artificial e resultados da Psicologia Cognitiva dando origem aos sistemas de Instrução Assistida por Computador Inteligentes (ICAI - Intelligent Computer Aided Instruction), que baseiam-se no conteúdo e são independentes do método de ensino utilizado.

Originalmente, os ICAI foram utilizados como sinônimos dos Sistemas Tutores Inteligentes (STIs), mas isto é contestado em Informática na Educação [Vicari e Giraffa, 2003]. Basicamente, o que os diferencia é que nos Sistemas Tutores Inteligentes existe um modelo de aluno que objetiva personalizar o trabalho conforme as diferenças individuais de cada aprendiz, além da utilização deste modelo para a seleção das estratégias de ensino. Um sistema especialista voltado ao processo de ensino-aprendizagem pode ser considerado um sistema de Instrução Assistida por Computador Inteligente. No entanto, ele não possui um modelo de aluno e uma arquitetura do tipo dos Sistemas Tutores Inteligentes. Desta forma, tanto os ICAI quanto os STIs partem do princípio da "solução de problemas" logo, apresentam aplicações similares em educação [Vicari e Giraffa, 2003].

A Inteligência Artificial na Educação (IA-ED) [Kearsley, 1987; Wenger, 1987] aplica técnicas de Inteligência Artificial (IA) [Klassner, 1996; Russell e Norvig, 1995; Pereira, 2002] a programas computacionais educacionais, com o objetivo de construir modelos e arquiteturas que gerenciem, com algum grau de autonomia, as interações de um tutor artificial com os estudantes na busca de um aprendizado mais eficaz. As pesquisas na área de IA-ED deram origem a alguns importantes sistemas [Murray, 1999], [Self, 1988], [Self, 1990], [Wenger, 1987], [Clancey, 1984].

O sistema GUIDON [Clancey, 1984] destaca-se por ser o marco inicial na tentativa de adaptar as regras de domínio de um sistema especialista pré-existente dentro de um Sistema Tutor Inteligente. GUIDON destinava-se ao ensino de diagnóstico de doenças infecciosas do sangue, desenvolvido a partir da base de conhecimento já formada do MYCIN [Shortliffe, 1976], um sistema especialista para diagnosticar infecções sangüíneas. GUIDON exerceu

um papel importante na evolução das técnicas de Inteligência Artificial para sistemas de aprendizagem baseados em computador. Apesar da sua importância ele apresentou inúmeras falhas, como estratégias de ensino ineficientes e interações limitadas com o usuário.

Desde o sistema GUIDON até os sistemas atuais que utilizam inovações tecnológicas como os sistemas hipermídias e a Internet, a tarefa de construir um Sistema Tutor Inteligente composto de um rico domínio e que apresente características realmente adaptativas, continua sendo bastante complexa. Sob o ponto de vista do usuário o domínio do conhecimento deve ser estruturado e planejado de uma maneira atrativa e interessante, levando em consideração os estilos de aprendizagem, preferências pessoais e conhecimento prévio. Entretanto, sob o ponto de vista do professor, o desenvolvimento de um curso deve ser o mais simples e sistemático possível, o que pode ser contraditório com a flexibilidade desejada pelo estudante [Brusilovsky e Vassileva, 2003].

A falta de metodologias para ferramentas de autoria, a dificuldade de compartilhamento e reuso dos componentes do STI, a lacuna conceitual entre os sistemas de autoria e os autores, a distância entre o planejamento instrucional e a estratégia tutorial para a adaptação dinâmica do Sistemas Tutores Inteligentes Mizoguchi e Bourdeau [2000], são algumas das limitações encontradas nos Sistemas Tutores Inteligentes atuais.

2.4.1 Arquitetura Básica de um Sistema Tutor Inteligente

Um Sistema Tutor Inteligente (STI) é definido como um sistema instrucional baseado em computador, que ensina o estudante de maneira interativa, usando os conceitos de Inteligência Artificial [Auberger, 1998]. Um dos objetivos dos STIs é ser capaz de modelar comportamentos de ensino complexos, os quais se adaptam às necessidades do estudante, à situação de aprendizagem e ao assunto da instrução [Murray, 1999].

As funções operacionais básicas de um STI são determinadas pelos seguintes módulos/modelos (ver figura 2.1):

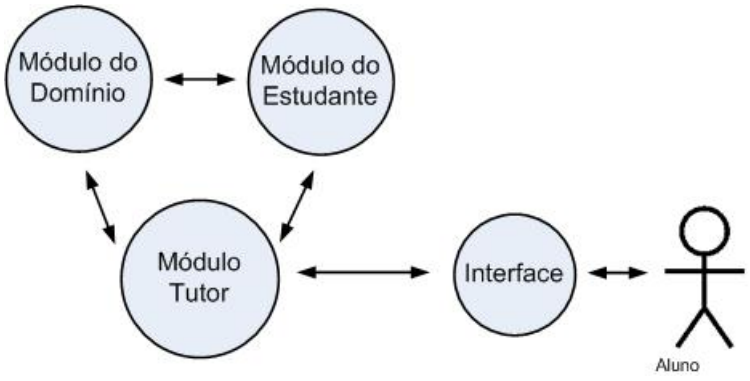


Figura 2.1: Arquitetura Clássica de um STI

- Domínio

  Este módulo, também denominado de Modelo Especialista, contém o conhecimento do domínio do sistema e os mecanismos de inferência [Auberger, 1998].

  O modelo de domínio é fundamentalmente uma base de conhecimento, contendo informações sobre um determinado domínio, organizada de forma a representar o conhecimento de um especialista ou professor. Geralmente, considera-se como o componente central de um STI, pois incorpora a maior parte da "inteligência" do sistema na forma do conhecimento necessário para solucionar problemas do domínio [Park, 1998].

  O grande desafio para cada novo STI é fornecer uma representação do seu domínio, rica o suficiente para suportar o nível desejado de compreensão proporcionando uma maior flexibilidade no ensino.

- Tutor

  Também conhecido como Modelo Pedagógico ou Instrutor, usa as informações do modelo do estudante para determinar *como ensinar*, ou seja, quais aspectos do conhecimento do domínio devem ser apresentados para o estudante. Representa os métodos e técnicas didáticas utilizadas no processo da comunicação de conhecimento, as estratégias de ensino-aprendizagem. O modelo pedagógico diagnostica as necessidades de aprendizagem do estudante com base nas informações do modelo do estudante e na solução do professor contida no modelo do especialista. Em geral, as decisões são sobre qual informação apresentar ao estudante e como apresentá-la, a partir das informações contidas nos modelos do aluno e de domínio.

  As decisões pedagógicas são tomadas no contexto de um ambiente educacional que determina o grau de controle sobre a atividade e sobre a interação possuídos respectivamente pelo sistema tutorial e pelo estudante [Wenger, 1987].

  Portanto, um processo de aprendizagem depende de uma grande variedade de fatores e o modelo pedagógico deve cuidar para não destruir a motivação pessoal do estudante ou o seu senso de descoberta.

- Interface

  É responsável por processar o fluxo de informações com o meio externo e o sistema. Traduz a representação interna do sistema numa linguagem compreensível para o aprendiz. "Para os usuários, a interface é o próprio sistema" [Hix e Hartson, 1993].

  Muitos princípios baseados em teorias cognitivas têm sido propostos para projetos de interface como resultado de pesquisas na área da interação homem-máquina. Entretanto, a meta da maioria destas pesquisas é que o usuário não necessite se adaptar à interface do sistema contudo, a interface deve ser projetada para que seja intuitiva tornando o aprendizado natural para o usuário.

  É importante salientar que em uma interação com o STI, o estudante não irá somente aprender o conteúdo das lições, mas também terá que aprender como utilizar o sistema,

portanto, a facilidade de uso deve ser uma das considerações principais no projeto destas interfaces. Uma interface consistente, ajudará a reduzir a carga cognitiva sobre o estudante [Shneiderman, 1992]. Mesmo sendo o modelo de interface muito relevante para o desempenho de um STI, esta tese não contempla este aspecto.

- Estudante

  Também conhecido como Modelo do Aluno, do Aprendiz ou do Usuário, tenta descrever o conhecimento do estudante num dado momento, juntamente com suas preferências e do histórico de suas atividades, sendo fundamental para o mecanismo de adaptação.

  O modelo do estudante pode ser descrito como uma representação abstrata do estudante no sistema, cuja principal função é permitir que o conteúdo instrucional seja adaptado para um indivíduo ou um grupo deles. Com as informações sobre o estudante, o tutor é capaz de controlar a ordem e a dificuldade do material a ser apresentado [Auberger, 1998].

  Este modelo deve contemplar todos os aspectos do conhecimento e do comportamento do estudante que tragam conseqüências para o seu desempenho e aprendizagem. Entretanto, a construção de um modelo como este é uma tarefa bastante complexa.

  A chave para um ensino personalizado e inteligente em um sistema tutorial é o conhecimento que o sistema deve ter de seu próprio usuário. A dimensão mais significativa de um STI é sua capacidade para modelar o conhecimento do estudante [Jonassen e Wang, 1993].

Para auxiliar o desenvolvimento de STI são desenvolvidas ferramentas com o objetivo de automatizar o processo de seleção e seqüenciamento dos elementos de aprendizagem, conhecidas como Ferramentas de Autoria.

2.4.2 Hipermídia Adaptativa

A Hipermídia Adaptativa estuda o desenvolvimento de sistemas capazes de promover a adaptação de conteúdos e recursos hipermídia, vindos de diversas fontes (bancos de dados, Internet, serviços etc.) e formatos (texto, áudio, vídeo, etc e suas combinações), de acordo com o perfil ou modelo de seus usuários [Palazzo, 2002].

A Hipermídia Adaptativa encontra aplicação direta, por exemplo, em educação, sistemas de informação, comércio eletrônico, marketing, medicina, lazer, necessidades especiais etc. Todavia, a hipermídia educacional e os sistemas de informações *on-line* são as duas áreas de maior concentração, tanto em pesquisas quanto em aplicações [Brusilovsky, 2001].

A Hipermídia Adaptativa Educacional foi inspirada na área dos Sitemas Tutores Inteligentes (STI) e surgiu na tentativa de se combinar o ensino fortemente guiado dos STI e

a busca livre da hipermídia educacional tradicional. Em outras palavras, existe a ação de um tutor que envia ao aprendiz material a ser estudado, de forma pedagógica, conforme o domínio e o modelo do estudante; há também a possibilidade do aluno discordar do tutor e pesquisar o material segundo sua vontade, podendo a qualquer momento voltar a ser auxiliado pelo tutor.

A pesquisa em Hipermídia Adaptativa situa-se na fronteira dos estudos em hipermídia e modelagem/modelo do usuário. Estes sistemas tentam antecipar as expectativas dos usuários a partir de modelos que representam seu perfil.

Para a construção do modelo do usuário é necessário identificar os próprios usuários, seus comportamentos, definir seu perfil, seu grau de conhecimento, antecipar suas preferências. Para definir um perfil, o sistema utilizará os dados cadastrais do usuário ou se baseará na sua navegação. O restante fica com os modelos, técnicas de adaptação e algoritmos.

Um sistema de Hipermídia Adaptativa deve satisfazer a três critérios básicos [Palazzo, 2002]:

1. ser um sistema hipertexto ou hipermídia;
2. possuir um modelo do usuário;
3. ser capaz de adaptar a hipermídia do sistema usando tal modelo.

Um sistema hipermídia é formado basicamente de **nodos** ou **nós** e a ligação entre os **nodos** é chamada de **link**. Os sistemas de Hipermídia Adaptativa podem se adaptar de duas maneiras: Apresentação adaptativa e Navegação adaptativa. Na apresentação adaptativa, o sistema adapta o conteúdo de um **nodo**, selecionando o conteúdo a ser apresentado. Na navegação adaptativa, o sistema adapta os **link** de ligação dos nodos. O sistema se adapta para mostrar conteúdos ou indicar o caminho para temas de sua preferência, previstos através de seu perfil de usuário.

2.5 Trabalhos Relacionados

Os avanços nos mecanismos de adaptação dos Sistemas Tutores Inteligentes e dos Sistemas Hipermídia Adaptativos apesar de lentos são bastante significativos para a evolução das referidas áreas de pesquisa. Um relato importante sobre a referida área é feito em [Brusilovsky e Peylo, 2003] e [Brusilovsky, 2000].

O uso de Redes de Petri em Sistemas Hipermídia é bastante difundido, algumas ferramentas e modelos hipermídia encontrados na literatura são: Trellis [Na e Furuta, 2001], MORENA [Botafogo e Moss, 1995] e HTSPN [Willrich et al., 2002]. Contudo nestes sistemas, o professor necessita de uma certa familiaridade com o formalismo pois ele é obrigado

a trabalhar diretamente com as Redes de Petri descrevendo a ordem de apresentação das páginas.

Um sistema hipermídia que pode ser comparado aos STI é LAOS [Cristea, 2004; Cristea e Kinshu, 2003], um modelo de autoria adaptativo hierárquico de 5 camadas (modelo de domínio, modelo de restrições e objetivos, modelo de usuário, modelo de adaptação e modelo de apresentação). A representação do conhecimento é feita através de mapas conceituais.

Algumas plataformas de Sistemas Tutores Inteligentes foram selecionadas através do critério de importância no cenário científico e devido as suas semelhanças com a proposta apresentada nesta tese.

ADIS [Warendorf e Tan, 1997] é um STI desenvolvido para assistência ao ensino no curso de Estrutura da Dados. ADIS tem a capacidade de apresentar o conteúdo graficamente na tela do computador permitindo a manipulação destes gráficos. O seu módulo tutorial apresenta exercícios, onde os estudantes podem aprender algoritmos básicos de Estrutura de Dados visualmente. ADIS é implementado em Java, o tutor é um Java *applet* que é carregado e executado na máquina Cliente. O modelo do aprendiz fica no Servidor permitindo assim, ao aprendiz acessar o tutor inúmeras vezes e de diferentes locais.

CALAT [Nakabayashi et al., 1997] é um STI cuja arquitetura é Cliente/Servidor. Usa um navegador de *Internet* no cliente onde, o estudante acessa o servidor CALAT que permite uma capacidade individual de adaptação. CALAT está dividido em 3 tipos de páginas de apresentação: (a) textos, que apresentam o conteúdo (b) exemplos, que são gerados dinamicamente em HTML (c) exercícios ou simulações, que podem ser questões, seleção de verdadeiro/falso ou descritivas.

I-Help [Vassileva et al., 2001] é um sistema para ambientes de aprendizagem baseado na *Web* visando auxiliar os aprendizes na solução de problemas. Explora recursos como fóruns e *chat*. Foi desenvolvido em Java e possui uma arquitetura multiagente. Os agentes utilizam ontologias e linguagem de comunicação comuns. Todos os agentes são autônomos e colaborativos.

Teach Nets [Liu et al., 2002] faz uso da descrição formal baseada nas Redes de Petri de alto nível para autoria e para navegação no curso. Fichas coloridas são usadas para modelar os estudantes e os objetos de aprendizagem. Teach Nets também se beneficia das características de modelagem das Redes de Petri na definição das seqüencias dos conteúdos e na estratégia pedagógica adotada.

Os projetos em andamento REDEEM [Grimshaw et al., 2000; Ainsworth e Fleming, 2006], CTAT [Aleven et al., 2006; V. Aleven e Koedinger, 2006] e XTA [Nuzzo-Jones et al., 2005] são descritos em um nível de detalhe com o intuito de se apresentar o estado da arte sobre as plataformas e ferramentas de autoria.

REDEEM

REDEEM [Grimshaw et al., 2000; Ainsworth e Fleming, 2006] é uma ferramenta de autoria e um arcabouço para STI.

A ferramenta de autoria permite aos autores de um curso descreverem como eles gostariam de ensinar através da descrição das estruturas, do fluxo e da seqüência do conteúdo do curso. A ferramenta de autoria é usada da seguinte forma:

- O autor nomeia as páginas e as classifica em sessões de acordo com o tipo de material (fácil, geral, introdutório,...);
- O autor descreve as relações entre as sessões, por exemplo as relações de pré-requisitos. Estas informações são utilizadas para a escolha de como organizar o material através do cálculo de pesos de preferências;
- O autor fornece as questões e o retorno dado ao estudante que explica porque uma resposta é correta. O autor pode criar cinco dicas diferentes para cada questão;
- O autor define um conjunto de categorias de estudante em qualquer grau de granularidade, usualmente baseada em performance ou tarefas;
- O autor pode criar estratégias do tipo: Livre Descoberta ou Guiado por exemplo. REDEEM suporta múltiplas estratégias.

  Por definição, um estudante vê todo o conteúdo do curso, no entanto, o professor pode remover uma sessão para uma determinada categoria de estudante.

O arcabouço para STI usa a saída da ferramenta de autoria juntamente com suas próprias definições para apresentar o material ao estudante de uma maneira adaptativa e interativa. O papel principal do arcabouço para STI é fornecer o material do curso ao estudante de acordo com as especificações feitas pelo professor na ferramenta de autoria. As ações tutoriais disponíveis no arcabouço são: ensinar um novo conteúdo, apresentar um problema, sugerir que o estudante faça anotações, etc. Para realizar estas funções REDEEM aplica um modelo *overlay* que registra a percepção do sistema sobre o aprendizado do estudante em um determinado assunto. O arcabouço possui também um histórico do estudante, utilizado para fornecer ao professor um relatório sobre o progresso de estudantes individuais ou uma categoria deles.

CTAT

O CTAT (Cognitive Tutor Authoring Tools) [Aleven et al., 2006; V. Aleven e Koedinger, 2006] é um projeto em andamento com o objetivo de criar um conjunto de ferramentas de autoria para o desenvolvimento de dois tipos de tutores: Cognitivos e Baseado em Exemplos.

Os tutores Cognitivos apresentam um modelo baseado em regras, exigindo programadores de Inteligência Artificial e os tutores Baseado em Exemplos possuem as mesmas funcionalidades dos tutores Cognitivos no entanto não exigem nenhum tipo de programação.

A ferramenta de autoria para tutores cognitivos possui três aplicativos sendo dois deles externos e um o corpo central da ferramenta Aleven et al. [2006]:

1. Construtor de Interface (externo): usado para criar a interface do estudante, em um ambiente baseado em problemas; Compatível com construtores de interface;
2. Registrador de Comportamento: ferramenta principal com três funções-chave:
   (a) Registra os exemplos de comportamentos corretos e incorretos demonstrados pelo autor, na interface do estudante na forma de um grafo de comportamento;
   (b) Implementa a função *Example-Tracing*;
   (c) Fornece suporte para o planejamento e teste de modelos cognitivos;

   Editor de Memória de Trabalho: permite ao autor inspecionar e modificar o conteúdo no modelo cognitivo na memória de trabalho, na forma de uma coleção de fatos Jess;

   Árvore de Conflitos e Janelas Porque não: ferramenta para depurar o modelo fornecendo informações sobre ativação de regras e ativações parciais exploradas pelo algoritmo *model-tracing*. A árvore de conflitos é específica para a trajetória do modelo, mas a janela porque não é útil para a programação das regras de produção gerais;

   Console do Jess: permite que o autor interaja diretamente com o interpretador de regras (Jess) pela linha de comando, usado para carregar as estratégias de depuração não suportadas diretamente pela CTAT.
3. Editor de modelos cognitivos (externo): usado para editar as regras Jess para o modelo.

Nos tutores Baseados em Exemplos, o autor demonstra comportamentos de resolução de problemas corretos e incorretos que ficam armazenados num banco de dados da ferramenta. O autor generaliza os exemplos registrados que servem de base para o tutor ou pode ainda utilizá-los como guia de desenvolvimento no tutor Cognitivo.

A ferramenta de autoria para tutores Baseado em Exemplos apresenta quatro recursos distintos V. Aleven e Koedinger [2006]:

1. Registro de Comportamento: ferramenta para criação de grafos de comportamentos mapeados no espaço de soluções;
2. Facilidade de produção em massa: todas as instâncias do problema são representadas em um único arquivo, produzidas através de modelos de autoria;

3. Loja de Tutor: componente usado para controlar a seqüência de problemas de tutoria em um ambiente baseado na *Web*; armazena as informações referentes ao percurso efetuado pelo estudante permitindo obter a sua localização na seqüência do problema;

4. Loja de Dados: ferramenta com serviços de análise de *log* e relatórios; responsável por padronizar todos os conjuntos de dados coletados.

XTA

XTA (eXtensible Tutor Architecture) [Nuzzo-Jones et al., 2005] é uma plataforma para criar STIs. Esta plataforma controla a interface e os comportamentos do STI através de unidades modulares. Estas unidades consistem conceitualmente de: uma unidade Currículo, uma unidade Problema, uma unidade Estratégia e uma unidade de *Logging*.

- Currículo

  A unidade Currículo pode ser dividida em duas partes: o currículo e as seções. O currículo é formado por uma ou mais seções e cada seção contém problemas ou outras seções. Esta estrutura recursiva permite uma hierarquia com diferentes tipos de seções e problemas.

  As sessões que formam um determinado currículo são armazenadas em um arquivo XML que indexa um currículo e seus problemas. Existe um arquivo por estudante e por currículo.

  Uma seção é uma abstração de uma lista particular de problemas. Esta abstração foi estendida para implementar o tipo de seção. Os tipos de seções incluem:

  - Linear: problemas ou sub-seções são apresentados linearmente;
  - Aleatório: problemas ou sub-seções são apresentados em uma ordem pseudo-aleatórias;
  - Experimental: um problema ou uma sub-seções é selecionado pseudo-aleatoriamente de uma lista, o restante é ignorado;

  Existem planos para a inserção de outros tipos de seções que incluem a seção Direta cuja seleção de problemas é conduzida pelo modelo de conhecimento do estudante.

- Problema

  A unidade do Problema representa um problema a ser ensinado, que inclui questões, respostas e outros componentes necessários para solucionar um problema.

- Estratégia

  A unidade Estratégia permite um alto grau de controle dos problemas e fornece suporte ao fluxo de controle entre os problemas. Esta unidade consiste de um tutor de estratégia

e a agenda. Diferentes tutores de estratégias podem fazer com que a apresentação de um mesmo problema seja apresentada de diferentes maneiras.

Os problemas podem ser organizados pelo tutor de estratégia em uma árvore. Quando um estudante responde incorretamente um problema, uma seqüência de outros problemas associados com a resposta incorreta entra na fila de apresentação.

Outros tutores de estratégia seriam:

- Mensagem: mostra uma seqüência de mensagens como dicas ou outras instruções;
- Explicação: mostra uma explicação para o problema além do próprio problema;
- Forçada: força o aluno a ver uma parte específica do problema.

A seleção dinâmica do conteúdo é feita pela Agenda. A Agenda é uma coleção de problemas organizados em uma árvore que deve ser completada. O conteúdo da agenda é operados pelos diversos tutores de estratégias que selecionam novos problemas para as seções com um currículo que adiciona e escolhe o próximo problema a ser navegado.

- *Logging*

  A unidade de *Logging* recebe as informações detalhadas por todas as outras unidades relacionadas com as ações do usuário e os componentes de interações. Estas mensagens são do tipo: iniciando um currículo, iniciando um problema, estudante respondendo uma questão, etc.

Comparativo

Dentre as ferramentas de autoria apresentadas, a forma de inserção e organização dos conteúdos são bastante semelhantes, entretanto a FAST apresenta vantagens em relação a outros aspectos. Por exemplo, uma das grandes vantagens da CTAT, apontada pelos próprios autores, é o uso de recursos externos como a adição *plug-and-play* de um construtor de interfaces do tipo Macromedia Flash, no entanto isto exige do professor uma certa familiaridade com este tipo de tecnologia, o que na maioria das vezes acaba sendo um empecilho para a utilização da própria ferramenta, cujo excesso de flexibilidade acarreta uma desmotivação antes mesmo de se utilizar a ferramenta propriamente dita. Em comparação ao XTA, a FAST permite múltiplas estratégias, enquanto que a XTA apresenta três modelos estratégicos pré-definidos. REDEEM assim como a FAST permitem múltiplas estratégias, no entanto a FAST tem a vantagem de que um mesmo estudante possa receber problemas modelados com estratégias distintas. Além de múltiplas estratégias a FAST permite a introdução de novas estratégias de forma declarativa, sem modificação do sistema.

A tabela 2.1 apresenta um comparativo entre o CTAT Cognitive, o REDEEM e o XTA. Verifica-se que nenhuma das ferramentas em questão fornecem suporte a adaptação de apresentação do conteúdo de acordo com o perfil do estudante. Tanto o CTAT Cognitivo, quanto o XTA

Propriedades	**CTAT Cognitivo**	**REDEEM**	**XTA**
Navegação Adaptativa	não	não	não
Regras de Produção	sim	não	sim
Grafos de Pré-requisitos	sim	não	sim
Múltiplas Estratégias	sim	sim	não

Tabela 2.1: Comparação de Ferramentas

usam técnicas de inteligência artificial como as regras de produção, além dos grafos de pré-requisitos que auxiliam na construção dos cursos por ser uma ferramenta gráfica. Em relação as múltiplas estratégias suportadas pode-se afirmar que mesmo CTAT Cognitivo e o REDEEM permitirem múltiplas estratégias não existe um mecanismo de expansão e atualização de novas estratégias fazendo com que o professor tenha que adotar uma das estratégias previamente definidas.

Capítulo 3

Projeto MathNet

As pesquisas para a construção de um arcabouço para Sistemas Tutores Inteligentes tiveram início em 1998 no contexto do projeto de pesquisa de Informática na Educação Math_Net (Uma abordagem via sistemas multiagentes para concepção e realização de ambientes interativos de aprendizagem cooperativa assistidos por computador [Mathnet, 2000]), apoiado pelo CNPq através do programa ProTeM-CC (processo n○ 68.0060/99-5). O tema central do projeto é a concepção e o desenvolvimento de um modelo computacional para Ambientes Interativos de Ensino/Aprendizagem Cooperativa com base em múltiplos agentes artificiais e humanos, dispostos em uma estrutura de rede de computadores. A pesquisa e o desenvolvimento de tal ambiente foi realizada através de um consórcio constituído por professores e pesquisadores das Universidades Federais de Alagoas (UFAL), de Santa Catarina (UFSC) e do Maranhão (UFMA). Além disso, o projeto recebeu o apoio da Fundação de Amparo à Pesquisa de Alagoas (FAPEAL) com bolsas de pesquisa. Apesar do financiamento do projeto ter sido encerrado em 2001 as pesquisas continuam sendo realizadas no âmbito de teses [Cardoso et al., 2004b,a; Pozzebon et al., 2006] e dissertações [Frigo, 2002; Postal, 2004; Yamane, 2006] apoiadas pela Capes e pelo CNPq.

O grupo da UFAL [do S. Da Silva e Hernandez-Dominguez, 1999; Labidi et al., 2000] investiu na proposta de evolução e consolidação da sociedade de agentes do MATHEMA tendo duas linhas de trabalho: (i) cuidar das estratégias de interações cooperativas (problema da decomposição e alocação de tarefas) entre os agentes tutores, e a segunda preocupou-se com aspectos internos na definição de um agente tutor; (ii) tratar os aspectos internos na definição de um agente investindo na questão da modelagem do aprendiz.

A UFMA [Coutinho et al., 2000, 2001] atuou fortemente em três grandes frentes: (i) o estudo e desenvolvimento de ferramentas para interação intra e inter grupos, grupos e professores, e grupos + professores com o sistema; (ii) relativas ao ambiente de autoria e (iii) a modelagem do aprendiz/grupos de aprendizes. Além disso, outros investimentos foram feitos em sistema para recuperação de informação na Web e no desenvolvimento de protótipos de sistemas.

O trabalho desenvolvido pelo grupo da UFSC [Freitas e Bittencourt, 2000; Costa e Bittencourt, 2000] concentrou-se no estudo de estratégias de cooperação entre agentes e na definição e desenvolvimento de sistemas tutores baseados no modelo MATHEMA. Além disso, tem havido neste grupo um forte e conseqüente investimento no tema de Recuperação de Informação. Este tema, como já foi dito, tem sido tratado pelos 3 grupos do consórcio.

Neste capítulo tem-se a apresentação do modelo MATHEMA [Costa, 1997] que serviu de suporte para estruturar a representação do domínio de conhecimento em um Sistema Tutor Inteligente e a arquitetura de implementação. Apresenta-se um breve histórico do projeto MathNet do grupo pertencente a Universidade Federal de Santa Catarina. Conclui-se este capítulo com a apresentação da versão atual do sistema que consiste de um arcabouço e uma ferramenta de autoria para os Sistemas Tutores Inteligentes chamados respectivamente de ASTI (Arcabouço para Sistemas Tutores Inteligentes) e FAST (Ferramenta de Autoria para Sistemas Tutores).

3.1 MATHEMA

Esta seção aborda os aspectos do MATHEMA usados como suporte ao desenvolvimento da FAST e da ASTI.

3.1.1 Modelagem do Conhecimento sobre um Domínio

No modelo proposto em [Costa, 1997] a modelagem do conhecimento sobre um dado domínio é particionada e organizada segundo duas formas de visualização: uma visão externa e uma visão interna.

A **visão externa** propõe-que um dado domínio do conhecimento seja particionado em diferentes subdomínios. Cada particionamento representa uma visão particular do domínio.

Visando alcançar um conhecimento com especialidades distribuídas, a busca foi orientada em três dimensões de conhecimento:

Contexto (C) compõe-se de diferentes interpretações a respeito de um domínio de conhecimento, constituindo-se de representações ou abordagens diferentes de um mesmo objeto de conhecimento. Representam os pontos de vista.

Profundidade (P) relativa a um contexto particular, refere-se a um refinamento na linguagem de percepção, ou seja, estratificação dos vários níveis epistemológicos de percepção do objeto de conhecimento.

Lateralidade (L) referente aos conhecimentos de suporte relacionados ao domínio de aplicação, proveniente de uma visão particular de contexto e profundidade.

De acordo com a visão externa, a um domínio de conhecimento D associa-se um conjunto de contextos distintos, onde:

$$D \rightarrow \{C_1, C_2, ..., C_n\} \tag{3.1}$$

Para cada C_i fixado, atribui-se um conjunto de níveis diferentes de profundidade:

$$C_i \rightarrow \{P_{i1}, P_{i2}, ..., P_{im}\} \tag{3.2}$$

Finalmente, a cada par $\langle C_i, P_{ij} \rangle$, associa-se um conjunto de lateralidades distintas:

$$\langle C_i, P_{ij} \rangle \rightarrow \{L_{ij1}, L_{ij2}, ..., L_{ijt}\} \tag{3.3}$$

O domínio do conhecimento pode ser modelado de acordo com n pontos de vista, onde cada um possui m_i abordagens pedagógicas que podem ser particionadas em $\sum_{i=1}^{n} m_i$ subdomínios estando cada um deles sob a responsabilidade de um agente tutor.

A **visão interna** equivale a um particionamento do domínio. Cada subdomínio é constituído por um conjunto de unidades pedagógicas, definidas em função de objetivos de ensino/aprendizagem específicos, associados a um **curriculum**. As unidades pedagógicas são relacionadas segundo uma ordem definida com base em critérios pedagógicos e a cada uma corresponde um conjunto de problemas. A cada problema está associado um conhecimento de suporte à sua resolução, que pode incluir conceitos, exemplos, contra-exemplos, dicas etc. Pode-se representar um curriculum como:

$$Curriculum = \{up_1, up_2, ..., up_n\} \tag{3.4}$$

Sendo que cada up_i denota uma unidade pedagógica do **curriculum** estando relacionadas segundo uma ordem definida com base nos critérios pedagógicos. A cada up_i corresponde um conjunto de problemas e a cada problema está associado um conhecimento de suporte à sua resolução, organizado conforme afigura 3.1.

3.1.2 Arquitetura

Costa [1997] propõe uma arquitetura multiagente para implementar um Sistema Tutor Inteligente responsável por um domínio do conhecimento onde cada subdomínio, como definido na seção 3.1.1, é implementado por um agente. Estes agentes formam a Sociedade de Agentes Tutores Artificiais (SATA). Esta arquitetura é mostrada na figura 3.2, na qual fazem parte, além da SATA, os seguintes componentes:

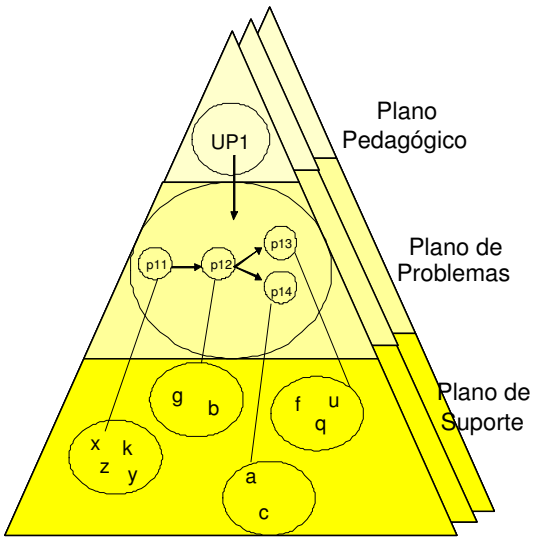


Figura 3.1: Planos

Aprendiz Humano (AH) : agente interessado em aprender o conteúdo sobre um dado domínio.

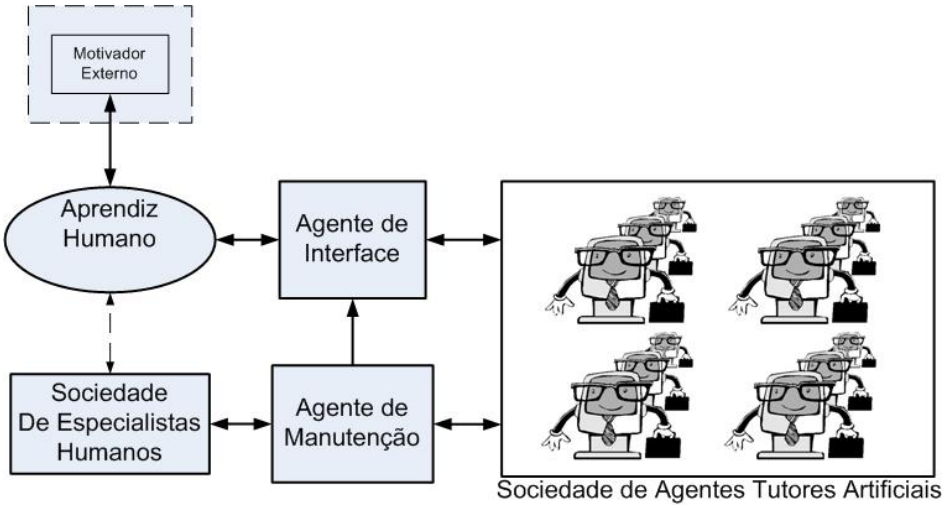
Sociedade de Agentes Tutores Artificiais (SATA) : conjunto de agentes capazes de cooperarem entre si a fim de promover a aprendizagem de um dado aprendiz. Cada agente integrante da SATA, chamado Agente Tutor (AT) é especializado em subdomínios relacionados a um dado domínio de conhecimento. Essa idéia foi inicialmente inspirada nas reflexões de Minsky contidas no livro Sociedade da Mente [Minsky, 1989 apud Costa, 1997].

Sociedade de Especialistas Humanos (SEH) : funciona como uma fonte de conhecimento externa ao sistema e se comunica com a SATA através dos agentes de manutenção. Suas principais funções são: (a) promover o processo de manutenção da SATA, que diz respeito à inclusão e exclusão de agentes, bem como alterações no conhecimento dos agentes; (b) estar disponível para que, em casos onde a SATA não consiga resolver um determinado problema, possa auxiliar os agentes e monitorar o andamento dos aprendizes.

Agente de Interface (AI) : é o agente responsável pela interação entre o aprendiz e a SATA.

Agente de Manutenção(AM) : é o agente responsável pela interface entre a SATA e a SEH. Sua função é a de prover esta interação, oferecendo meios necessários para a percepção, comunicação e manutenção da SATA.

Motivador Externo (ME) : representa as entidades humanas externas que podem motivar o aprendiz à utilização do ambiente computacional baseado no MATHEMA.



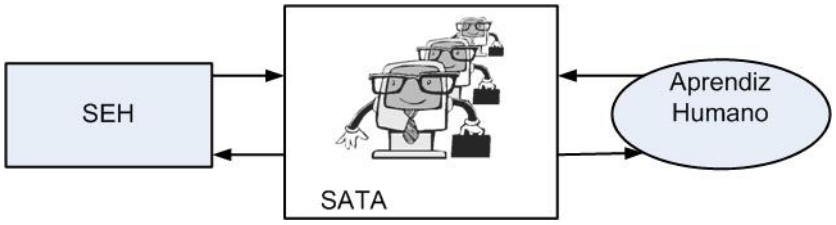
Fonte: [Costa, 1997]

Figura 3.2: Arquitetura MATHEMA

3.1.3 Interações no MATHEMA

A arquitetura proposta em [Costa, 1997] considera, essencialmente, três tipos de interações (ver figura 3.3):

- Interações entre um aprendiz e a SATA através de um agente tutor.
- Interações entre os agentes tutores.
- Interações entre SATA e SEH.



Fonte: [Costa, 1997]

Figura 3.3: Interações

As interações mais usuais no modelo MATHEMA podem ser descritas da seguinte forma:

Inicialmente, um Motivador Externo incentiva um aprendiz a trabalhar no ambiente computacional gerado a partir do modelo MATHEMA. A partir deste momento, acontece uma interação entre o aprendiz e o Agente de Interface (AI), sendo que o aprendiz informa ao AI os seus objetivos. O AI por sua vez, deve prover informações sobre o ambiente computacional e deve auxiliá-lo mediante análise de seus objetivos a escolher um supervisor na

SATA. Nesta etapa, inicia-se um processo de interação cooperativa e didática entre o aprendiz e o agente Supervisor, sendo que este passa a ser o responsável pelo aprendiz. Durante as interações no sistema, situações com diferentes níveis de complexidade podem ocorrer.

As interações dinâmicas entre um aprendiz e um agente tutor ocorrem num contexto de resolução de problemas, ativando diferentes funções pedagógicas durante o processo adaptativo. O agente responsável pela interação com o aprendiz pode solicitar a cooperação de outros agentes quando verificar que não pode realizar determinada tarefa. As interações entre as sociedades SATA e SEH ocorrem quando um agente tutor faz alguma solicitação a SEH ou quando a SEH constata que existe necessidade de intervenção. Esta constatação pode vir da observação do processo interativo entre aprendiz e SATA ou através de uma notificação da SATA.

3.1.4 Sociedade de Agentes Tutores Artificiais

Modelo de agente tutor

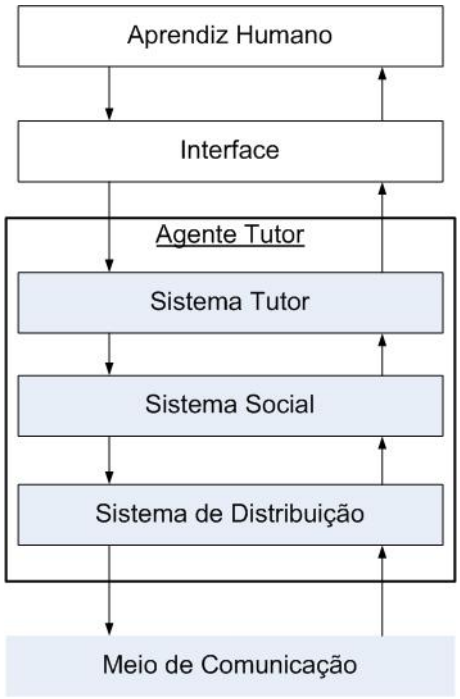
Os agentes cooperam entre si através de linguagens e protocolos, a fim de oferecer ao aprendiz condições mais efetivas no suporte às atividades de aprendizagem. São agentes do tipo inteligentes (ou cognitivos) e são assumidos como benevolentes, pois concordam em cooperar com os demais agentes quando solicitados.

Cada agente é uma entidade especializada em um domínio específico possuindo como principais características autonomia, onde cada agente têm seus próprios objetivos; habilidade social, onde os agentes interagem entre si e com o mundo externo à SATA e ainda são orientados a objetivos, onde cada agente exibe um comportamento de acordo com suas capacidades.

A arquitetura de um agente tutor, segundo o MATHEMA, pode ser vista em dois níveis distintos de abstração: macro e micro. Apresenta-se apenas o Agente Tutor no Nível Macro pois a contribuição desta tese refere-se a este nível.

Segundo o nível macro um agente é composto por três componentes principais:

- Sistema Tutor: interage diretamente com o aprendiz e armazena os conhecimentos que o agente possui para efetuar operações pedagógicas no domínio de aplicação.
- Sistema Social: possui bases de conhecimento e mecanismos de raciocínio necessários ao comportamento cooperativo entre os agentes tutores, refletindo o comportamento social do agente.
- Sistema de Distribuição: manipula a troca de mensagens entre agentes, através do meio de comunicação. Gerencia internamente a distribuição das mensagens no agente tutor.



Fonte: [Costa, 1997]

Figura 3.4: Agente Tutor - Nível Macro

Cada agente possui um sistema tutor inteligente associado, responsável por interagir diretamente com o aprendiz. Neste sistema o domínio de conhecimento, o aprendiz e ainda outros componentes são modelados de forma distribuída, buscando atingir um processo educacional que gere resultados efetivos, através de uma solução eficaz para a questão da adaptabilidade do sistema ao estado cognitivo do aprendiz.

Com base na modelagem do conhecimento, a Sociedade de Agentes Tutores é definida. Para cada subdomínio de um domínio D é associado um agente específico, obedecendo a seguinte relação:

$$D_{ij} \rightarrow AT_{ij} \tag{3.5}$$

A mesma idéia é considerada no tratamento do conhecimento lateral, ou seja, a cada visão de lateralidade é atribuído um agente:

$$dl_{ijk} \rightarrow AT_{ijk} \tag{3.6}$$

Sendo assim, a sociedade de agentes tutores artificiais (SATA) em relação a um domínio de conhecimento D é definida como:

$$SATA = AT \cup ATL \tag{3.7}$$

onde AT é o conjunto dos agentes tutores relativos a um domínio D, e ATL representa o conjunto dos agentes tutores associados a um domínio lateral DL.

3.2 Histórico do Projeto MathNet-UFSC

A construção de um arcabouço para Sistemas Tutores Inteligentes é uma tarefa bastante árdua e o objetivo final ainda não foi alcançado ora pela complexidade de desenvolvimento ora pela falta de recursos para se construir uma equipe de desenvolvimento. Grande parte dos resultados alcançados advém de esforços individuais que são somados e pouco a pouco deram origem as duas versões de um STI específico conhecido como MathTutor que serviu de suporte ao desenvolvimento da FAST e ASTI apresentadas nesta tese.

3.2.1 MathTutor v.1

O MathTutor é um STI específico para um domínio de conhecimento da área de informática, a disciplina de Fundamentos da Estrutura da Informação [Bittencourt, 1998a] cuja modelagem é apresentada no Capítulo 5.

Terezinha de Fátima Faria [de F. Faria, 2001] deu início aos trabalhos para a construção do MathTutor.

A navegação permitia seguir para o próximo tópico da lição ou mudar de texto para exercício ou vice-versa, utilizando a estrutura convencional de um livro.

As páginas estavam estruturadas em *frames* que correspondiam às seguintes funções: Busca, Navegação, Conteúdo, Índice das lições.

As ferramentas de desenvolvimento eram: Java como linguagem de programação; JESS (Java Expert System Shell) [Friedman-Hill, 1997], para o sistema especialista; JATLite (Java Agent Template, Lite) [JATLite, 1997], para a construção dos agentes e páginas Servlets para a interface com o usuário. Os Servlets usam o protocolo HTTP e podem ser acessadas de qualquer navegador, permitindo o uso do sistema como uma ferramenta de ensino à distância.

Os principais esforços foram quanto à escolha das ferramentas, buscando sempre uma interoperabilidade entre as mesmas e que fossem, fundamentalmente, ferramentas de domínio público. A portabilidade do MathTutor sempre esteve entre os fatores determinantes nas escolhas.

Nesta fase uma arquitetura da operação dos agentes foi definida e alguns testes de comunicação foram realizados através de um simulador do servidor de páginas. Embora simples, esta versão permitiu aos desenvolvedores conhecerem as reais dificuldades em se construir um arcabouço para STI.

Entretanto, na versão 1 do MathTutor aspectos importantes não eram levados em consideração, como por exemplo: as diferenças entre os aprendizes, o seu conhecimento prévio, as preferências pessoais e suas prévias interações com o tutor. O modelo do estudante era bastante simplificado e o sistema apenas respondia aos estímulos fornecidos pelo estudante.

3.2.2 MathTutor v.2

Na segunda versão do MathTutor desenvolvida por esta autora [Frigo, 2002] foram realizadas modificações buscando-se uma maior aproximação com o modelo conceitual MATHEMA, sendo a inclusão do agente de interface a mais significativa. Ocorre, a partir dessa versão, uma integração efetiva dos agentes da SATA através da troca de mensagens entre eles.

As mensagens provenientes do estudante ou do professor são recebidas e tratadas pelo agente de interface que repassa aos demais agentes da SATA, de acordo com as responsabilidades de cada um. Mas isto não impede que um agente receba um problema fora do seu domínio de conhecimento. Quando isto acontece, o agente deve perceber e avisar aos demais agentes que uma busca está sendo realizada.

A principal contribuição desta versão do MathTutor foi de tornar os agentes tutores mais especializados, deixando a responsabilidade de comunicação com os aprendizes e especialistas sob a responsabilidade do agente de interface. Assim, os agentes da SATA adquirem cada vez mais as propriedades propostas para eles no modelo conceitual MATHEMA.

Além dos aspectos relativos aos agentes, outras modificações foram realizadas, com o intuito de aperfeiçoar o sistema em questão, como por exemplo: (a) alterações da base de regras que compõem o MathTutor na medida em que o STI foi evoluindo e solicitando novas funções e adaptações; (b) organização dos arquivos que formam a interface através da separação física dos arquivos que formam a base de conhecimento; (c) padronização das páginas e dos conteúdos a serem apresentados.

As diferenças básicas entre as versões 1 e 2 do MathTutor:

Versão 1	**Versão 2**
Servidor Simulado	Servidor real
Interface simples	Interface amigável
Esquema do agente	Agente de Interface Simples operando
Sem servidor de páginas	Com servidor de páginas

A próxima seção trata dos principais aspectos que evoluíram em relação ao modelo MATHEMA e as versões 1 e 2 do MathTutor.

3.3 Arquitetura do Sistema

Para a construção do arcabouço para sistemas tutores refinou-se a arquitetura do MATHEMA (ver seção 3.1.2 e figura 3.2) em uma nova arquitetura representada na figura 3.5. Tanto os agentes humanos quanto os componentes computacionais sofreram modificações que serão apresentadas a seguir.

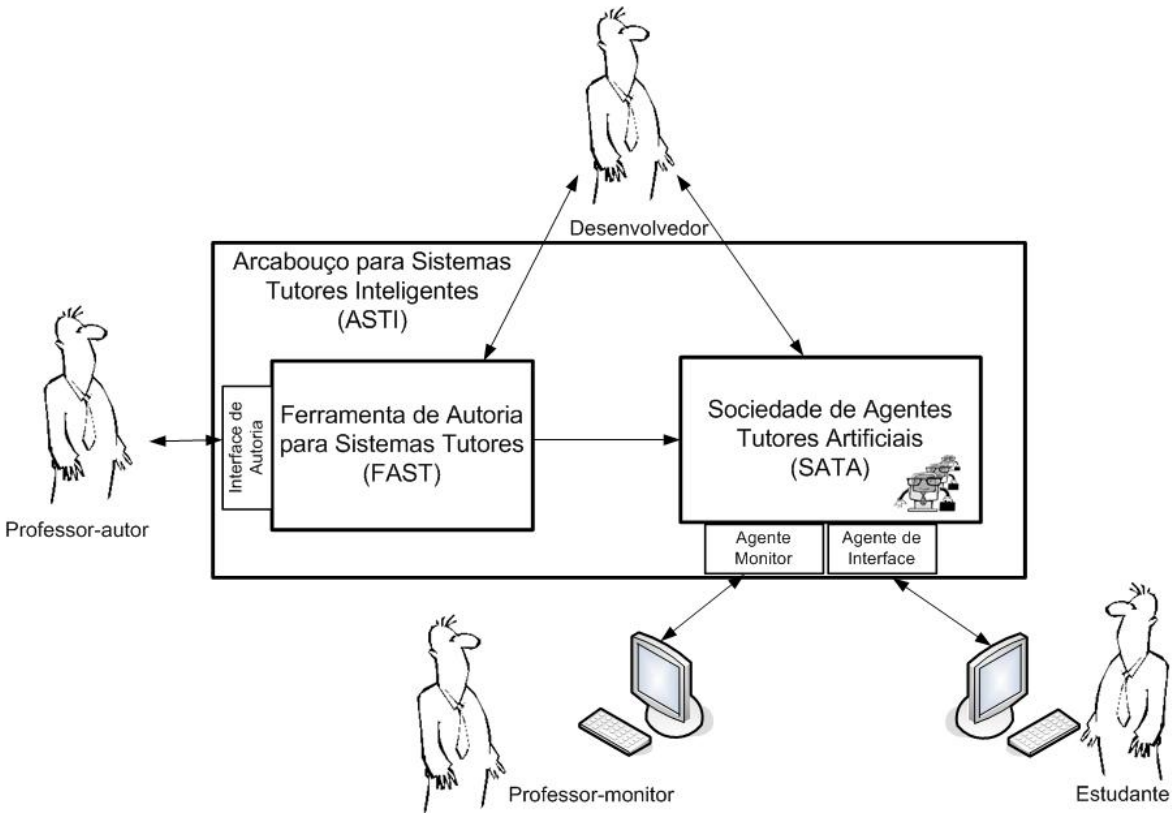


Figura 3.5: Arquitetura do Sistema

3.3.1 Agentes Humanos:

O motivador externo, que interagia apenas com o estudante, passou a ser chamado de professor-monitor e agora interage também com a Sociedade de Agentes Tutores (figura 3.5). A Sociedade dos Especialistas Humanos foi dividida em duas entidades: desenvolvedor e professor-autor/ autor que interagem de maneiras distintas com o Sistema. O desenvolvedor trabalha diretamente na ferramenta de autoria e precisa conhecer ferramentas como a rede de Petri, enquanto que o autor faz esta interação através de uma interface amigável. Os novos papéis são os seguintes:

- Professor-autor:

- Estrutura o material didático particionando o domínio em subdomínios de acordo com a visão externa do modelo MATHEMA (seção 3.1.1) definindo o número de agentes tutores da SATA;

- Propõe para cada subdomínio, um conjunto de currículos especificando para cada um deles, conforme a visão interna, as unidades pedagógicas, os problemas associados e seus pré-requisitos

- Determina a estratégia e táticas pedagógicas, através da especificação das unidades de interação necessárias à resolução de cada problema.

- Professor-monitor/tutor:

  - Monitora o estudante, agindo como um facilitador na aquisição do conhecimento;

  - Monitora as dificuldades que os alunos manifestam, principalmente em relação a linguagem e estratégias inadequadas;

  - Atua no sentido de facilitar a interação dos alunos com a sociedade de agentes tutores artificiais.

  Em algumas situações os papéis do professor-autor e do professor-monitor podem ser exercidos por uma mesma pessoa.

- Desenvolvedor:

  - Especifica o modelo do estudante;

  - Especifica os controles para as estratégias pedagógicas baseadas em informações do modelo do aluno explorando a estrutura do domínio;

  - Promove a manutenção da SATA no que diz respeito aos processos de comunicação e interação.

- Estudante:

  - Fornece suas informações pessoais;

  - Utiliza o sistema tutor.

3.3.2 Componentes Computacionais:

Na arquitetura MATHEMA (figura 3.2) a Sociedade de Especialistas Humanos (SEH) interage com a SATA através do Agente de Manutenção, entretanto refinou-se esta arquitetura e o Agente de Manutenção passa a ser chamado de Agente Monitor e é através deste que o professor-monitor poderá se comunicar com a SATA e eventualmente poderá bloquear um determinado agente da SATA para um curso ou grupo de alunos. O Agente de Interface, a Interface e a SATA de maneira geral apresentam as mesmas funcionalidades definidas no MATHEMA. No entanto, foram realizadas modificações importantes como a:

1. definição de novos papéis para os agentes humanos, apresentados na seção 3.3.1;
2. criação de um Arcabouço para Sistemas Tutores Inteligentes (ASTI) com uma especificação interna do Sistema Tutor dos agentes da SATA;
3. introdução de uma Ferramenta de Autoria para Sistemas Tutores (FAST) juntamente com uma interface de Autoria para o professor-autor.

A seguir tem-se uma descrição sucinta das alterações realizadas:

- ASTI:

  Arcabouço para Sistemas Tutores Inteligentes constituído por uma Sociedade de Agentes Tutores Artificiais (SATA) e por uma Interface. A estrutura de cada subdomínio fornecida pela FAST é implementada na forma de um agente tutor que pertence a SATA.

- FAST:

  Ferramenta de Autoria para Sistemas Tutores com a finalidade de auxiliar o professor-autor na construção de um STI. A FAST fornece a partir das informações sobre o domínio inseridas pelo professor-autor (ver seção 3.1.1): (i) o modelo pedagógico – na forma de uma base de regras JESS – de cada agente tutor pertencente a SATA (a ser criado na ferramenta ASTI) que determina o comportamento de cada agente tutor na apresentação do curso; (ii) o modelo de domínio, sob a forma de páginas HTML. A estrutura do modelo do estudante é criada pelo desenvolvedor e as informações contidas neste modelo são usadas pelo modelo pedagógico para inferir o que apresentar ao estudante.

- Interface de Autoria:

  A interface de autoria permite que o professor-autor insira na FAST as informações, na forma de grafos, pertinentes ao conhecimento do domínio a ser apresentado ao aluno, além das estratégias e táticas pedagógicas relacionadas.

Apesar da SATA não ter sofrido grandes modificações o modelo do agente tutor (figura 3.4) está sendo estendido conforme a descrição a seguir:

- Sistema Tutor: constituído pelos modelos de domínio, pedagógico e do estudante fornecidos pela FAST que representam as interações entre o tutor e o estudante. O sistema faz parte agora de um arcabouço (ASTI) que permite desenvolver Sistemas Tutores Inteligentes nas mais diversas áreas do conhecimento. Tanto o modelo de autoria que deu origem a FAST quanto um protótipo parcial da ASTI foram desenvolvidos no âmbito desta tese;

- Sistema Social: abordado na tese em andamento de Eliane Pozzebon que trata da aprendizagem em grupo fornecendo um comportamento colaborativo mais desenvolvido entre os agentes [Pozzebon et al., 2006];
- Sistema de Distribuição: é feita através da plataforma de desenvolvimento de agentes JADE [Bellifemine et al., 2004] que possui os mecanismos de comunicação implementados de acordo com o padrão internacional FIPA (Foundation for Intelligent Physical Agents).

A Ferramenta de Autoria para Sistemas Tutores (FAST) é detalhada no capítulo 4 e o Arcabouço para Sistemas Tutores Inteligentes (ASTI) é apresentado no capítulo 5.

Capítulo 4

FAST: Ferramenta de Autoria para Sistemas Tutores

A busca de uma solução para o gerenciamento das interações entre o aluno e o tutor artificial, sem exigir do professor um treinamento específico para a criação de cursos deu origem a um modelo conceitual de autoria implementado em uma ferramenta chamada FAST. Este mecanismo visa a integração dos modelos de domínio, pedagógico e do estudante que resulte num STI com características realmente adaptativas sem acarretar novas atribuições para o professor no momento de criação do curso.

4.1 Concepção do Modelo de Autoria

As principais decisões de projeto, visando o compromisso entre a simplicidade no uso da ferramenta de autoria juntamente com um comportamento complexo e adaptativo dos STIs gerados, são:

(i) a representação explícita, usando ontologias [Mizoguchi e Bourdeau, 2000], do conhecimento que descreve os modelos do domínio e do estudante, incluindo a relação entre eles. As ontologias permitem também o reuso das informações;

(ii) o uso de um formalismo expressivo, *Redes de Petri a Objetos (RPO)* [Sibertin-Blanc, 1985], para especificar o modelo pedagógico do sistema, cujas transições controlam a tomada de decisão durante as interações de acordo com as condições estabelecidas pelo modelo do estudante e o retorno do aluno fornecido através das suas intervenções no STI;

(iii) a utilização de controle multi-nível para aumentar a flexibilidade do comportamento sem sacrificar a simplicidade na especificação, facilitando também a visualização de caminhos pedagógicos alternativos e a sincronização das ações.

As razões da escolha das ontologias e das RPO foi principalmente devido à expressividade destas ferramentas para representar conhecimento, permitindo a reutilização, manutenção e expansão.

As ontologias são adotadas com os seguintes objetivos:

- compartilhar as informações através da representação explícita do conhecimento que descreve os modelos do domínio e do estudante, incluindo a relação entre eles;
- facilitar a integração da informação distribuída;
- permitir que as informações armazenadas possam ser utilizadas como fonte de consulta do domínio;
- reutilizar o conhecimento representado nas ontologias.

A utilização de Redes de Petri a Objetos (RPO) para representar e implementar o modelo pedagógico e de domínio, permite efetuar uma adaptação utilizando o modelo do estudante, obtida diretamente no modelo através de uma estrutura de dados associada às fichas, para determinar o percurso de aprendizagem mais apropriado. Além disso, as RPO permitem a visualização de caminhos pedagógicos alternativos, sincronização de ações, quando exigido, e a repetição de eventos. Permite ainda, uma integração suave entre os diferentes níveis de controle proporcionando uma solução elegante para o suporte às interações simultâneas com vários estudantes trabalhando individualmente, usando fichas que contenham objetos que representam os estudantes.

Uma análise das qualidades estruturais das RPO pode ser usada para verificar possíveis problemas na etapa de construção do curso.

4.1.1 Representação do Modelo de Domínio

O modelo de domínio é formado pelas definições da visão interna do modelo conceitual MATHEMA. Um curso desenvolvido usando o modelo proposto é representado como uma instância do modelo de domínio e contém todas as informações fornecidas pelo professor e são de dois tipos: propriedades e conteúdos. Exemplos de propriedades são as relações de pré-requisitos, o nível de detalhe e de dificuldade, etc. Conteúdo é o que é apresentado ao estudante, usualmente a apresentação é feita através de páginas HTML.

Um fragmento da ontologia que representa o modelo de domínio é mostrado na figura 4.1 (* significa múltiplas instâncias). As classes `Curriculum`, `Pedagogical_Unit`, `Problem` e `Interaction_Unit` correspondem aos conceitos da visão interna do domínio no modelo MATHEMA. Classes `Prerequisite` e `Node` são exemplos de como as ontolologias podem ser utilizadas na

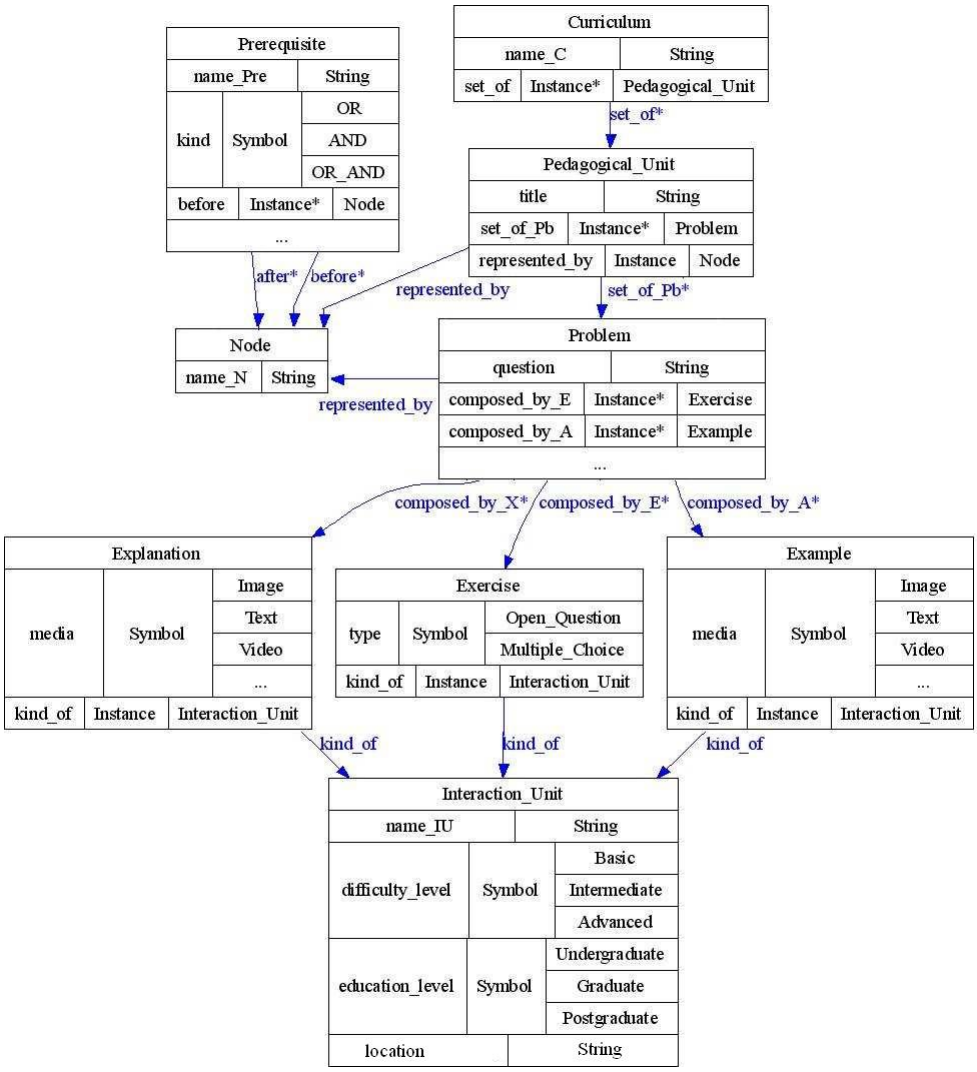


Figura 4.1: Ontologia do Modelo do Domínio

definição de grafos de pré-requisitos. Estes grafos são usados para definir as relações entre as unidades pedagógicas e entre os problemas. Os nós destes grafos são parcialmente ordenados de acordo com as relações de pré-requisito entre UP e entre problemas fornecidas pelo professor.

Para construir os grafos, o professor deve saber que cada nó *n* (relacionado com uma UP ou um Pb) deve ter o seguinte número de arcos de entrada:

Nenhum: o nó *n* não tem pré-requisitos e pode ser executado em qualquer tempo. Este é o caso de um nó inicial.

Um: o nó *n* tem apenas um nó como pré-requisito e este nó deve ser executado antes de *n* ser habilitado para execução.

Dois ou mais arcos necessários: o nó *n* tem vários nós como pré-requisitos e todos devem ser executados, em qualquer ordem, antes do nó *n* estar habilitado para execução.

Dois ou mais arcos alternativos: o nó *n* tem vários nós como pré-requisitos mas somente um deles deve ser executado antes do nó *n* estar habilitado para execução.

Arcos necessários e alternativos podem ocorrer simultaneamente num mesmo nó. Nós e diferentes tipos de arcos podem ser combinados em um grafo complexo, de acordo com a seqüência de curso pretendida. Entretanto, uma condição deve ser satisfeita: cada grafo deve ter somente um nó inicial e somente um nó final. Esta restrição parece ser bastante razoável pois, uma UP e um Pb devem ter um ponto inicial e final.

A classe `Interaction Unit` especifica as unidades de interação. As classes `Explanation`, `Example` e `Exercise` representam tipos específicos de unidades de interação, outras classes podem ser acrescentadas conforme a necessidade. Estas classes contêm o (slot `location`), que aponta para as páginas de interação, cujo conteúdo também é especificado pelo professor.

4.1.2 Representação do Modelo do Estudante

O modelo do estudante contém definições dos conceitos necessários para caracterizar um estudante e seu histórico de interações com o sistema. A construção deste modelo foi inspirada nos trabalhos de Chen e Mizoguchi [2004]. O modelo do estudante é baseado na abordagem do estereótipo que classifica os estudantes de acordo com o nível de seu conhecimento. As informações são obtidas através de um teste preliminar e da atualização, que é feita durante as novas interações, permitindo uma re-classificação caso esta se faça necessária.

O sistema permite ao estudante a visualização do seu perfil e das suas preferências. Além disso é possível, caso ele discorde da sua classificação de perfil ou estilo de aprendizado, modificar estas informações permitindo que ele reflita sobre sua performance. No entanto, o professor não pode mudar o modelo do estudante que contém informações dinâmicas compostas pelas descrições das atividades realizadas pelo estudante durante suas interações com o sistema. Os atributos básicos são: (a) `choice`, os problemas que ele pode fazer em uma determinada interação; (b) `doing_IU` / `doing_Pb`, as interações unidades/problema que ele está fazendo; (c) `done`, uma lista que contém os relatórios de todos os problemas e as unidades de interação que o estudante fez, com a data em que ele realizou esta atividade e, no caso dos exercícios, também lhe é atribuída uma nota.

É importante ressaltar que o STI criado é um sistema multiagente, estando modelo do estudante distribuído entre os agentes. Os detalhes das interações entre o estudante e um determinado Agente Tutor (AT) são armazenados *localmente* e somente a lista de Unidades Pedagógicas que o estudante já realizou é compartilhada como *conhecimento social*.

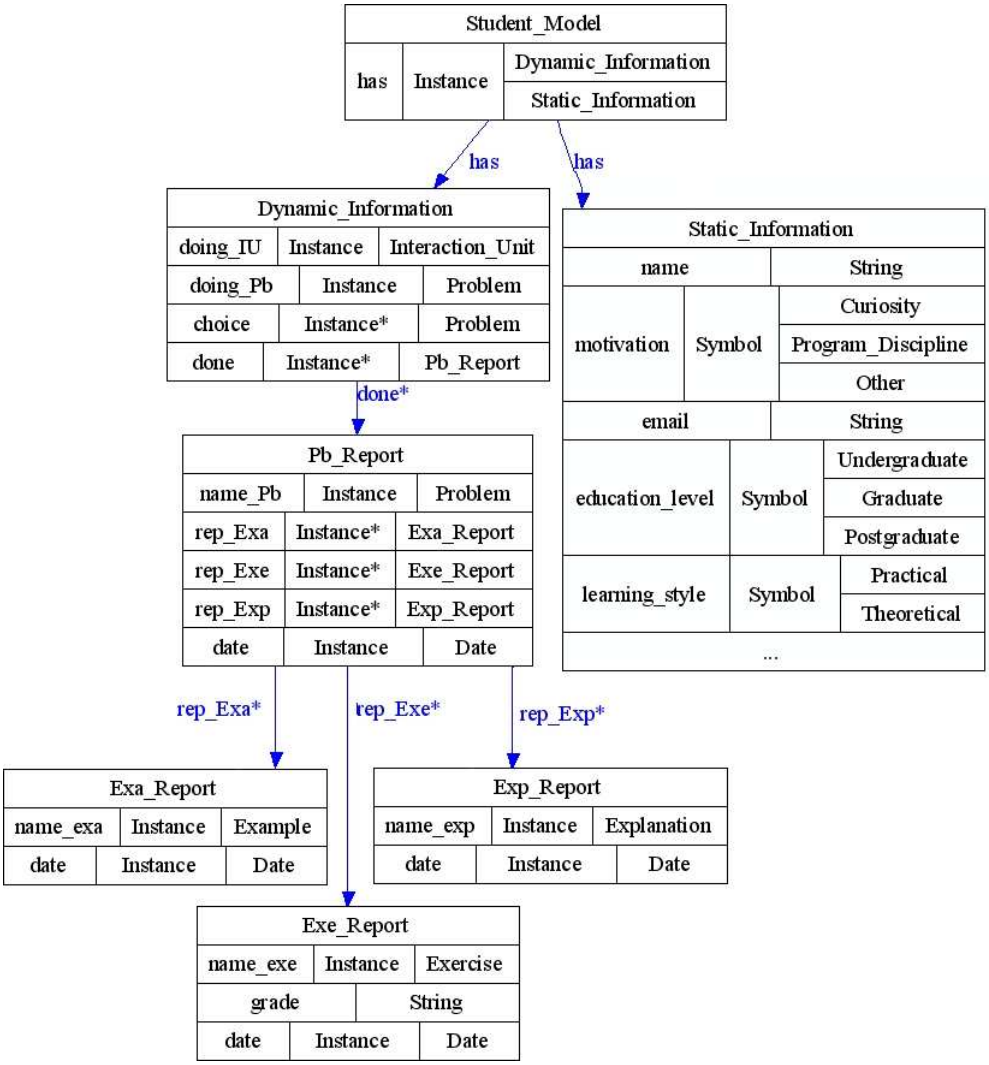


Figura 4.2: Ontologia do Modelo do Estudante

4.1.3 Representação do Modelo Pedagógico

O modelo pedagógico, também chamado de instrucional ou tutor, de um Sistema Tutor Inteligente (STI) utiliza as informações do modelo do estudante para determinar como o domínio do conhecimento deve ser apresentado ao estudante.

O modelo pedagógico gerencia as estratégias que controlam as seqüências do conteúdo, tomando as decisões cabíveis de acordo com as necessidades individuais de cada aluno, indicando qual tópico deve ser apresentado e em qual momento.

O desenvolvedor é o responsável por criar uma representação do modelo pedagógico que seja suficientemente abrangente de forma a atender as estratégias utilizadas pelo professor. Caso seja necessário, o desenvolvedor pode criar uma biblioteca de representações que atenda as solicitações do professor. No modelo de autoria esta representação é feita utilizando RPO.

4.2 Mecanismo de Integração

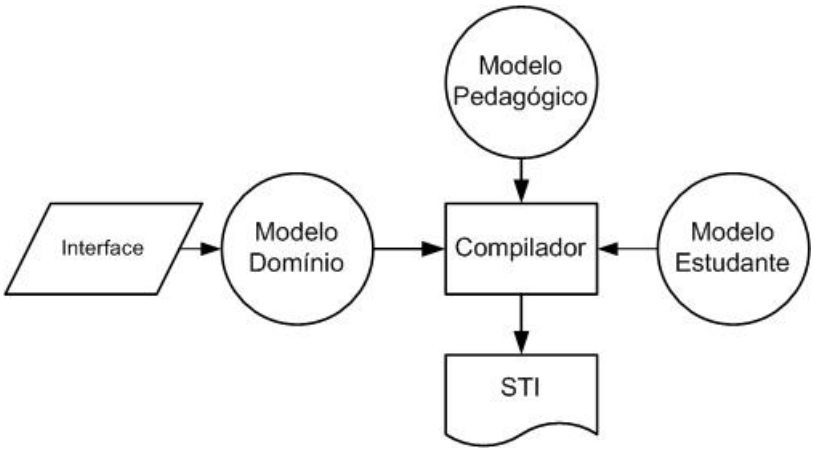


Figura 4.3: Modelo Conceitual da FAST

A busca de um mecanismo que permitisse a integração dos modelos de domínio, pedagógico e do estudante fornecendo um STI com características adaptativas resultou em um modelo conceitual (figura 4.3) concretizado na ferramenta da autoria FAST [Frigo et al., 2007], apresentada a seguir.

Este modelo conceitual é formado pelos seguintes ítens:

- Uma interface interativa onde o professor adiciona o conteúdo do curso de acordo com a estrutura do modelo do domínio (ver seção 4.1.1). Os pré-requisitos associados às UP e aos problemas são definidos na forma de grafos.
- Um compilador que transforma a definição do modelo do domínio, vista como uma instância da respectiva ontologia, em uma RPO que implementa a semântica do curso. O uso e a atualização das informações do modelo do estudante são automaticamente introduzidos na RPO. A RPO resultante é usada para controlar o modelo pedagógico da SATA.

Para aumentar a flexibilidade dos comportamentos possíveis do modelo pedagógico, sem sacrificar a simplicidade na especificação, o modelo utiliza um controle de dois níveis: o nível do *Currículo* e o nível das *Estratégias*.

O nível do Currículo especifica os percursos possíveis do curso de acordo com suas Unidades Pedagógicas (UP) e problemas (PB), respeitando os pré-requisitos definidos pelo professor, e o nível das Estratégias especifica a interação entre estudante e sistema tutor durante a resolução de um problema específico. Ambos os níveis são representados por RPO organizadas de forma hierárquica, conforme detalhado nas seções seguintes.

4.2.1 Nível do Currículo (RPO-CV)

O nível do Currículo é formado por uma rede de Petri a Objetos, chamada de rede *RPO-CV*, que especifica as estruturas de controle necessárias para a navegação nos problemas. As decisões de navegação são baseadas nas informações armazenadas do modelo do estudante e no retorno que o estudante fornece ao utilizar o sistema durante a sua interação.

A rede RPO-CV é gerada automaticamente pelo compilador da figura 4.3 à partir dos dois grafos de pré-requisitos que constituem o modelo do domínio, ambos definidos pelo professor (ver seção 4.1.1):

- grafo Gr_UP, que define as relações entre as k unidades pedagógicas, como representado na figura 4.4.a, onde $k = 6$ (unidades $UP1$ à $UP6$);
- grafo Gr_Pb, que define as relações entre os n problemas de uma unidade pedagógica, como representado na figura 4.4.b, que mostra a relação entre os 5 problemas que definem a $UP2$.

Cada uma das demais UP da figura 4.4.a deve ser refinada num grafo definindo as relações entre os seus problemas como o da figura 4.4.b.

O compilador combina o grafo Gr_UP e os k grafos Gr_Pb referentes às unidades pedagógicas de Gr_UP em único grafo Gr_CV, substituindo cada nó UP_i do grafo Gr_UP pelo grafo Gr_Pb$_i$ correspondente.

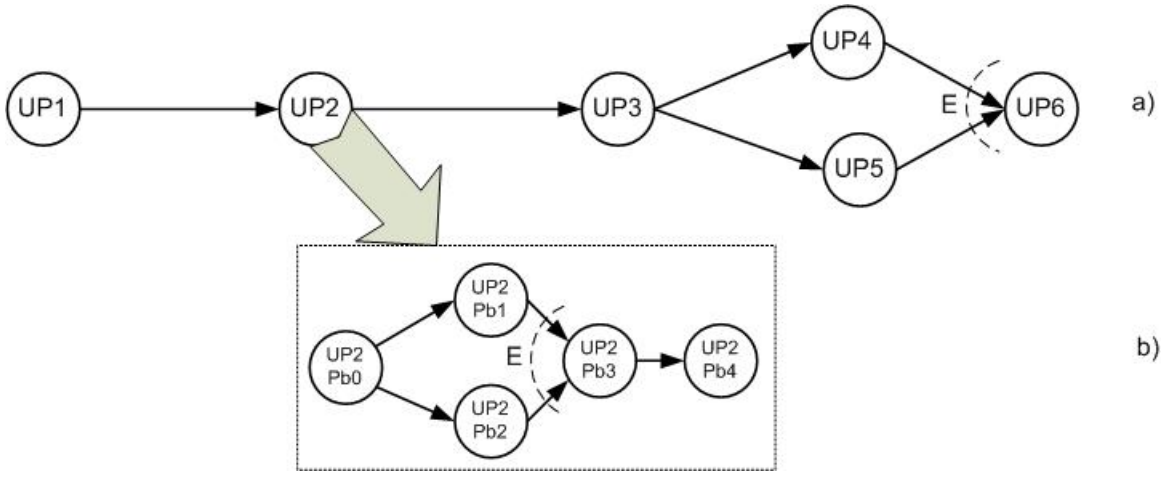


Figura 4.4: a) Gr_UP b) Gr_Pb

A RPO-CV é construída automaticamente à partir do grafo *Gr_CV* em três etapas:

Etapa 1 traduzir o grafo *Gr_CV* em uma rede de Petri RdP_CV,

Etapa 2 transformar a rede de Petri RdP_CV em uma rede de Petri a objetos RPO_CV de modo à considerar diretamente o modelo do aluno,

Etapa 3 adicionar *lugares de comunicação* para permitir a interação com o nível das *Estratégias*.

As duas primeiras etapas são explicadas a seguir e a terceira etapa será explicada na seção 4.3.1.

Etapa 1

A tradução do grafo *Gr_CV* em uma rede de Petri RdP_CV que implementa a política de pré-requisitos entre os problemas de todas as unidades pedagógicas é feita de acordo com as seguintes regras:

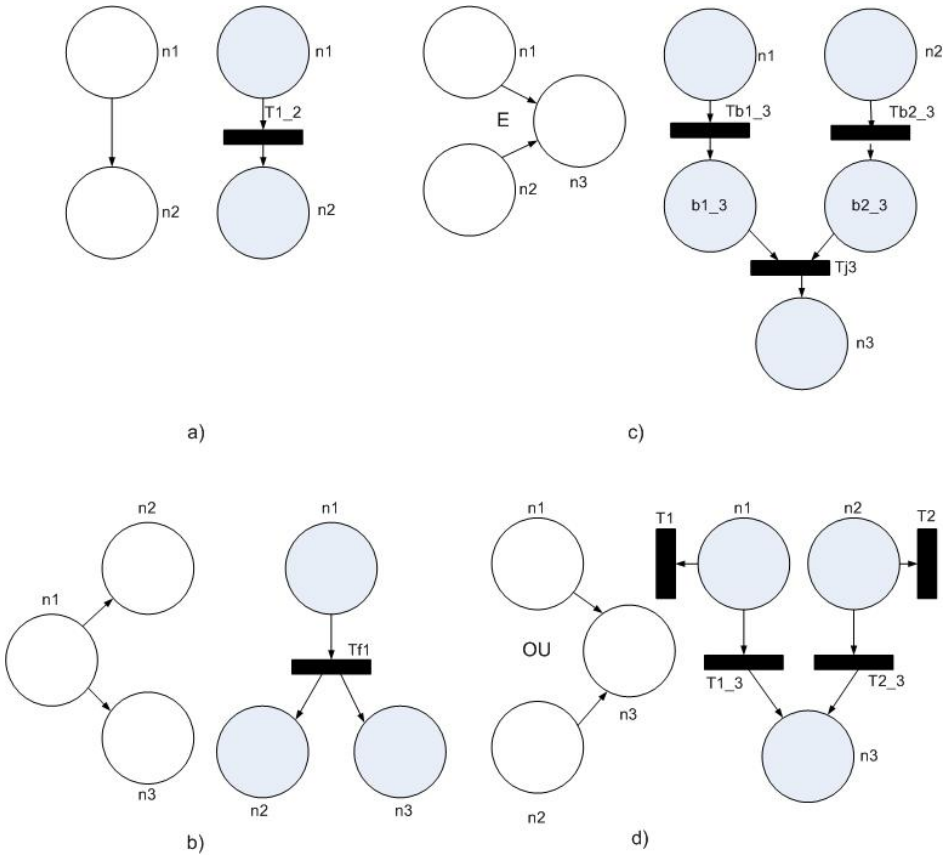


Figura 4.5: Topologias de Grafos e Redes de Petri

- Uma seqüência simples entre um nó `n1` e um nó `n2` é implementada por uma transição `T1_2`, um lugar de entrada `n1` e um lugar de saída `n2` (ver figura 4.5.a).
- Um nó com dois arcos de saída é implementado por uma transição *fork* `Tf_1` com um lugar de entrada `n1` e dois lugares de saída, `n2` e `n3`. Depois que a atividade associada com o lugar de entrada está terminada, o estudante pode escolher a atividade associada com um ou outro lugar de saída (ver figura 4.5.b).

- Um nó com dois arcos de entrada necessários é implementado por uma transição *join* `Tj_3` com dois lugares de entrada, `b1_3` e `b2_3`, e um lugar de saída, `n3`. Os lugares `b1_3` e `b2_3` são lugares *buffers* e não fazem correspondência com UPs ou problemas. Eles são necessários pois após a atividade associada ao lugar `n1` ou `n2` for terminada, a informação deve ser armazenada e somente quando a atividade associada com o outro lugar de saída estiver terminada a atividade associada com o lugar de saída `n3` pode ser realizada. Para isto, duas transições extras são necessárias, `Tb1_3` e `Tb2_3`, conectando, respectivamente, os lugares de entrada `n1` e `n2` aos *buffers* `b1_3` e `b2_3` (ver figura 4.5.c).

- Um nó com dois arcos de entrada alternativos é representado por duas transições `T1_3` e `T2_3`, com lugares de entrada `n1` e `n2`, respectivamente, e com lugar de saída `n3`. Transições `T1` e `T2` são usadas para remover a ficha do aprendiz dos lugares `n1` e `n2`, respectivamente, no caso das atividades associadas a `n3` já terem sido realizadas, evitando deste modo a repetição de `n3` (ver figura 4.5.d).

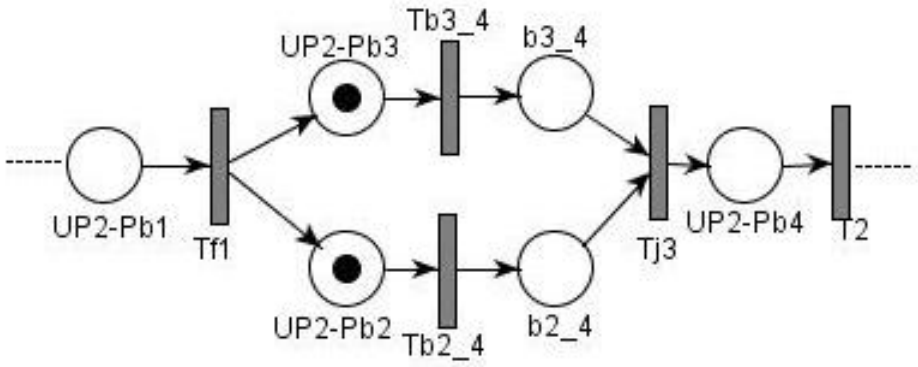


Figura 4.6: Fragmento da rede de Petri RP-CV gerada à partir da figura 4.4

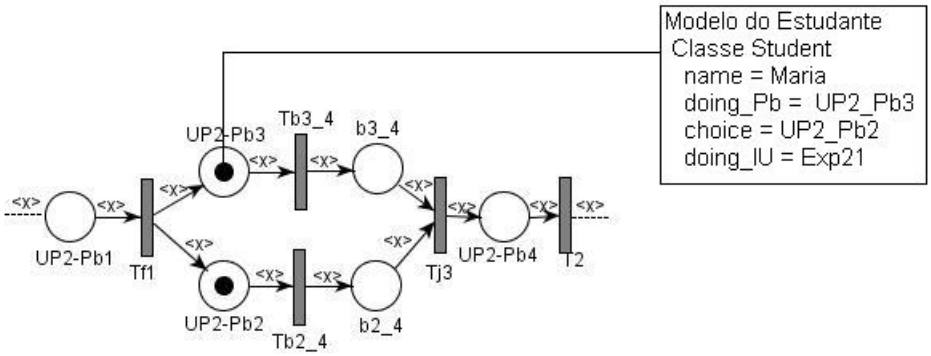
A figura 4.6 mostra a RP-CV obtida a partir dos grafos da figura 4.4. A marcação do lugar p, m(p), indica o estado parcial desta rede. Nesta etapa a ficha não é um objeto, e indica somente que um estudante interagindo com o sistema **pode** fazer o problema Pb_j da unidade pedagógica UP_i. Neste exemplo, o estudante pode realizar UP_2-Pb_2 e UP_2-Pb_3.

Etapa 2

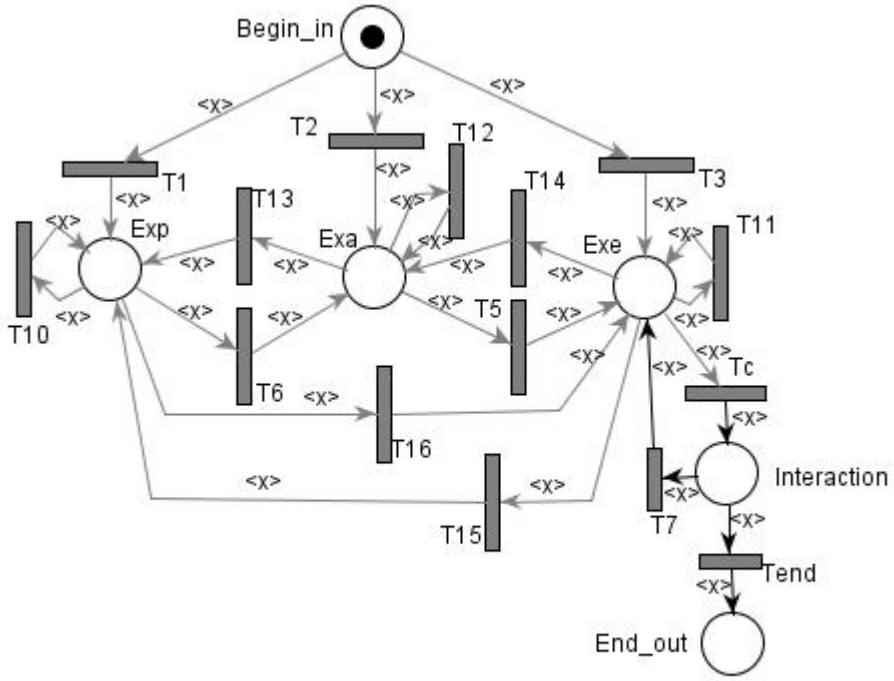
Esta etapa consiste em transformar a rede de Petri RdP_CV em uma rede de Petri a objetos RPO_CV de modo à considerar diretamente o modelo do aluno. A RPO_CV, sendo uma rede de Petri a objetos, é constituída de:

- uma estrutura de dados definida a partir de classes de objetos associada às fichas dado pelo modelo do estudante;

- uma estrutura de controle chamada de rede subjacente dada pela rede RP-CV obtida à partir dos grafos na etapa 1. É ainda associado à cada arco da rede uma variável *x* de forma a poder instancia a ficha associada ao modelo de um determinado estudante.



a)



b)

Figura 4.7: a) RPO-CV b) RPO-E

A rede RPO-CV permite considerar agora a identidade de cada estudante representado pela ficha pois a variável *x* poderá instanciar uma das diversas fichas contidas num lugar como será explicado mais em detalhe na seção 4.3.2.

Todos os lugares e fichas possuem a mesma classe `Student` e as transições têm pré-condições que se referem aos atributos da ficha e/ou as variáveis externas associadas as ações do estudante. A figura 4.7.a representa um fragmento da RPO-CV obtida a partir da figura 4.6.

Se um estudante representado pela instância `Maria` (figura 4.7.a) está resolvendo o problema p então m(p)=`Maria`. Uma ficha `Maria` no lugar UP$_i$-Pb$_j$ da RPO-CV significa que o estudante **pode** fazer o problema Pb$_j$ da unidade pedagógica UP$_i$. O problema que ele está realmente fazendo neste instante é definido pelo atributo `doing`. Considerando a figura 4.7.a: após o disparo de **tf1**, `Maria.choice`= (UP_2-Pb_2, UP_2-Pb_3). Se Maria escolhe fazer UP_2-Pb_3, tem-se `Maria.doing`=(UP_2-Pb_3) and `Maria.choice`=(UP_2-Pb_2), significa que neste ponto da interação ela pode escolher somente fazer o problema Pb_2. Cada vez que um problema possui como valor do atributo `doing`, a ficha `Maria` associada é colocada no lugar `Begin`$_{in}$ da RPO que representa o nível das estratégias (veja na próxima sessão a figura 4.7.b).

Apesar da estrutura de transições da RPO-CV estar restrita aos pré-requisitos definidos pelo professor, outras decisões relacionadas à navegação podem apresentar um certo grau de liberdade. Algumas destas decisões devem ser tomadas pelo estudante através das opções disponíveis na interface: interromper uma sessão, revisar um problema anterior, editar o seu perfil, comunicar-se com o professor ou outros estudantes, etc. Alguns atributos dos estudantes (veja figura 4.2) usados neste nível são: `name`, `doing_IU`, `doing_Pb`, `choice` e `done`.

4.2.2 Nível das Estratégias (RPO-E)

O nível das estratégias é também implementado por uma RPO, chamada de *RPO-E*, no entanto esta rede não é definida pelo professor e sim pelo desenvolvedor. Esta rede controla as interações na solução de problemas, levando em consideração o retorno e os atributos do modelo do estudante. O objetivo da RPO-E é apresentar ao estudante as unidades de interações associadas com um determinado problema.

No nível das estratégias, os tipos de interações com o estudante podem ser muito mais flexíveis e o objetivo neste nível é unir um conjunto pré-definido de unidades de interação com os protocolos de interação. As unidade de interação podem ser de diferentes tipos, por exemplo, elas podem usar diferentes tipos de mídias ou ainda requerer diferentes tipos de respostas do estudante. A combinação destas unidades com os protocolos de interação que implementam um cenário de aprendizagem é de responsabilidade dos desenvolvedores do sistema.

Cada cenário de aprendizagem no nível das estratégias consiste em adequar extensões às definições da classe `Problem` na figura 4.1. Estas extensões devem incluir as definições da classe `Interaction_Unit` e *slots* `Location` associados com os conteúdos fornecidos pelo professor.

Possíveis extensões da classe `Interaction_Unit` são as classes `Explanations`, `Examples` e `Exercises`, que contém, respectivamente, explicações, exemplos e exercícios. O conteúdo de instâncias destas classes são fornecidos pelo professor. Cada conteúdo deve ter especificado seu nível de dificuldade e eventualmente o público ao qual se destina.

Um exemplo de cenário para o nível das estratégias é representado pela RPO-E da figura 4.7.b. Neste cenário uma RPO apresenta explicações, exemplos e exercícios em uma ordem e em nível de detalhe determinados pelas informações do modelo do estudante e pelo retorno por ele fornecido. Este cenário é relativamente simples, mas é complexo o suficiente para mostrar como o conteúdo do modelo de domínio fornecido pelo professor é integrado ao modelo do estudante. Claramente, as RPO são suficientemente expressivas para permitirem a definição de cenários de aprendizagem que incluem protocolos de interações mais complexos com os estudantes.

O lugar **Begin**$_{in}$ (figura 4.7.b) está associado com a unidade de interação introdutória onde, as questões centrais associadas com o problema são mostradas. Os atributos do modelo do estudante usados neste nível de interação são, por exemplo, `education_level, preferred_media`, etc.

Os lugares **Exp**, **Exa**, **Exe** estão associados com as unidades de interação que apresentam ao estudante, respectivamente, explicações, exemplos e exercícios. Cada vez que uma ficha é colocada em um destes lugares, o conteúdo correspondente é apresentado ao estudante. A atividade é diferente de acordo com as interações e com os resultados destas. Todas estas informações são automaticamente armazenadas no modelo do estudante.

A transição t_c (figura 4.7.b) é um exemplo de como uma decisão que depende do retorno do estudante pode ser implementada na RPO e representa uma solicitação feita pelo estudante ao sistema responsável pela correção do exercício. Esta condição associada é `Request = true` (retorno do estudante). A solicitação é representada pela flecha saindo de t_c. Depois disso, se o estudante obtém a nota mínima, a transição t_{end} é disparada e ela está apta a continuar suas interações. Caso contrário, a transição t_7 é disparada e ela deve refazer o exercício.

As transições remanescentes podem ser divididas em três grupos (ver figura 4.8): T_p (T1, T10, T13 and T15), T_a (T2, T6, T12 and T14) and T_e (T3, T5, T11 and T16). Elas referem-se, respectivamente, aos lugares Exp, Exa e Exe e são disparadas de acordo com as condições contidas no modelo do estudante. Mesmo neste exemplo simples é possível verificar como as transições podem gerar um grande número de diferentes comportamentos, dependendo da ordem e do número de vezes que um lugar é visitado.

Os grupos T_p, T_a e T_e têm condições associadas do tipo, respectivamente, `isExp(x)=True`, `isExa(x)=True` e `isExe(x)=True`. As ações relacionadas com todas estas transições é `doing_IU(x) = next(doing_IU(x))`. Já a função *next* depende do modelo do estudante x, como explicado anteriormente e é responsável pela interação com o estudante.

4.3 Controle das Interações

As RPO que representam os níveis do Currículo e das Estratégias têm uma relação hierárquica: uma ficha depositada em um lugar da RPO-CV, quando da execução de uma de suas transições,

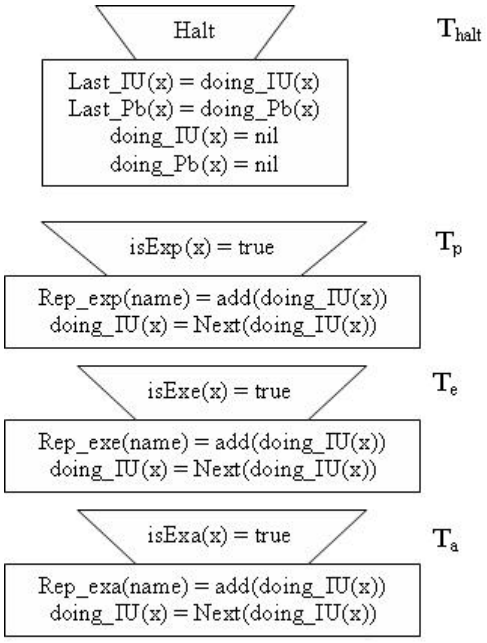


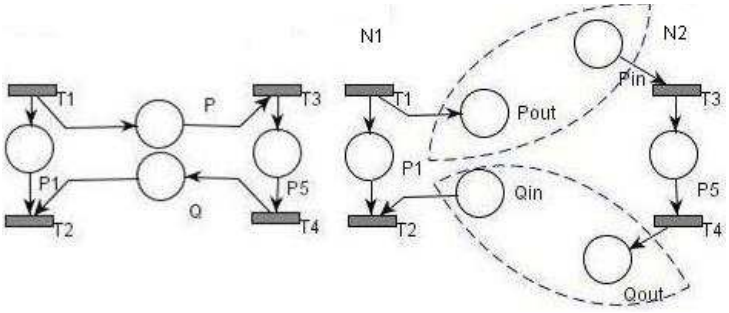
Figura 4.8: Legenda das Transições

é transferida automaticamente para o lugar inicial da RPO-E associada ao problema em questão. Após a execução completa da RPO-E, a ficha retorna ao lugar da RPO-CV e sua execução pode prosseguir. Esta sessão descreve como o formalismo de RPO hierárquicas é usado para controlar as interações entre o estudante e o sistema.

4.3.1 Controle multi-níveis

A RPO-CV indica qual problema Pb_i de uma Unidade Pedagógica UP_j é apresentado ao estudante, e a rede RPO-E controla as atividades de resolução deste problema, decidindo que tipo de conteúdo e em que ordem ele deve ser apresentado. O mecanismo que permite a troca de informações entre estas RdPs está ilustrado na figura 4.9: a figura 4.9.a mostra a rede completa e a figura 4.9.b mostra esta rede dividida em duas redes, N1 e N2. A rede N1 corresponde à rede RPO-CV e N2 a rede RPO-E. O lugar p na figura 4.9.a está separado em dois lugares na figura 4.9.b: p_{out} em N1, e p_{in} em N2, chamados *lugares de comunicação*. O mesmo processo é feito para o lugar q. Quando uma transição t_1 é disparada na rede completa da figura 4.9.a, os lugares p_1 e p estão marcados, e então a transição t_3 está habilitada. Nas redes correspondentes N1 e N2 da figura 4.9.b, após o disparo de t_1 os lugares p_1, p_{out} e p_{in} são marcados, e t_3 está habilitada. Logo a rede completa e as redes divididas são equivalentes, e este mecanismo de passagem de ficha permite implementar uma rede em um ambiente distribuído ou em um contexto de RdPs hierárquicas. A ficha colocada em p_{out} representa uma mensagem enviada para um agente tutor/interface e uma ficha em p_{in} representa uma mensagem recebida. Uma ficha em p_1 significa que uma atividade associada foi iniciada.

Após o disparo da transição t_3 em N2, p_5 é marcada e t_4 pode ser disparada: uma ficha é colocada em q_{out} e q_{in}. A partir deste momento t_2 em N1 pode ser disparada e N1 não pode evoluir sem a resposta de N2.



a) b)

Figura 4.9: Comunicação entre duas redes

A rede de Petri RPO-CV apresentada na figura 4.7.a é fornecida pelo compilador conforme descrito na seção 4.3.1 e a rede RdP-E é fornecida pelo desenvolvedor. A execução destas duas redes é realizada pelo mecanismo mostrado na figura 4.9. Portanto, é necessário realizar uma terceira etapa, de modo a permitir a comunicação por passagem de mensagem como representado na figura 4.9, que consiste a representar na rede de Petri todos os lugares de comunicação.

Etapa 3

De modo a permitir a comunicação entre os níveis de Currículo e o nível das Estratégias, adicionar à rede RPO-CV:

- um lugar de comunicação de saída Begin-out a cada transição de entrada de um problema $Pb_i - UP_j$,
- um lugar de comunicação de entrada Begin-in a cada transição de saída de um problema $Pb_i - UP_j$.

Assim, a rede RPO-CV que será realmente executada, obtida após a etapa 3 (a partir da rede representada na figura 4.7.a) é a rede apresentada na figura 4.10.

4.3.2 Suporte às Interações Simultâneas

O uso dos dois níveis de controle também distribui o carregamento do sistema quando diversos estudantes estiverem acessando o sistema simultaneamente.

Quando um estudante **S** inicia um Currículo, uma ficha **S** correspondente a este estudante é criada na rede RPO-CV. Cada vez que este estudante escolhe um problema na unidade pedagógica, esta ficha é **enviada** para a rede RPO-E. Esta rede é compartilhada com todos os estudantes que resolvem problemas. Este currículo mostra como o professor modelou a seqüência dos problemas: o estudante inicia com Pb_1 depois disto ele pode escolher Pb_2 ou Pb_3 mas ele deve resolver ambos para poder prosseguir e estar hábil a resolver o problema Pb_4 e assim por diante.

As pré-condições em uma RdP são similares aos de um sistema de regras. Sendo assim, T5 (figura 4.10) dispara se as pré-condições $PU_1 - Pb_2\_d(x)$ e $PU_1 - Pb_3\_d(x)$ e $End_{in}(x+x)$ são verdadeiras. Lembrar que na expressão $(x+x)$ o sinal $+$ é uma adição e não um OU lógico. A expressão $(x+x)$ significa que para a condição $End_{in}(x+x)$ ser verdadeira, as duas fichas associadas ao mesmo estudante devem estar disponíveis num mesmo lugar. Em uma RdP, quando uma transição dispara, as fichas são retiradas do lugar de entrada.

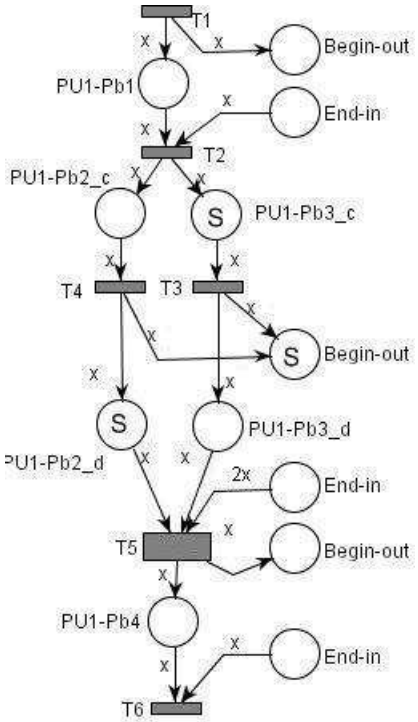


Figura 4.10: Nível Superior - envio de mensagens

4.4 Exemplos do Mecanismo de Integração

4.4.1 Caso 1: um único estudante

Na figura 4.11 nota-se que todas as fichas instanciam um mesmo objeto, a estudante `Maria`. Cada vez que uma transição dispara, os atributos do objeto instanciado são atualizados. Por

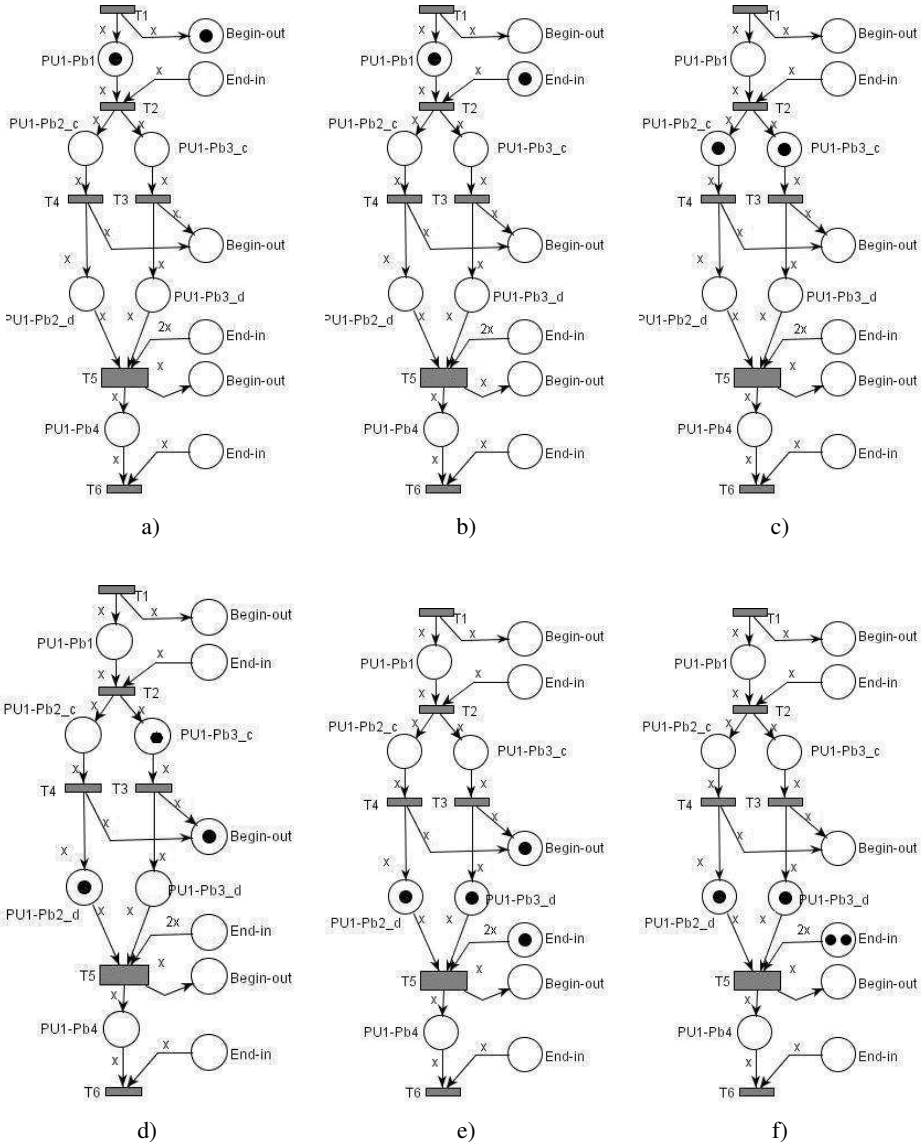


Figura 4.11: Evolução em RPO-CV

exemplo, quando t_1 dispara as ações associadas com $x.doing = Pb_1$ são executadas (x é uma instância de um estudante).

Maria, ao se conectar com o sistema tutor escolhe fazer o currículo C1.

- Após o disparo de t_1, os lugares PU_1-Pb_1 e $Begin_{out}$ estão marcados como mostra a figura 4.11.a e a ficha $x = Maria$ é colocada no lugar $Begin_{in}$ da rede RPO-E (figura 4.7.b) que é responsável pela execução das unidades de interação (UI) do problema PU_1-Pb_1.
- Assumindo que Maria fez alguma UI (podendo ser a seguinte seqüência: um exemplo, uma explicação, outro exemplo e por fim um exercício = Exa;Exp;Exa;Exe), disparando as seguintes transições (figura 4.7.b): $t_2 \rightarrow t_{13} \rightarrow t_6 \rightarrow t_5$.
- Assim que ela termina o exercício, ela solicita a correção do mesmo (t_c dispara).

- Caso ela tenha obtido sucesso na resolução do exercício, t_{end} da rede RPO-E dispara; uma ficha é colocada em End_{out} e também em End_{in} na rede RPO-CV (figura 4.11.b): isso significa que ela concluiu com sucesso este problema e t_2 está habilitada.
- Se Maria deseja continuar a usar o sistema (veja sessão 4.2.1) então t_2 dispara, resultando na marcação da figura 4.11.c e isto permite que neste momento ela possa escolher entre Pb_2 ou Pb_3 (`Maria.choice`= $\{PU_1\text{-}Pb_2, PU_1\text{-}Pb_3\}$).
- Supondo que a escolha de Maria tenha sido Pb_2: t_4 da rede RPO-CV dispara, figura 4.11.c, apresentando a nova marcação (estado) mostrada na figura 4.11.d.
- Após terminar este problema com sucesso (as interações foram controladas pela rede RPO-E na figura 4.7.b e End_{in} também está marcada agora), o tutor apresenta o problema Pb_3 como a única opção de escolha neste momento.
- Se ela deseja continuar, a transição t_3 dispara apresentando uma nova marcação como mostra a figura 4.11.e.
- Após ter feito Pb3 (t_{end} foi disparada na rede RPO-E), o estado resultante na rede RPO-CV pode ser visto na figura 4.11.f.
- A transição t_5 na rede RPO-CV é habilitada pela marcação porque as pré-condições associadas a ela são verdadeiras: uma ficha $x = Maria$ aponta para os lugares Pb_2 e Pb_3 e duas vezes para End_{in}.

4.4.2 Caso 2: mais de um estudante

A marcação da rede RPO-CV representada na figura 4.12.a, onde as fichas M e J representam respectivamente os estudantes Maria e José, descrevem a seguinte situação:

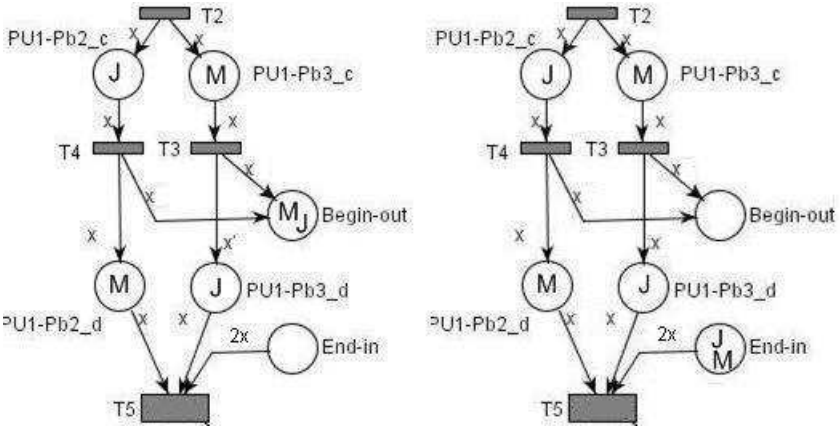


Figura 4.12: RPO-CV: Mais de um estudante

- A ficha M aponta para 3 lugares: i) $PU_1\text{-}Pb_3$-c, significando que Maria pode fazer Pb_3 (Maria.choice = $\{PU_1\text{-}Pb_3\}$); ii) $PU_1\text{-}Pb_2$-d, significando que Maria pode fazer Pb_2;

iii) $Begin_{out}$ significando que Maria pode iniciar a UI de Pb_2 (controlada pela rede RPO-E na figura 4.7.b).

- A ficha J também aponta para 3 lugares i) PU_1-Pb_2-c, significando que José pode fazer Pb2 (Jose.choice = $\{PU_1\text{-}Pb_2\}$); ii) PU_1-Pb_3-d significando que José está fazendo Pb_3; iii) $Begin_{out}$, significando que José pode iniciar a UI de Pb_3 (controlada pela rede RPO-E na figura 4.7.b).

Como explicada na seção 4.3.1, quando uma ficha é colocada no lugar $Begin_{out}$ na rede RPO-CV (figura 4.12.a) ela também aparece no lugar $Begin_{in}$ na rede RPO-E (figura 4.7.b). Então, neste cenário, o lugar $Begin_{in}$ possui as fichas J e M. Após algumas interações e solicitações de correção do exercício (t_c dispara como visto na figura 4.7.b), Maria (e Jose respectivamente) obtiveram sucesso em Pb_2 (e também em Pb_3). A transição t_{end} dispara e as fichas M e J são colocadas de volta no lugar End_{in} como mostra a marcação da figura. 4.12.b.

Maria e José terminaram os problemas Pb_2 e Pb_3 respectivamente, mas nenhum deles fez os dois problemas. Sendo assim, a transição t_5 (figura 4.12.b) não pode ser disparada porque suas pré-condições são falsas.

4.5 Aspectos de Implementação

A prática das implementações anteriores direcionou o trabalho para a busca de soluções aos principais problemas encontrados durante o processo de desenvolvimento de um arcabouço para Sistemas Tutores Inteligentes. Levando em consideração o grande esforço necessário para a construção do MathTutor ficou evidente, devido a complexidade do processo e a estreita relação do conteúdo com a implementação, porque a maioria dos STI não são reutilizáveis. Com o intuito de tornar um STI reutilizável no que tange ao reaproveitamento da estrutura do sistema e do conteúdo surge a idéia de desenvolver uma ferramenta de autoria cujos passos iniciais foram dados no trabalho de Postal [2004].

Outro problema encontrado na maioria dos sistemas de ensino-aprendizagem está relacionado a questão da adaptação, como fornecer ao estudante um sistema que tenha a capacidade de apresentar um conteúdo direcionado a um determinado estudante individualmente ou a uma categoria deles sem que o professor necessite explicitar como devem ser feitas estas interações.

Após diversas tentativas em minimizar estes problemas, chegou-se ao modelo de autoria proposto neste capítulo e que se concretizou na forma de uma ferramenta de autoria (FAST) que será descrita a seguir.

As ferramentas de suporte à implementação são apresentadas no Apêndice A (mais detalhes de implementação podem ser vistos em [Yamane, 2006]).

A figura 4.13 mostra as etapas da FAST correspondentes à obtenção de regras JESS a partir dos grafos de pré-requisitos descritos pelo professor e da rede RPO-E fornecida pelo desenvolvedor. Os elementos são os que seguem:

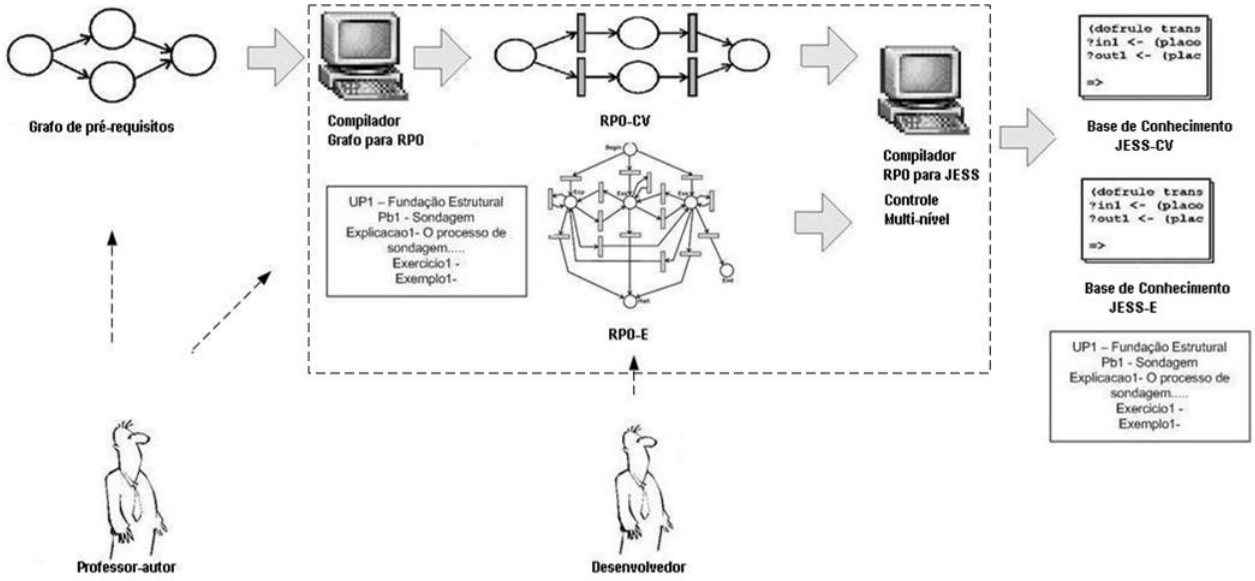


Figura 4.13: Funcionamento da FAST

Professor-autor: O autor deve modelar o domínio conforme a especificação da FAST (de acordo com a definição do MATHEMA). Feita a modelagem, ele utiliza a interface de autoria para inserir os grafos de pré-requisitos das Unidades Pedagógicas (UPs) e na seqüência os grafos de pré-requisitos dos Problemas (Pbs). Atualmente, os grafos são inseridos na forma textual. Para cada Problema são fornecidos os conteúdos necessários a resolução dos mesmos. O tipo de informação fornecida (exemplos, exercícios, explicações) está relacionada com as estratégias pedagógicas utilizadas pelo professor;

Desenvolvedor: As redes de Petri do nível das estratégias (RPO-E) são criadas diretamente pelo desenvolvedor da ferramenta e devem ser suficientemente genéricas para permitir múltiplas estratégias;

FAST:
1. O compilador utiliza as informações dos grafos (UPs e Pbs) juntamente com as informações conceituais do modelo do estudante e monta as RPOs do nível do currículo (RPO-CV), seguindo as etapas descritas nas seções 4.2.1 e 4.3.1;
2. Para que as RPOs sejam executadas, elas são transformadas em regras são interpretadas pelo motor de inferência de um Sistema Especialista (JESS).

Saída: A saída da FAST fornece os conjuntos de regras JESS que serão utilizadas para a construção de um STI específico através do Arcabouço para Sistemas Tutores Inteligentes (ASTI), assim como as páginas HTML com o conteúdo dos problemas fornecido pelo professor.

As duas etapas da FAST são detalhadas a seguir.

4.5.1 Compilador Grafo - Rede de Petri a Objetos

O compilador (figura 4.13) que transforma grafo em Redes de Petri recebe como entrada um grafo na forma textual. Na representação descrita a seguir os nós representam os problemas da figura 4.4. A notação usada para descrever os grafos da seção 4.2.1 é a seguinte:

Código 4.1 Grafo

```
PREREQGRAPH
UP2Pb1 := [ UP2Pb0 ] ,
UP2Pb2 := [UP2Pb0 ] ,
UP2Pb3 := [ (UP2Pb1,UP2Pb2) ] ,
UP2Pb4 := [UP2Pb3 ]
ENDGRAPH
```

O Código 4.1 mostra que se:

- um nó n possui dois ou mais arcos de entrada necessários tem-se $n := [(n_1, n_2, ...)]$ como UP2Pb3;
- um nó n possui dois ou mais arcos de entrada alternativos tem-se $n := [(n_1), (n_2), ...)]$.

A partir do grafo de pré-requisitos o compilador gera uma Rede de Petri também na forma textual.

As fichas são formadas por objetos de dados que apontam para o modelo do estudante e para o modelo de domínio. No Código 4.2 apresentam-se as informações dinâmicas (figura 4.2) e que fazem parte do conhecimento local do agente. As informações estáticas pertencem ao conhecimento social dos agentes e são obtidas a partir de formulários e questionários a serem preenchidos pelo próprio estudante. O campo *answer* contém a informação da interação do estudante com a interface que pode ser do tipo: *halt* indicando que ele quer se desconectar do sistema ou ainda *exercise, example ou explanation* que indica que ele gostaria de realizar alguma destas unidades de interação.

Os lugares (Código 4.3) correspondem aos problemas que constituem as Unidades Pedagógicas. O lugar *GhostPlace* é usado para enviar as fichas dos estudantes que já terminaram a execução na rede, mas que continuam num lugar da rede. Por exemplo, depois de passar por um OR, uma ficha pode ter ido parar no final, mas outra pode não ter sido disparada, e ter continuado num lugar antes do OR. Os lugares que iniciam o nome com buf representam *buffers* do tipo apresentados na figura 4.5.c. Observe que, devido a etapa 2 (transformar RdP em RPO, seção 4.2.1) os lugares agora possuem a classe *Student*, como indicado no Código 4.3.

A estrutura da rede apresenta as transições, que indicam como no exemplo do Código 4.6 o pré-requisito para o problema 4 é o problema 3, ambos pertencentes a unidade pedagógica

Código 4.2 Rede de Petri - Estudante

```
Class := tokenclass: Student:
(ID)
(name)
(doing_Curriculum)
(doing_Pb)
(doing_IU)
(multifield where)
(multifield done)
(multifield report)
(answer)
(bestgrade);
```

Código 4.3 Rede de Petri - Lugar

```
Places := GhostPlace: (Student);
UP2Pb1: (Student);
UP2Pb4: (Student);
UP2Pb0: (Student);
UP2Pb2: (Student);
UP2Pb3: (Student);
bufUP2Pb1UP2Pb2: (Student);
bufUP2Pb1bufUP2Pb1UP2Pb2: (Student);
bufUP2Pb2bufUP2Pb1UP2Pb2: (Student);
```

2. As transições da rede são controladas por condições lógicas (Código 4.5) que fazem referência ao modelo do estudante e o disparo destas transições produzem ações (Código 4.6) que atualizam o modelo do estudante. Como agora o lugar possui a classe Student, a variável *x* associada aos arcos pode instanciar a ficha que possui também a classe `Student`. É então necessário adicionar a cada transição condições que fazem intervir a variável formal *x* (associada aos arcos de entrada) e os atributos da classe Student associada a ficha (ver apêndice B).

Código 4.4 Rede de Petri - Estrutura

```
Structure :=
tUP2Pb3UP2Pb4: (UP2Pb3 (Student) ) → (UP2Pb4 (Student) );
```

Um exemplo de condição suplementar de disparo é a condição `isExp=True` associada a T1 na RPO-E (figura 4.7.b); a ação associada a esta mesma transição é doing_IU = next (doing_IU), onde *x* será instanciado pela classe Student.

4.5.2 Tradução da RPO em JESS

Neste trabalho, o modelo pedagógico é implementado em JESS e portanto, as RPOs são transformadas em regras (detalhes sobre a construção do tradutor RPO-JESS no Apêndice C). Contudo, uma outra forma de implementar as RPOs seria usando um jogador de Redes de

Código 4.5 Rede de Petri - Condição

```
Conditions :=
tUP2Pb3UP2Pb4: neq (member$(UP2Pb3, Student.done), FALSE);
```

Código 4.6 Rede de Petri - Ação

```
Actions :=
tUP2Pb3UP2Pb4: Student.doing_Pb := Next_Problem(Student);
```

Petri, ou seja, um motor de inferência especializado. A opção por adotar um *shell* de Sistemas Especialistas baseado em regras se deu basicamente pela possibilidade de adicionar habilidades que não estejam diretamente relacionadas com a RPO.

Em termos de implementação, as regras do JESS que correspondem a estrutura da RPO é obtida da seguinte maneira:

- Cada lugar $UPiPbj$ da rede corresponde a um fato $UPiPbj$ em JESS (*template*) do tipo `PLACE`;
- Cada transição $tUPiPbjUPyPbz$ da rede corresponde a um fato $tUPiPbjUPyPbz$ tipo `TRANSITION`;

  Arcos são implementados como propriedades dos fatos tipo `TRANSITION`;

  As condições de disparo e as ações são funções relacionadas com as `TRANSITION`.
- Cada ficha S da rede corresponde a um fato S do tipo `STUDENT`;

A definição dos fatos `PLACE` correspondente a um lugar da RPO é no Código 4.7. Supondo-se que a rede modela um domínio relacionado à matemática possíveis fatos são apresentados.

Código 4.7 Place - Lugar

```
(deftemplate place
    (slot name)
    (slot type (default problem))
    (multislot content))
(N1
    (name UP2Pb0)
     (type pedagogicalUnit)
     (content (Soma, Adição, Exponenciação)))
(N2
    (name UP2Pb1)
     (type problem)
     (content (Exponenciação)))
```

Segundo o número de lugares de entrada e de saída de uma transição, um *template* diferente é gerado. Para uma transição simples (um lugar de entrada, um lugar de saída), o *template* gerado é mostrado no Código 4.8. Os fatos gerados a partir deste template são exemplificados pela transição tUP2-Pb0UP2-Pb1.

Código 4.8 Trans - Transição

```
(deftemplate trans-1to1
    (slot name)
    (slot place-in1)
    (slot place-out1)
    (slot condition)
    (slot action))
(trans-1to1
    (name tUP2Pb0UP2Pb1)
    (place-in1 UP2Pb0)
    (place-out1 UP2Pb1)
    (condition cond_tUP2Pb0UP2Pb1)
    (action act_tUP2Pb0UP2Pb1))
```

No Apêndice D tem-se um código JESS completo para o estudo de caso Fundamentos da Estrutura da Informação modelado no capítulo 5.

O compilador não adiciona a semântica da rede automaticamente pois isto não permitiria a reutilização do código. É necessário estender o sistema da RPO compilado em JESS, adicionando regras que reflitam as particularidades das semânticas de cada RPO.

4.5.3 Comunicação Multi-nível

Neste trabalho, as redes RPO-CV e RPO-E foram traduzidas em duas bases de conhecimento (base de regras, correspondendo à estrutura tipada da RPO, e base de fatos correspondendo ao modelo do estudante): JESS-CV e JESS-E respectivamente. Estas duas bases serão executadas por duas instâncias do motor de inferência Jess.

Os lugares de comunicação nas redes RPO-CV e RPO-E, apresentados na seção 4.3.1 são traduzidos por condições extras adicionadas às regras Jess-CV e Jess-E, de forma à permitir a estas duas bases de regras de se comunicar. Na seção 5.2 descreve-se como é feita a interação entre o agente da SATA (ASTI) e cada uma das bases JESS-CV e JESS-E.

Base de Conhecimento JESS-CV

O algoritmo 4.9 permite a base de conhecimento JESS-CV verificar o momento de passar o controle para a base de conhecimento JESS-E:

Base de Conhecimento JESS-E

Na base de conhecimento JESS-E, os lugares representam UI ou uma semântica de controle. Os lugares de UI são: Explanation (Explicação), Example (Exemplo) e Exercise (Exercício),

Código 4.9 JESS-CV

if
A Ficha F está no lugar L
E o estudante, representado por esta ficha, ainda não está executando nenhuma UI
then
Envia a ficha F para JESS-E

e os de controle são *Begin*, *Halt* e *End* (ver especificação da rede de Petri da figura 4.7.b). Exceto os lugares de controle Halt e End, a regra que implementa a semântica dos lugares de UI é:

Código 4.10 JESS-E

if
A Ficha F está no lugar L
E o estudante representado pela ficha F ainda não deu uma resposta para a UI
E a ficha não está em modo de espera pela resposta
then
Envia os dados da UI para a interface do estudante aguardando a resposta

Os lugares Halt e End têm regras distintas: ao invés de enviar os dados da UI, enviam um comando informando que a interação com o usuário terminou, além dos dados do estudante para persistência.

Capítulo 5

ASTI: Arcabouço para Sistemas Tutores Inteligentes

A criação de um Sistema Tutor Inteligente costuma ser bastante onerosa no que diz respeito às exigências de tempo e pessoal. O desenvolvimento de arcabouços objetivam minimizar tais problemas fazendo uso de modelos complexos pré-estabelecidos.

Neste capítulo apresenta-se o arcabouço ASTI em desenvolvimento no grupo de pesquisa MathNet da UFSC. A ASTI se inspira e estende a arquitetura proposta pelo MATHEMA integrando a ferramenta FAST para a concepção do curso como mostrado na seção 3.3. E na seqüência tem-se o estudo de caso de dois domínios distintos modelados para a construção de Sistemas Tutores Inteligentes fazendo uso das ferramentas apresentadas nesta tese.

5.1 Arquitetura do Sistema

A arquitetura do arcabouço ASTI (ver figura 5.1) é composta por:

FAST : Fornece a estrutura dos modelos de domínio, pedagógico e estudante para o Sistema Tutor de cada agente da SATA, além das páginas com os conteúdos, como visto na seção 3.3.2;

SATA: Constitui uma sociedade de agentes tutores com conhecimento distribuído. Cada agente é responsável por um subdomínio do conhecimento e apresenta a estrutura de um STI clássico (ver figura 5.2):

1. O modelo de domínio é uma base de dados com os conteúdos a serem ensinados através de páginas HTML interativas, geradas a partir dos Java Servlets [Microsystems, b]. O conteúdo das páginas HTML é obtido a partir da Interface de Autoria.

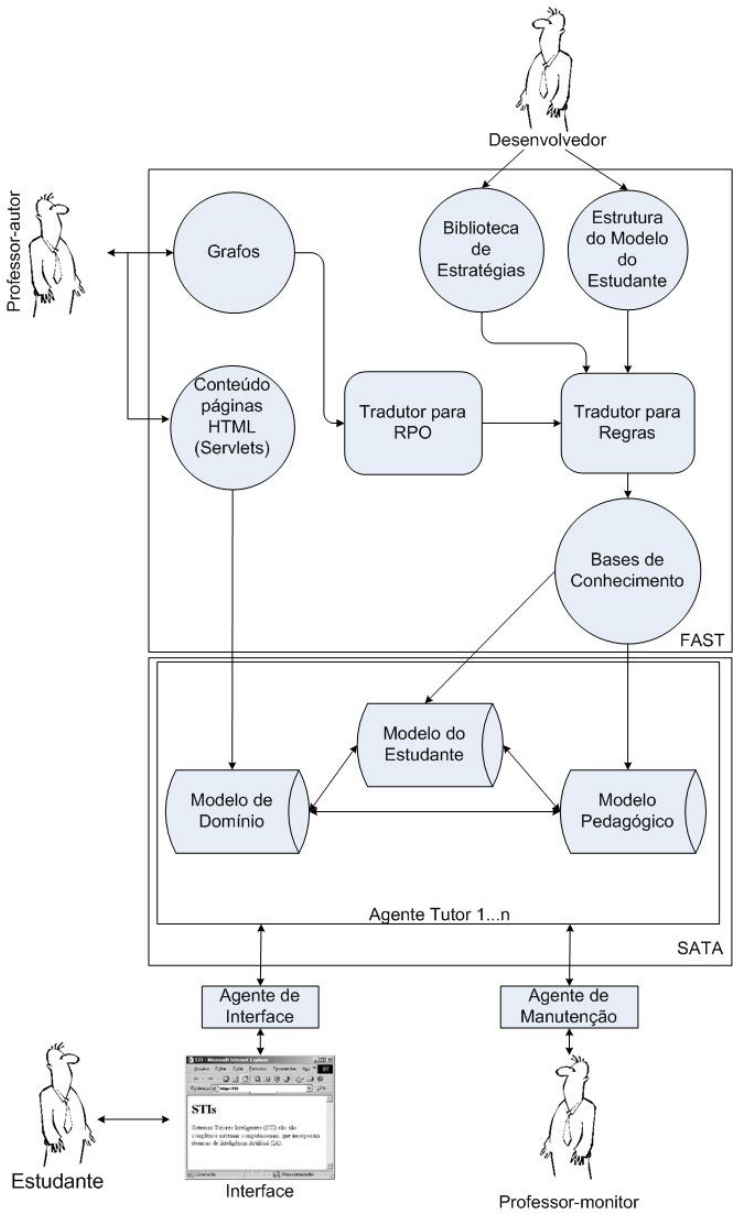


Figura 5.1: Arquitetura ASTI

2. O modelo pedagógico é gerado a partir dos grafos de pré-requisitos do professor-autor e das RPO criadas pelo desenvolvedor. O desenvolvedor cria uma biblioteca de RPO abrangendo as possíveis táticas e estratégicas pedagógicas a serem adotadas pelo professor. O professor escolhe quais os recursos utlizar. A partir do modelo pedagógico é possível controlar a interação entre o aluno e cada agente da SATA. O modelo pedagógico infere o que apresentar ao estudante a partir das regras e fatos contidos nas bases de conhecimento JESS-E e JESS-CV.
3. O modelo do estudante é criado pelo desenvolvedor do arcabouço a partir da ontologia da seção 4.1.2. Uma parte do modelo faz parte da base de conhecimento local do agente (informações dinâmicas) e a outra da base de conhecimento social (informações estáticas).

A arquitetura multiagente permite a distribuição do domínio de conhecimento e das informações do modelo do estudante entre os diversos agentes que trocam informações e cooperam na tarefa de auxiliar o estudante no processo de ensino-aprendizagem.

Agente de Interface : para cada estudante que se conecta no sistema é criado um agente de interface e é responsável pela troca de informações entre os Servlets e os agentes da SATA;

Agente de Manutenção : é através deste agente que o professor-monitor pode fazer eventuais alterações na SATA ou receber informações referente a mesma;

Interface: É através da interface que o estudante acessa o STI. A interface permite que o estudante altere suas informações pessoais, visualize os problemas e resolva os exercícios com o intuito de demonstrar a sua compreensão sobre como solucionar os problemas que lhe são apresentados. O professor-monitor utiliza uma interface própria para obter os relatórios de acompanhamento dos alunos e também para inserir ou modificar informações que estejam sob sua responsabilidade.

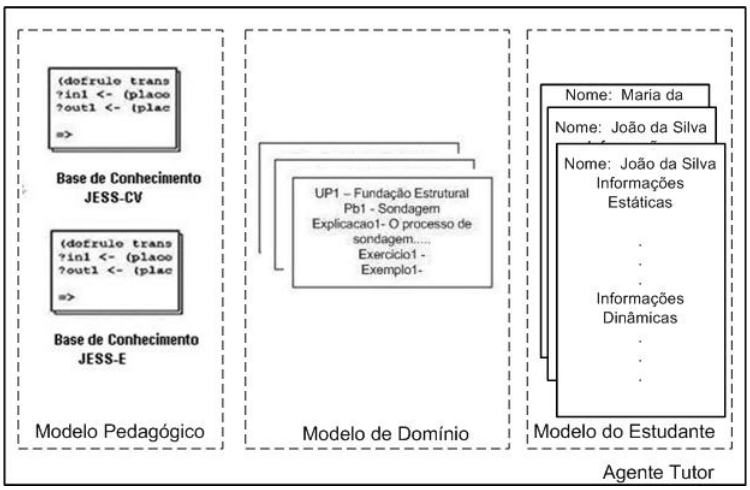


Figura 5.2: Sistema Tutor do Agente Tutor

5.2 Agentes

Sociedade de Agentes Tutores Artificiais

Entre os agente da SATA não existe hierarquia, todos eles podem conversar, negociar e trocar informações dependendo dos seus interesses. A SATA permite a distribuição do conteúdo do domínio e das informações do modelo do estudante entre os agentes que cooperam durante a tarefa de auxiliar o estudante.

Para simplificar o processo de compartilhamento e raciocínio os agentes utilizam uma representação do conhecimento comum. Cada agente possui seu próprio mecanismo de ra-

ciocínio para operar sobre o modelo de suas capacidades e o modelo das capacidades dos outros agentes. Para que isso seja possível existem três módulos denominados de:

- Autoconhecimento ou Conhecimento Local: é um modelo representando o que o agente sabe sobre as suas próprias habilidades e conhecimentos.
- Conhecimento Social: é um modelo no qual o agente representa explícitamente o conhecimento sobre os outros agentes tutores na sociedade.
- Alocação: é o módulo responsável pelo processo de seleção dos agentes identificados como aptos a resolver uma determinada tarefa que foi submetida pelo Sistema Tutor para cooperação.

Os agentes se comunicam a fim de solicitar cooperação e solucionar os problemas mais rapidamente. A comunicação entre os agentes é feita pela troca de mensagens facilitada pelo ambiente JADE [Bellifemine et al., 2004]. Além do Jade, outras plataformas para a implementação de agentes foram analisadas: FIPA-OS e Zeus. Apesar de todas as três plataformas seguirem os padrões FIPA, serem desenvolvidas em Java e possuírem distribuição gratuita (GNU Lesser General Public License), Jade se destaca pela facilidade de utilização, pela grande comunidade ativa e pelas atualizações constantes. Além disso, permite a representação do conhecimento com regras e inferência lógica através do JESS. Por estes motivos adotou-se o Jade.

Outros recursos que podem ser explorados através do Jade são: (i) as ontologias (RDF, ontologias) através do plug-in *BeanGenerator* para o Protégé [Stanford, b; Noy et al., 2000]; (ii) a implementação de autômatos através da classe *FSMBehaviour* que simula o comportamento de uma máquina de estados finita; (iii) o farto material, inclusive sobre o pacote jade.content que fornece suporte para as ontologias desenvolvidas no Protégé através do plug-in *BeanGenerator* permitindo aos agentes conversarem e raciocinarem sobre "coisas/conceitos e fatos".

A comunicação entre os agentes é feita através de performativas que armazenam ordens implícitas aos agentes. O comportamento de cada agente tutor é dado: (i) pelas bases de conhecimento JESS-E e JESS-CV (seção 4.5.3); (ii) pela interação com os outros agentes da SATA, em resposta as mensagens recebidas. Durante a comunicação os agentes enviam mensagens cujos conteúdos são fatos que serão inseridos na máquina JESS (Java Expert System Shell).

Comportamento do Agente Tutor

As bases de conhecimento JESS-CV e JESS-E permitem realizar a interação entre o estudante e o agente tutor criado em JADE através de Comportamentos (*Behaviours*) que representam as ações a serem executadas pelo agente. Assim que uma regra dispara, os agentes

recebem uma mensagem solicitando sua colaboração. O agente responsável pelo conhecimento necessário à resolução do problema para qual um "lugar" da rede está apontando (na verdade, um fato JESS da classe Place ver seção 4.5.2), deve passar a acompanhar o estudante. Caso o agente perceba que o estudante apresenta uma deficiência num assunto que está fora do conhecimento do agente, este deve solicitar o auxílio de outro agente. As comunicações e interações entre os agentes seguem as ontologias do capítulo 4.

Cada agente pertencente a SATA tem dois motores de inferência, um para cada base de conhecimento JESS-CV e JESS-E. O mecanismo de comunicação descrito na seção 4.3.1 é implementado da seguinte forma:

- JESS-CV envia a ficha para JESS-E;

- JESS-CV bloqueia a ficha para que não sejam disparadas outras regras;

- JESS-CV envia mensagem para o agente, contendo todos os dados da ficha necessários para JESS-E;

- agente recebe a mensagem da base JESS-CV;

- agente envia mensagem para JESS-E, indicando que deve ser inserida a ficha naquela *rede*;

- JESS-E recebe a mensagem para inserção da ficha;

- JESS-E roda regra que insere a ficha;

- A ficha percorre toda a *rede* em JESS-E e deve então retornar para a JESS-CV;

- JESS-E envia mensagem para o agente, contendo os dados da ficha, para que esta volte para JESS-CV;

- JESS-E elimina a ficha para que nenhuma regra seja disparada;

- agente recebe a mensagem da JESS-E;

- agente envia mensagem para JESS-CV, indicando que a ficha terminou a execução na JESS-E;

- JESS-CV recebe mensagem do agente;

- JESS-CV desbloqueia a ficha para que as regras possam ser disparadas e reinicia o ciclo.

Agente de Interface

Um estudante ao conectar-se no STI, ativa a criação de um agente de interface que o representa no sistema multiagente. No protótipo implementado criou-se um agente de interface

que reside num *Container* (máquina virtual ou processo - ver figura 5.3) pertencente ao servidor do Tomcat (servidor de aplicações Java para *Web*). Os agentes da SATA residem em um outro *Container*. Assim, o Agente de Interface possui um forte vínculo com os Servlets sem prejudicar a sua comunicação com os agentes da SATA. O Agente de Interface só tem comportamentos para interagir com os agentes da SATA e com os Servlets.

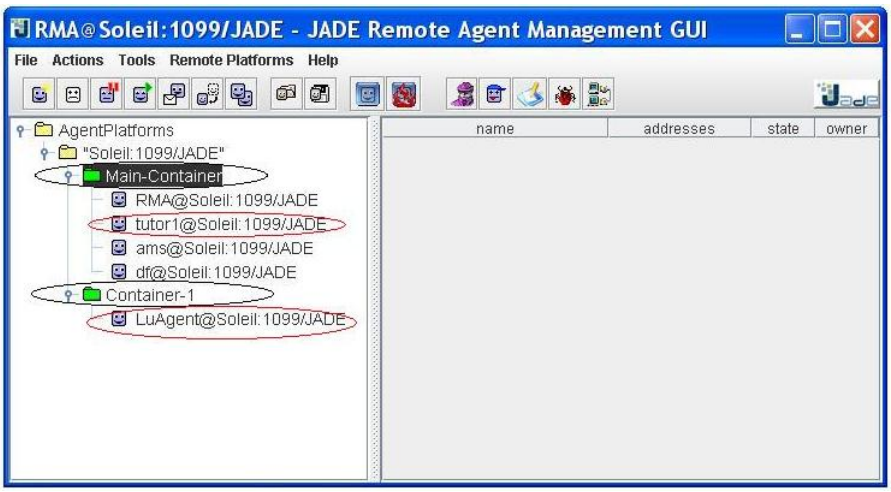


Figura 5.3: Agentes e seus *Containers*

5.3 Sistema Tutor Inteligente criado com a ASTI: Estudos de Caso

O funcionamento de um STI construído com a ASTI pode ser descrito da seguinte maneira:

- De acordo com as informações do modelo do estudante ou por escolha do estudante, um currículo é selecionado;
- As regras que implementam o modelo pedagógico do currículo escolhido são carregadas e o motor de inferência é inicializado;
- O motor de inferência, baseado nas regras e nas informações do modelo do estudante, inferem quais as unidades pedagógicas ou problemas podem ser apresentadas ao estudante;
- O agente de interface extrai as informações associadas com um problema inferido e envia para a interface na forma de uma página HTML;
- Caso o problema não necessite de nenhuma interação externa, por exemplo um ambiente de programação, o resultado da interação com o estudante é utilizado para atualizar diretamente o modelo do estudante;
- Quando a interação da interface com o estudante é finalizada, o motor de inferência executa novamente e o processo se repete.

Durante o processo de interação entre o STI e o estudante podem ocorrer dois eventos distintos que param o processo:

1. O currículo chega ao fim ou o estudante desiste encerrando a sessão: nesta situação um novo currículo é escolhido ou a sessão termina;
2. As regras inferem que a próxima interação devem ser controladas por um outro agente: o agente não possui o conhecimento necessário para auxiliar o estudante e um outro agente é requisitado.

Nas próximas seções são apresentados dois estudos de caso, o primeiro sobre a disciplina de Fundamentos da Estrutura da Informação e o segundo sobre o Preparo de Refeições. Os estudos de caso ilustram a criação de um curso através da ASTI.

5.3.1 Caso 1: Fundamentos da Estrutura da Informação

A disciplina de Fundamentos da Estrutura da Informação do Departamento de Automação e Sistema (DAS/UFSC) foi dividida em dois contextos, um teórico e outro prático.

A partir destas informações tem-se:

Domínio : Fundamentos da Estrutura da Informação

Contextos : C_1 = Visão Teórica C_2 = Visão Prática

Profundidades : P_{11} = Abstração Procedural P_{12} = Abstração de Dados

O conhecimento lateral inclui arquiteturas computacionais, linguagens de programação, em particular a linguagem Scheme [Felleisen et al., 2001], análise de complexidade, técnicas de engenharia de *software*, como por exemplo, para o par $\langle C_1, P_{11} \rangle$, pode-se ter:

Lateralidades : L_{111} = Ordens de Crescimento L_{112} = Modelo de Substituição

Definidos os subdomínios, estes podem ser associados a um curriculum, identificando as unidades pedagógicas que o constituem. Com base na experiência didática acumulada e na estrutura do livro texto adotado na disciplina [Abelson e Sussman, 1996], chegou-se à seguinte estrutura curricular: 7 unidades pedagógicas associadas tanto à visão teórica quanto à visão prática da abstração procedural e 8 unidades pedagógicas associadas às visões teórica e prática da abstração de dados.

(a) Abstração Procedural

- (UP1) Funções Primitivas

- (UP2) Funções Compostas
- (UP3) Iteração e Recursão
- (UP4) Funções como Objetos Manipuláveis
- (UP5) Funções como Argumentos
- (UP6) Funções como Métodos Gerais
- (UP7) Funções como Resultados de Funções

(b) Abstração de Dados

- (UP8) Dados Compostos
- (UP9) Barreiras de Abstração
- (UP10) Dados Hierárquicos
- (UP11) Seqüências como Interfaces Convencionais
- (UP12) Dados Simbólicos
- (UP13) Múltiplas Representações para Dados Abstratos
- (UP14) Programação Direcionada a Dados e Aditividade
- (UP15) Sistemas com Operações Genéricas

A SATA, para este domínio, consiste de quatro agentes, cada um deles é responsável por um dos seguintes subdomínios:

- PT – abstração procedural teórica;
- PP – abstração procedural prática;
- DT – abstração de dados teórica;
- DP – abstração de dados prática.

Cada *curriculum*, de acordo com o modelo MATHEMA, consiste em um conjunto de Unidades Pedagógicas (UP). O curriculum é formado por uma seleção e seqüenciamento do conteúdo a ser apresentado ao aprendiz (figura 5.4, baseada nas Unidades Pedagógicas da Abstração Procedural). O conhecimento associado a cada subdomínio é organizado em um ou mais *curricula*.

Cada UP é formada por um conjunto de problemas, que formam o plano de problemas. Cada problema possui uma série de exemplos, exercícios e explicações que formam o plano

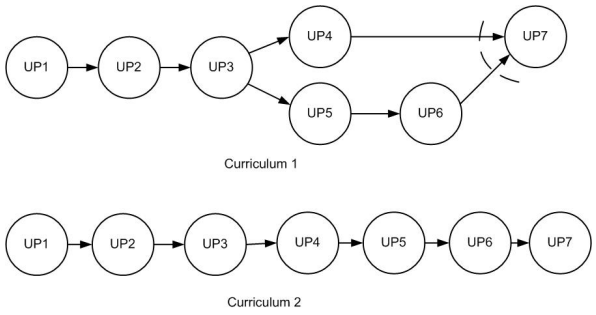


Figura 5.4: Curricula = currículo 1 + currículo 2

de suporte, ou seja, são as condições necessárias para que um determinado problema possa ser solucionado.

A partir da modelagem pode-se inserir os grafos na FAST. Para mostrar como são as informações obtidas através dos compiladores, apresenta-se o currículo 1 como exemplo.

A RPO para o currículo 1 (figura 5.4) contém as seguintes informações:

```
PETRINET

Class := tokenclass: Student:
          (ID)
          (name)
          (doing_Curriculum)
          (doing_Pb)
          (doing_IU)
          (multifield where)
          (multifield done)
          (multifield report)
          (answer)
          (bestgrade)
;

Places := GhostPlace: (Student);
UP3: (Student);
UP5: (Student);
UP7: (Student);
UP1: (Student);
UP2: (Student);
UP6: (Student);
UP4: (Student);
bufUP4UP6: (Student);
```

```
bufUP6bufUP4UP6: (Student);
bufUP4bufUP4UP6: (Student);

Structure :=
tUP2UP3: (UP2 (Student) ) -> (UP3 (Student) );
tUP3UP4UP5: (UP3 (Student) ) -> (UP4 (Student) , UP5 (Student) );
tbufUP4UP6UP7: (bufUP4UP6 (Student) ) -> (UP7 (Student) );
tUP1UP2: (UP1 (Student) ) -> (UP2 (Student) );
tUP5UP6: (UP5 (Student) ) -> (UP6 (Student) );
tbufUP6bufUP4UP6 : (UP6 (Student) ) -> (bufUP6bufUP4UP6 (Student) );
tbufUP4bufUP4UP6 : (UP4 (Student) ) -> (bufUP4bufUP4UP6 (Student) );
tbufUP4UP6 : (bufUP6bufUP4UP6 (Student) ,
bufUP4bufUP4UP6 (Student) ) -> (bufUP4UP6 (Student) );

Conditions :=
tUP2UP3: neq (member$(UP2, Student.done), FALSE);
tUP3UP4UP5: neq (member$(UP3, Student.done), FALSE);
tbufUP4UP6UP7: neq (member$(bufUP4UP6, Student.done), FALSE);
tUP1UP2: neq (member$(UP1, Student.done), FALSE);
tUP5UP6: neq (member$(UP5, Student.done), FALSE);
tbufUP6bufUP4UP6: neq (member$(UP6, Student.done), FALSE);
tbufUP4bufUP4UP6: neq (member$(UP6, Student.done), FALSE);

Actions :=
tUP2UP3: Student.doing_Pb := Next_Problem(Student);
tUP3UP4UP5: Student.doing_Pb := Next_Problem(Student);
tbufUP4UP6UP7: Student.doing_Pb := Next_Problem(Student);
tUP1UP2:Student.doing_Pb := Next_Problem(Student);

tUP5UP6: Student.doing_Pb := Next_Problem(Student);
tbufUP6bufUP4UP6: Student.doing_Pb := Next_Problem(Student);
tbufUP4bufUP4UP6: Student.doing_Pb := Next_Problem(Student);
tbufUP4UP6: Student.done := insert$ (Student.done, 1, bufUP4UP6);

ENDNET
```

As regras JESS correspondentes a RPO do currículo 1 (figura 5.4) são apresentadas no Apêndice D.

Para testar a modelagem do Sistema Tutor Inteligente da disciplina FEI foi criado um protótipo contendo algumas páginas de exemplos, exercícios e explicações como ilustram as figuras 5.5 e 5.6.

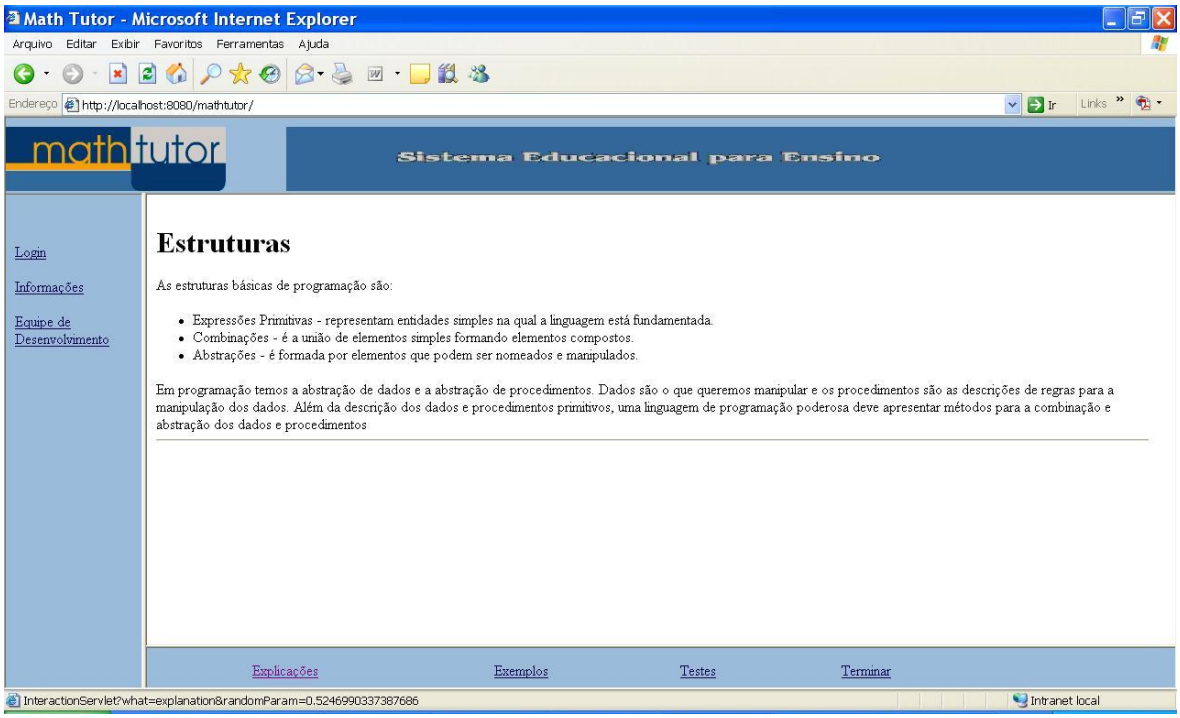


Figura 5.5: Página de explicação

5.3.2 Caso 2: Preparo de Refeições

Com o objetivo de apresentar as flexibilidade do STI, os recursos de adaptação e as formas de interação entre os agentes durante a apresentação de um currículo utiliza-se o curso destinado ao Preparo de Refeições. O público deste curso pode ser:

1. Aprendizes de *Chef* de Cozinha.
2. Treinamento de funcionários para restaurantes e para a indústria alimentícia.
3. Pessoas interessadas em aperfeiçoar seus conhecimentos culinários.

O Domínio *Preparo de Refeições* pode ser modelado como segue:

Este domínio pode ser particionado em dois diferentes contextos:
C_1 = Familiar
C_2 = Profissional

Para o contexto Familiar tem-se as seguintes Profundidades:
P_{11} = Cotidiano
P_{21} = Para Convidados

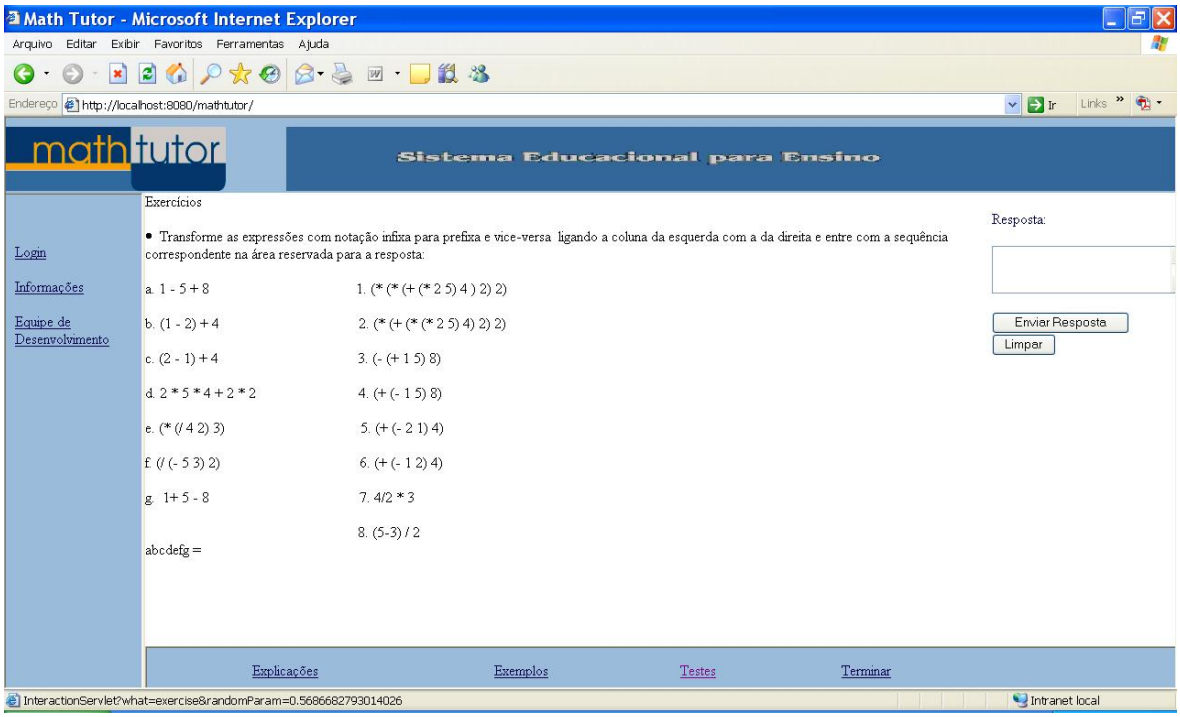


Figura 5.6: Página de exercício

E para o contexto Profissional as Profundidades podem ser:
P_{21} = Bistrô
P_{22} = Restaurante
P_{23} = Indústria de *Catering*

A notação adotada para relacionar um agente com o subdomínio sob sua responsabilidade é a seguinte: $A_{ij} = \langle C_i, P_{ij} \rangle$, por exemplo o agente A_{11} é responsável pelas Unidades Pedagógicas referentes ao par $\langle C_1, P_{11} \rangle$, ou seja, Familiar Cotidiano.

Para o contexto familiar, as Unidades Pedagógicas sob a responsabilidade do agente A_{11} são:
(UP1) Conhecendo os Alimentos
(UP2) Higiene e Segurança I
(UP3) Pesos e Medidas
(UP4) Utensílios Básicos
(UP5) Pratos Rápidos
(UP6) Pratos Elaborados
(UP7) Lista de Compras I
(UP8) Organização do Cardápio

Para o contexto familiar, as Unidades Pedagógicas sob a responsabilidade do agente A_{12} são:
(UP1) Cozinha Fria e Técnicas
(UP2) Cozinha Quente e Técnicas
(UP3) Cálculo de Quantidades
(UP4) Utensílios Especiais

Para o contexto profissional, as Unidades Pedagógicas sob a responsabilidade do agente A_{21} são:
(UP1) Custos I
(UP2) Higiene e Segurança II
(UP3) Criação de Menus
(UP4) Bebidas
(UP5) Cozinha Fria e Técnicas II
(UP6) Cozinha Quente e Técnicas II
(UP7) Lista de Compras II
(UP8) Cozinhas Especializadas I

Para o contexto profissional, as Unidades Pedagógicas sob a responsabilidade do agente A_{22} são:
(UP1) Cozinhas Especializadas II
(UP2) Tipos de Serviços

Para um estudante realizar as UPs do agente A_{22} ele necessita realizar anteriormente as UPs sob responsabilidade do agente A_{21}: Custos I, Higiene e Segurança II, Criação de Menus, Bebidas, Cozinha Fria e Técnicas II, Cozinha Quente e Técnicas II, Lista de Compras II.

O conhecimento de outros agentes não faz parte do STI do agente A_{22}. Neste caso o agente A_{22} necessita solicitar a ajuda de outros agentes da SATA. Um conhecimento não disponível em nenhum agente da SATA é chamado de conhecimento lateral. Para o agente A_{22} uma Lateralidade possível seria:

L_{221} = Alquimia dos Alimentos (aspectos físico-químicos)

O agente A_{23}, Profissional para indústria de *Catering* é o responsável pelo conteúdo relacionado ao preparo e distribuição de refeições (figura 5.7). Este agente foi selecionado para servir de exemplo para a modelagem dos problemas. As Unidades Pedagógicas e os Problemas associados a cada UP são:

Figura 5.7: Fornecimento de Refeições

(UP1) Custos II
Pb1 - Evitando Desperdícios
Pb2 - Controle de Estoque
Pb3 - Planilha de Custos

(UP2) Controle de Qualidade
Pb1 - Controle das Etapas de Fabricação
Pb2 - Inspeção de Qualidade
Pb3 - Controle na Indústria de Alimentos
Pb4 - Acondicionamento de produtos alimentícios para transporte e armazenamento

(UP3) Tratamentos Térmicos
Pb1 - Pasteurização
Pb2 - Esterilização
Pb3 - Liofilização
Pb4 - Congelamento

(UP4) Regeneração de Alimentos
Pb1 - Preparo
Pb2 - Métodos

(UP5) Logística
Pb1 - Cronograma de Entrega
Pb2 - Transporte e Distribuição
Pb3 - Estocagem

Para o agente A_{23} possíveis Lateralidades seriam:

L_{231} = Tabela nutricional
L_{232} = Cálculo de Calorias

Nos grafos que seguem as UPs relacionadas a um outro agente são representadas por um círculo incolor e fazem referência ao agente em questão.

A figura 5.8 ilustra um currículo modelado pelo professor fixando o agente responsável pelo par $\langle C_2, P_{23} \rangle$. O curso poderia ser destinado aos funcionários da linha de produção cujo maior interesse se destina as áreas de preparo dos alimentos.

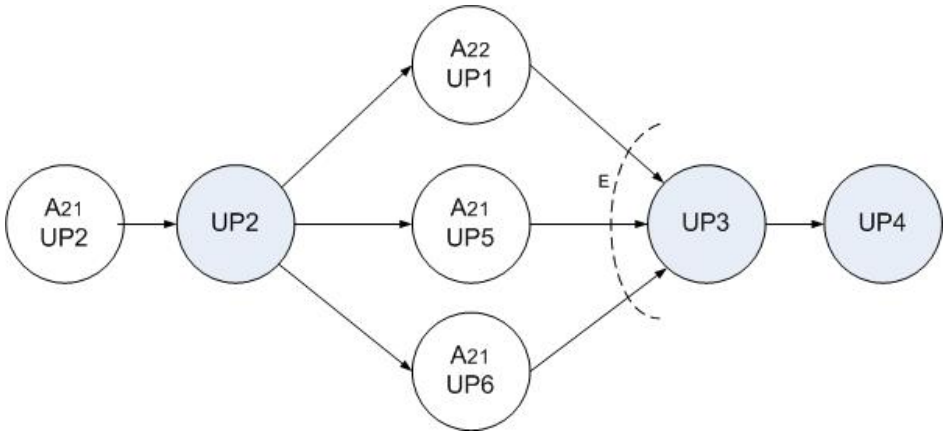


Figura 5.8: Currículo: Linha de Produção

Outras possibilidades poderiam ser modeladas de acordo com os objetivos de cada professor ou com o público o qual se destina. Por exemplo, um *Chef* tem como principais responsabilidades a criação do cardápio e a verificação da execução do mesmo, no entanto os seus conhecimentos devem ser abrangentes o suficiente para que ele possa realizar um bom trabalho. Um curso destinado a este tipo de profissional é apresentado na figura 5.9.

Após inserir os grafos dos currículos o professor deve inserir o grafo de pré-requisitos dos problemas. Possíveis grafos para os problemas do agente A_{23} são apresentados na figura 5.10. Para todas as UPs, no grafo dos problemas adota-se um nó inicial $Pb0$ que contém a explicação introdutória da UP.

Após entrar com os grafos dos problemas, o professor deve inserir também os conteúdos associados: Exercícios, Exemplos e Explicações. Esta etapa costuma ser bastante lenta pois todo o material precisa ser passado para a FAST.

As explicações são questões relacionadas ao problema como por exemplo: Para a unidade pedagógica que se refere aos Custos uma explicação seriam informações relacionadas a como evitar desperdícios através do reaproveitamento do alimentos. E um possível exemplo seria utilizar a casca de legumes no preparo de outros pratos. Vários tipos de exercícios são

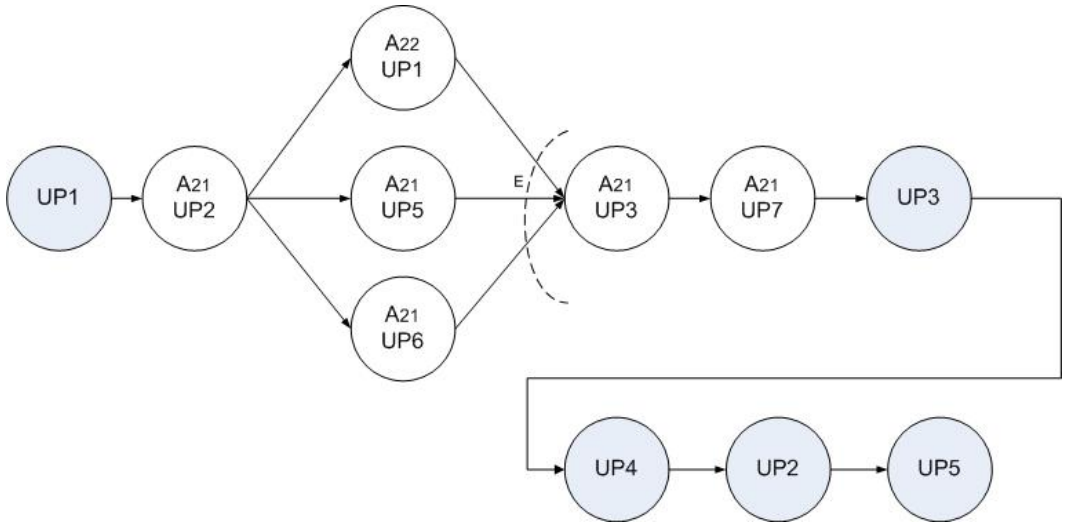


Figura 5.9: Currículo: *Chef* de Cozinha

possíveis de serem introduzidos, algumas possibilidades são: responder a questões do tipo verdadeiro ou falso, correspondência de colunas, múltipla escolha, etc.

O trabalho do professor termina quando ele fornece a FAST todas as informações necessárias para a construção do STI.

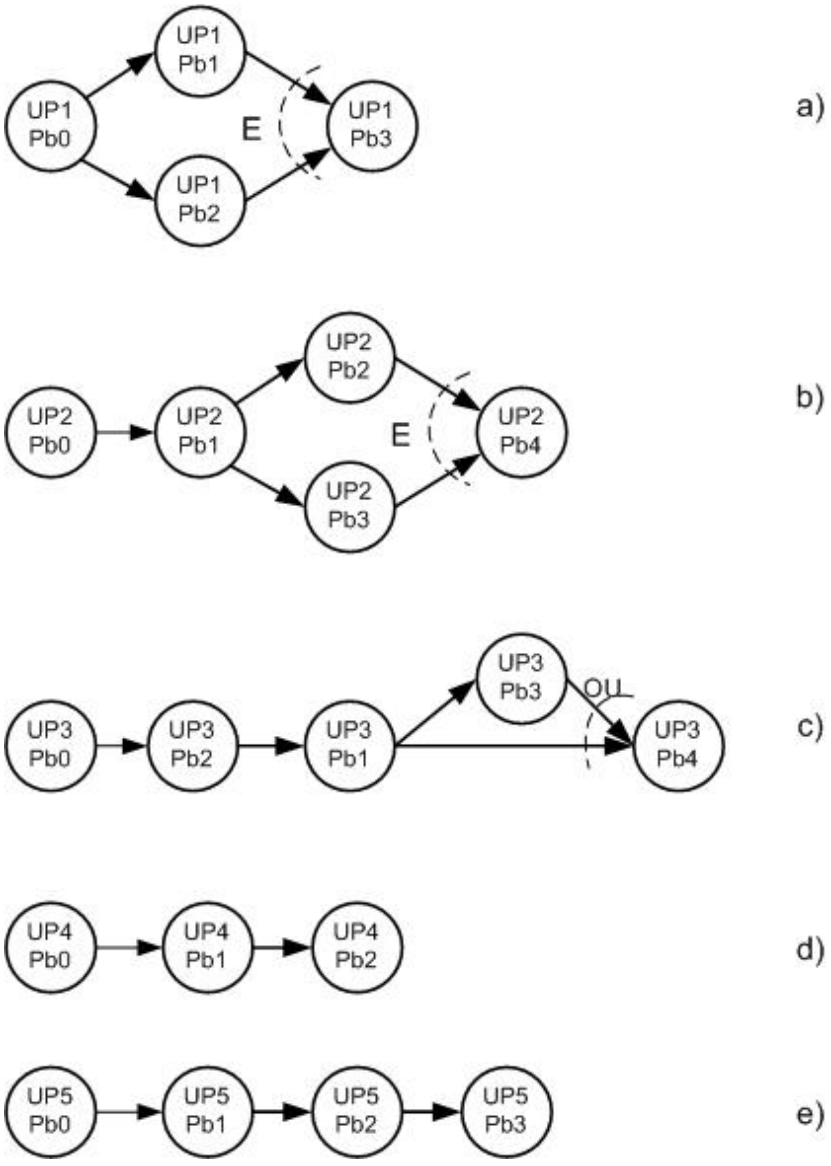


Figura 5.10: Grafo dos Problemas do Agente A_{23}

Capítulo 6

Conclusões e Perspectivas

Esta tese propõe um quadro metodológico acompanhado de ferramentas computacionais para desenvolver um mecanismo de controle adaptativo para as interações de um estudante com um Sistema Tutor Inteligente. Este quadro metodológico de criação de cursos deve permitir uma integração automática do conteúdo fornecido pelo professor com um mecanismo de adaptação ao aluno. O modelo conceitual foi concretizado na ferramenta FAST que permite gerar a descrição dos modelos de domínio, pedagógico e do estudante. Além disso, um Arcabouço para Sistemas Tutores Inteligentes (ASTI) que integra a FAST foi parcialmente realizado.

O mecanismo de controle é baseado em ontologias e em Redes de Petri a Objetos (RPO) e integra os modelos de domínio, do estudante e pedagógico. Todo o gerenciamento das informações armazenadas no modelo do estudante é feito automaticamente através do modelo pedagógico. O professor, durante o desenvolvimento de um curso, não precisa se preocupar em especificar como estas informações do modelo do estudante devem ser utilizadas para guiar a exposição dos conteúdos. No entanto, o professor deve criar o curso de acordo com uma estrutura pré-definida, inspirada no modelo MATHEMA. Somente o desenvolvedor do sistema, que propõe os protocolos de interação e suas linguagens de especificação (a serem usadas pelos professores) utiliza os formalismos de RPO e ontologias, facilitando a utilização do sistema por pessoas que desconhecem estes formalismos.

Estende-se o modelo original MATHEMA incluindo novas funcionalidades como: (i) uma classificação genérica dos usuários do ambiente educacional computacional através da definição dos papéis que cada um desempenha; (ii) um mecanismo de adaptação do ambiente educacional computacional ao estudante; (iii) simplificação das tarefas a serem exercidas pelo professor para criar o conteúdo e a apresentação do curso, tornando os modelos do estudante e pedagógico transparentes ao professor.

As principais contribuições do modelo proposto são:

- Sob o ponto de vista do desenvolvedor: a utilização das RdP para integrar os mode-

los da arquitetura de um STI, facilitando a visualização das possíveis trajetórias efetuadas pelo estudante, a fim de evitar caminhos inconsistentes; o uso de ontologias para a representação dos modelos de domínio e do estudante, facilitando inspeção, manutenção, verificação e reuso.

- Sob o ponto de vista do professor: a redução do trabalho exaustivo e improdutivo, atenuando a complexidade inerente ao processo ensino-aprendizagem, se preocupando somente com a organização do conteúdo.
- Sob o ponto de vista do estudante: a utilização de um ambiente complexo capaz de atender as suas necessidades.

Perspectivas Futuras

A interação entre estudante e tutor tenta se aproximar cada vez mais da interação estudante e professor tornando o sistema uma extensão da sala de aula e fazendo com que esta interação ocorra de maneira mais natural possível. Contudo, existe uma lacuna em relação a modelagem do estudante, por exemplo, quando o professor percebe uma determinada expressão na face do estudante que o tutor artificial não é capaz de perceber. Por fatores semelhantes a estes, a modelagem do estudante ainda é uma área carente de evolução.

Uma perspectiva de curto prazo é a expansão do modelo do estudante com o objetivo de refinar as interações do estudante com o tutor. A mais longo prazo, pretende-se estender o modelo teórico de interação proposto para permitir o aprendizado em grupo. Isto demonstra a extensibilidade do modelo pois, em termos do modelo do estudante, além de especificar uma nova ontologia sobre grupos, será necessário apenas estender a ontologia do modelo do estudante introduzindo novos atributos relevantes à interação de grupos. Já em relação ao modelo pedagógico será necessário desenvolver apenas novas, e mais complexas, RPO do nível de estratégias que sejam capazes de gerenciar as atividades do grupo.

Em relação à ferramenta de autoria, diversas possibilidades estão em aberto, como, por exemplo:

- Desenvolvimento de uma biblioteca de unidades de interação que permitam a utilização de multimídia.
- Definição de APIs (Application Programming Interface) de maneira a permitir a integração entre o Arcabouço para Sistemas Tutores Inteligentes (ASTI) e programas resolvedores de problemas tais como ambientes de programação, provadores automáticos de teoremas, ferramentas gráficas, etc.
- A definição de um mecanismo para a criação automática de novos agentes da SATA.

- Um método para automatizar o desenvolvimento de RPO para o controle das estratégias pedagógicas de modo que estas possam ser especificadas sem a necessidade de intervenção do desenvolvedor.

Certamente novas possibilidades surgirão com a experiência de desenvolvimento de novos cursos utilizando a ferramenta.

Apêndice A

Ferramentas de Suporte à Implementação

As ferramentas necessárias para implementação da FAST são brevemente descritas a seguir.

Java

Java[Microsystems, a] é uma linguagem independente de plataforma, ou seja, pode ser entendida e processada por máquinas que rodam diferentes sistemas operacionais.

Java é composto basicamente por três elementos: uma linguagem orientada a objetos, um conjunto de bibliotecas e uma máquina virtual no qual os programas são interpretados.

O funcionamento de um programa Java, da criação à execução, pode ser descrito da seguinte forma: cria-se um código fonte que passa por um compilador gerando um novo documento codificado conhecido como *bytecode* (todos os equipamentos capazes de executar um programa Java saberão interpretar este documento); para uma correta interpretação do documento basta ter uma máquina virtual Java (JVM - Java Virtual Machine) que é capaz de ler e executar o *bytecode*.

A linguagem Java é bastante utilizada, principalmente para aplicações via **web** e de código livre. A escolha do Java deu-se basicamente pela sua portabilidade, afinidade com a filosofia de código livre e pela grande comunidade que trabalha no desenvolvimento de aplicações semelhantes.

JavaCC

Java Compiler Compiler [tm] [JavaCC] é um gerador de compiladores para uso com aplicações em Java. Esse gerador de compiladores é uma ferramenta que lê uma especificação de

gramática e converte-a em um programa Java capaz de reconhecer se um determinado texto pertence à gramática especificada (análise léxica e sintática).

O JavaCC toma como entrada um arquivo, de especificação de gramática, que pode conter também código Java embutido entre as construções da gramática, o que permite a geração de código segundo a semântica associada a cada expressão sintática da gramática.

JADE

Jade (Java Agent DEvelopment framework) [Bellifemine et al., 2004] é um ambiente para desenvolvimento de aplicações distribuídas baseado em agentes conforme as especificações da FIPA (Foundation for Intelligent Physical Agents) para interoperabilidade entre sistemas multiagentes, como por exemplo: serviço de nomes e páginas amarelas, transporte de mensagens, serviços de codificação e decodificação de mensagens e uma biblioteca de protocolos de interação pronta para ser usada. JADE é totalmente implementado em Java e segue a filosofia de código livre (LGPL).

A comunicação entre agentes é feita via troca de mensagens. Outros aspectos que não fazem parte do agente em si e que são independentes das aplicações, tais como transporte de mensagens, codificação e interpretação de mensagens e ciclo de vida dos agentes, também são tratados pelo JADE.

JESS

JESS (Java Expert System Shell) é um arcabouço para sistemas especialistas desenvolvido por Friedman-Hill [1997] no Sandia National Laboratories como um projeto de pesquisa interno do laboratório. O motor de inferência é de uso livre para a comunidade acadêmica.

O JESS possui as mesmas funcionalidades básicas providas por uma máquina de inferência como Prolog [Colmerauer, 1985] e tem a vantagem de ser uma biblioteca Java, permitir que dentro de um programa Java seja executado o motor de inferência, fornecendo assim uma interface entre a aplicação e o motor de inferência.

O JESS é utilizado em diversas aplicações, mas o uso do JESS com a tecnologia dos Applets deixa o sistema muito pesado. Por isso quando a idéia é utilizar aplicações com JESS via navegador, devemos considerar o uso do JESS do lado do servidor, como o que ocorre no caso dos Servlets [Microsystems, b]. Assim, envia-se apenas o resultado do processamento para o estudante, dispensando-o de carregar grande parte do sistema para sua máquina, o que torna a interação com o sistema bastante lenta e entediante sob o ponto de vista do estudante.

Servlets

Java Servlets [Microsystems, b] são aplicações Java que rodam dentro de um servidor *Web*. Os Java Servlets possuem um modelo de programação similar aos *scripts* de CGI (tecnologia que permite gerar páginas dinâmicas permitindo a um navegador passar parâmetros para um programa alojado num servidor *Web*), na medida em que eles recebem uma solicitação HTTP de um navegador *Web* como entrada e espera-se que localizem e/ou construam o conteúdo apropriado para a resposta do servidor. Todos os Servlets associados a um servidor *web* rodam dentro de um único processo. Ao invés de criar um processo para cada solicitação, o JVM (do inglês *Java Virtual Machine*, programa específico da plataforma para rodar programas Java compilados) cria um encadeamento Java para tratar de cada solicitação de Servlet. Já que a JVM persiste além de uma única solicitação, os Servlets também podem evitar muitas operações demoradas, como conexão a um banco de dados, ao compartilhá-lo entre as solicitações. Pelo fato de serem escritos em Java, os Servlets aproveitam todos os benefícios da plataforma Java básica como: um modelo de programação Orientado a Objetos, gerenciamento automático de memória, portabilidade compatível com várias plataformas etc. Os Servlets fornecem uma metodologia baseada em Java para mapear solicitações de HTTP em respostas HTTP (ver Figura A.1). A geração do conteúdo da *web* dinâmico usando Servlets é realizada através de código Java que fornece a HTML representando aquele conteúdo [Fields e Kolb, 2000].

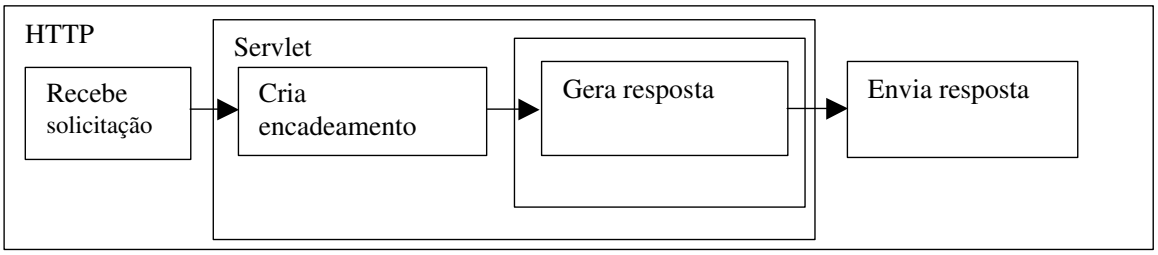


Figura A.1: Servlets

Tomcat

O servidor de aplicações Java para web utilizado é o Tomcat, que é distribuído como software livre e desenvolvido como código aberto dentro do projeto Apache Jakarta. Sua principal característica técnica é estar centrada na linguagem de programação Java, mais especificamente nas tecnologias de Servlets e de Java Server Pages (JSP). A Fundação Apache, mais conhecida pelo seu servidor web de mesmo nome, permite, como no caso do servidor Apache, que o Tomcat seja usado livremente.

TINA

TINA (TIme petri Net Analyzer) [Berthomieu et al., 2004] é uma ferramenta desenvolvida para edição e análises de redes de Petri, redes de Petri Temporais e autômatos. Apresenta uma interface razoavelmente amigável, dispondo de um editor que descreve a rede de maneira gráfica ou textual. Além disso, faz análises estruturais e fornece as principais propriedades da rede.

Protégé

Protégé [Stanford, b; Noy et al., 2000] é um editor de ontologias que possui uma interface gráfica de fácil utilização para o desenvolvimento de sistemas baseados em conhecimento.

Protégé é feito em Java, tem código aberto e permite a criação, visualização e manipulação de ontologias em vários formatos de representação (RDF, OWL e XML Schema).

Uma vantagem do Protégé é a existência de diversas extensões como por exemplo o componente que integra o Jess chamado JessTab [Eriksson, 2003]. Este componente permite que o Jess manipule ontologias através do Protégé.

Apêndice B

Redes de Petri

A Rede de Petri (RdP) é um modelo matemático com representação gráfica que se adapta a muitas aplicações como, sistemas de manufatura, de comunicação, de transporte, de informação entre outros, onde as noções de eventos e evoluções simultâneas são importantes.

A teoria das Redes de Petri (RdP) foi proposta na tese, intitulada *Comunicação com autômatos*, defendida por Carl Adam Petri em 1962 na Universidade de Darmstadt, Alemanha.

Os elementos básicos de uma RdP são [Cardoso e Valette, 1997]:

Lugar representado por um círculo e pode ser interpretado como uma condição, um estado parcial, uma espera, um procedimento, etc.

Transição representada por uma barra ou retângulo e está associada a um evento que ocorre no sistema.

Ficha representada por um ponto num lugar e indica que a condição associada ao lugar é verificada.

Os lugares e as transições são conectados através de **arcos**, representados por setas, com pesos associados. Por definição, um arco não marcado tem peso 1.

O comportamento de muitos sistemas pode ser descrito em termos de seus estados e da evolução desses estados. O estado do sistema é dado pela distribuição das fichas (marcação) nos lugares da RdP e cada lugar representa um estado parcial do sistema.

De modo a simular o comportamento dinâmico do sistema modelado, a marcação da RdP e, por conseguinte, o estado do sistema modelado, deve ser modificada de acordo com a regra de habilitação e disparo de uma transição:

(a) cada um dos lugares de entrada da transição deve possuir um número de fichas maior ou igual ao peso do respectivo arco (a transição está sensibilizada);

(b) o disparo de uma transição consiste em retirar de cada lugar de entrada um número igual ao peso do respectivo arco e em adicionar a cada lugar de saída um número de fichas igual ao peso do respectivo arco.

Uma transição T está habilitada ou sensibilizada, se e somente se o número de fichas em cada um dos lugares de entrada for maior ou igual ao peso do arco que liga este lugar P à transição [Cardoso e Valette, 1997]. As Figuras B.1 e B.2 mostram exemplos de RdP antes e depois dos disparos, onde L_i são os lugares e T_j as transições.

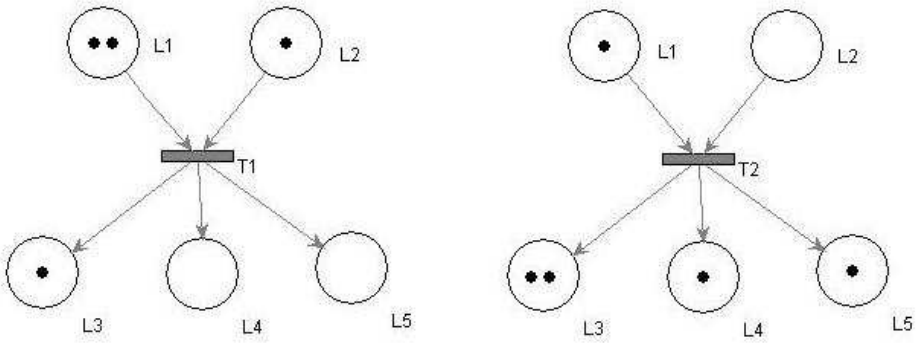


Figura B.1: Exemplo de uma RdP com transições AND

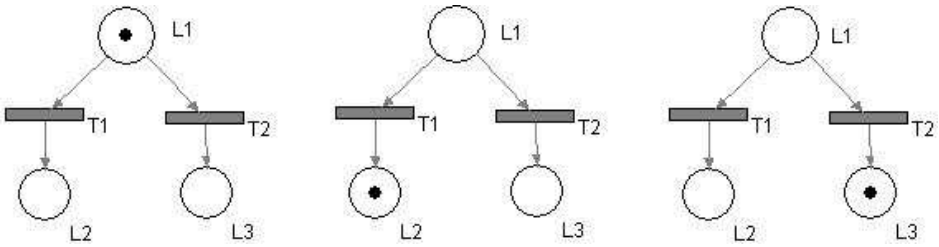


Figura B.2: Exemplo de uma RdP com transições OR

A RdP pode ser considerada como um sistema de regras de produção baseado em uma representação da forma *se* condição *então* ação.

Um sistema de regras forma-se por [Cardoso e Valette, 1997]:

- uma base de fatos, representando o conhecimento disponível no sistema;
- uma base de regras, que permite deduzir novos fatos;
- um mecanismo de inferência, chamado *motor de inferência*, que permite realizar novas deduções, aplicando as regras aos fatos.

O mecanismo de inferência elementar dispõe de um conjunto de regras e as percorre seqüencialmente. Caso a *condição* da regra seja verdadeira no contexto atual, a regra é aplicada ou disparada. Caso nenhuma regra seja satisfeita, o mecanismo de inferência pára.

Quando um sistema necessita de um formalismo mais complexo que possa descrever com precisão o comportamento e a organização dos seus processos, principalmente quando estes acontecem de maneira concorrente, utilizam-se na maioria das vezes as RdPs por serem um modelo de representação para sistemas multi-processos.

As RdPs podem ser consideradas como uma ferramenta de modelagem de processos assíncrona baseada na representação gráfica e matemática [Ferber, 1999]. As RdPs são utilizadas para modelagem de processos nas mais distintas áreas onde há a necessidade de mecanismos de representação que são implementados de modo paralelo.

Redes Elementares

As redes elementares são utilizadas como blocos básicos que possibilitam a especificação de aplicações mais complexas.

Seqüenciamento: as atividades são realizadas umas após as outras.

Distribuição: permite que as atividades possam ser feitas de maneira independente (ou ao mesmo tempo).

Junção: quando uma atividade só pode ser realizada depois que um conjunto anterior de atividades foi realizado.

Escolha Não-determinística: após a realização de uma atividade, deve ser feita uma escolha entre outras atividades. Ao fazer a escolha, automaticamente exclui-se as outras possibilidades (OU exclusivo).

Atribuição: permite que duas ou mais atividades possibilitem a execução de uma mesma atividade.

Propriedades

Após a modelagem de um sistema, é necessário verificar que o modelo construído a partir da especificação informal do sistema seja correto. Para tal, é necessário que a RdP apresente boas propriedades: *limitação* e *vivacidade*. Da análise destas propriedades conclui-se sobre a correção do modelo e conhece-se algumas características de comportamento do sistema, como a ausência de bloqueios. As boas propriedades de uma RdP são chamadas de comportamentais, e são dependentes da marcação da rede:

Limitacao (*Boundedness*):

Definicao: seja um lugar $p_i \in P$, de uma RdP marcada $N_1 = (R, M_0)$. Este lugar é

dito $k-limitado(k \in \mathbb{N})$ ou simplesmente limitado, se para toda marcação acessível, $M(p_i) \leq k$.

Caso a propriedade da definição acima não seja observada, o lugar é dito não-limitado. O limite k é o numero de fichas que um lugar pode acumular. Uma RdP marcada $N = (R, M_0)$ é dita $k-limitada$ se o número de fichas de cada lugar de N não excede k em qualquer marcação acessível. Como as fichas representam recursos disponíveis, a limitação de uma rede significa que o sistema modelado possui recursos e operações finitas, como um sistema real.

Vivacidade:

Definição: uma rede $N_1 = (R, M_0)$ é dita viva se qualquer transição puder ser disparada a partir de qualquer marcação alcançável. Se a RdP é viva então não há bloqueios nem partes inativas, pois todas as transições pertencem a alguma seqüência de disparo.

Uma marcação em bloqueio (*deadlock*) não sensibiliza nenhuma transição. A vivacidade é condição suficiente, mas não necessária, para a ausência de bloqueios na rede. Um *live lock* é um ciclo de disparos de transições repetitivo, sem saída, não passando por M_0. Outras boas propriedades podem ser observadas: alcançabilidade (*reachability*), segurança (*safeness*), cobertura (*coverability*), persistência, reversibilidade, justiça (*fairness*), distância síncrona.

Existem várias extensões das RdPs, de acordo com as características do sistema que está sendo modelado: aspectos temporais (RdP temporal, RdP temporizada, etc), aspectos de informação (RdP a objetos, RdP coloridas, etc). No decorrer da presente tese serão utilizadas as Rede de Petri a Objetos [Sibertin-Blanc, 1985] descritas a seguir.

B.1 Redes de Petri a Objetos - RPO

Uma abordagem que tem merecido destaque para o problema da estruturação é a integração das RdPs com os conceitos de orientação a objetos (OO). A idéia de Rede de Petri a Objetos (RPO) é utilizar as construções da OO para organizar os modelos. Logo, a construção chave da notação é a classe. Cada classe descreve um tipo de entidade do sistema. Neste caso, os objetos, que são as instâncias das classes, modelam entidades autônomas e concorrentes. Cada marcação da rede determina um possível estado das instâncias da classe. De forma análoga, as transições da rede modelam possíveis ações dos objetos. A comunicação entre os objetos do sistema completa a teoria de RPO. O mecanismo básico de comunicação assumido é a passagem assíncrona de mensagens. A Rede de Petri a Objetos vem suprir a necessidade de se modelar Sistemas Computacionais, onde a estrutura de dados tem um efeito no comportamento destes sistemas [Sibertin-Blanc, 1985].

Para descrever objetos é preciso definir um conjunto de atributos, um conjunto de métodos e um corpo. Os valores dos atributos são o estado do objeto; os métodos são um conjunto

de eventos ou operações que podem ocorrer no objeto e alterar seu estado; o corpo determina em que circunstâncias os métodos são ativados e como alteram o valor dos atributos, ou seja, a descrição do comportamento dos objetos da classe. Além dos atributos definidos pela classe, um atributo implícito contém o nome do lugar em que o objeto está localizado. As operações são associadas às transições, e correspondem às pré-condições que operam sobre os atributos dos objetos situados nos lugares de entrada, e às ações que modificam estes valores. Uma operação associada a uma transição t só poderá ser executada por um objeto se este estiver localizado em um lugar de entrada de t [Cardoso e Valette, 1997].

A Rede de Petri a Objetos pode ser considerada como uma utilização da rede de Petri predicado-transição no contexto de uma abordagem a objetos. Nas redes predicado-transição, as transições de uma rede de Petri ordinária são consideradas como regras de um sistema de lógica proposicional (sem variáveis), e o poder de descrição é aumentado substituindo-se por regras da lógica de primeira ordem (regras com variáveis) [Cardoso e Valette, 1997].

Na Rede de Petri a Objetos as fichas não são mais constantes, mas sim instâncias de n-uplas de classes de objetos. Além dos atributos definidos pela classe, um atributo implícito contém o nome do lugar em que o objeto está localizado. As operações são associadas às transições, e correspondem às pré-condições que operam sobre atributos dos objetos situados nos lugares de entrada, e às ações que modificam estes valores.

Uma rede de Petri a objeto é definida pela 9-upla [Cardoso e Valette, 1997]:

$N_o =< P, T, C_{lass}, V, Pre, Post, A_{tc}, A_{ta}, M_0 >$,

onde:

- C_{lass} é um conjunto finito de classes de objetos, eventualmente organizado em uma hierarquia e definindo para cada classe um conjunto de atributos;
- P é um conjunto finito de lugares cujos tipos são dados por C_{lass};
- T é um conjunto finito de transições;
- V é um conjunto de variáveis cujos tipos são dados por C_{lass};
- Pre é a função *lugar precedente* que a cada arco de entrada de uma transição faz corresponder uma soma formal de n-uplas de elementos V;
- $Post$ é a função *lugar seguinte* que a cada arco de saída de uma transição faz corresponder uma soma formal de n-uplas de elementos V;
- A_{tc} é uma aplicação que a cada transição associa uma condição fazendo intervir as variáveis formais associadas aos arcos de entrada e aos atributos das classes correspondentes;

- A_{ta} é uma aplicação que a cada transição associa uma ação fazendo intervir as variáveis formais associadas aos arcos de entrada e aos atributos das classes correspondentes;
- M_0 é uma marcação inicial que associa a cada lugar uma soma formal de n-uplas de instâncias de objetos (os objetos devem ser representados por identificadores, como por exemplo, seu nome).

RPO é uma linguagem formal de modelagem de sistemas distribuídos e concorrentes. Embora possa ser considerada de propósito geral, RPO foi proposta com o objetivo de ser particularmente adequada para a validação de sistemas de informação. Formalmente, RPO é uma linguagem que integra conceitos de orientação a objetos a redes de Petri de alto nível. Na prática, RPO permite usar os princípios da orientação a objetos para construir modelos complexos em redes de Petri, sem que seja necessário abrir mão completamente da viabilidade de análise matemática.

Apêndice C

Construção do Compilador (Rede de Petri - JESS)

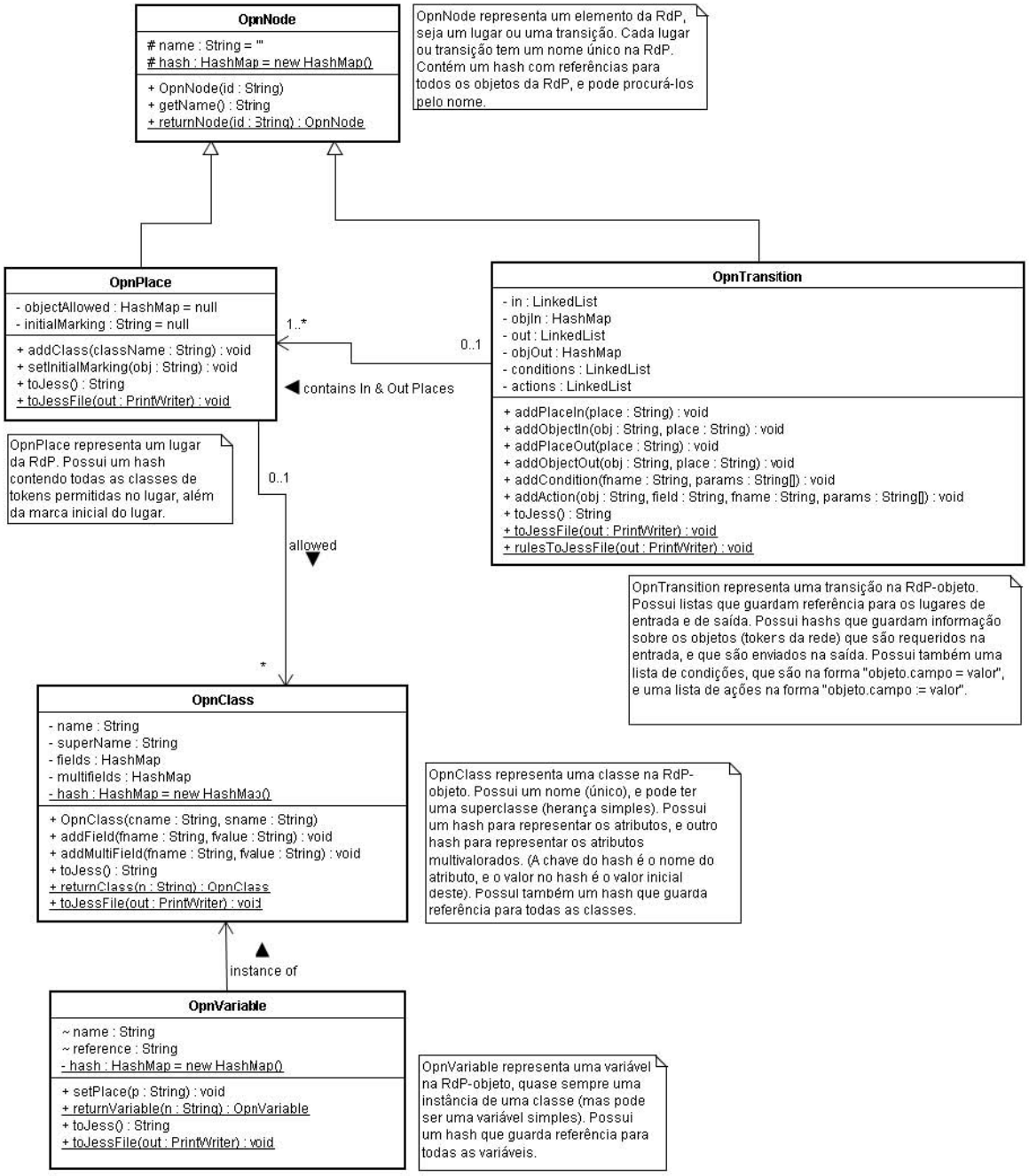
O primeiro passo é fazer a análise sintática (*parsing*) de um arquivo contendo uma representação da RPO. Para executar o parsing, foi usada a ferramenta JavaCC [JavaCC]. O JavaCC gera um programa (na verdade, uma classe) em Java, que analisa sintaticamente uma determinada construção de uma gramática. O JavaCC permite expressar uma Gramática Livre de Contexto, apenas substituindo as variáveis/produções por métodos.

O segundo passo é instanciar estruturas de dados representando a RPO. Somente com a descrição da gramática, o JavaCC não gera resultados muito interessantes: a classe só consegue apontar erros quando, na entrada, os dados não estão conforme a gramática. Portanto, é preciso inserir alguma semântica dentro das produções da gramática. Isso é feito instanciando/manipulando algumas estruturas de dados.

O diagrama de classes contendo as estruturas usadas no compilador está na figura C.1.

O terceiro passo é implementar nas estruturas de dados, métodos que transformem a RPO num sistema de regras. Executado o parsing, e instanciadas as estruturas de dados sem erros seguido pela tradução destas estruturas em forma de um sistema de regras.

Visando a portabilidade e reutilização do código, o compilador não adiciona semântica a RPO, ou seja, se numa determinada RPO os lugares significam diferentes tarefas a serem realizadas por uma entidade, representada por uma ficha, o sistema em JESS não passa a informação para esta entidade automaticamente sobre como a tarefa deve ser desempenhada. Por isso, é necessário extender o sistema da RPO compilado em JESS, adicionando regras que reflitam as particularidades das semânticas de cada RPO.



Fonte: [Yamane, 2006]

Figura C.1: Diagrama de classes do compilador RPO/JESS

Apêndice D

Regras JESS: Fundamentos da Estrutura da Informação

Apresenta-se o código JESS do estudo de caso da disciplina de Fundamentos da Estrutura da Informação.

```
(clear)


;;; ======== Class Templates

(deftemplate Student
(slot doing_Pb)
(slot bestgrade)
(slot busy (default FALSE))
(slot doing_IU)
(slot answer)
(slot doing_Curriculum)
(slot name)
(slot ID)
(multislot where)
(multislot report)
(multislot done)
)

;;; ======== Place Template and Place Instances

(deftemplate place
```

```
(slot name)
(slot type (default problem))
(multislot content))

(deffacts ped_places
(place (name bufUP4UP6) (type buffer))
(place (name bufUP6bufUP4UP6) (type buffer))
(place (name GhostPlace))
(place (name UP7))
(place (name UP2))
(place (name UP6))
(place (name UP3))
(place (name UP5))
(place (name UP1))
(place (name bufUP4bufUP4UP6) (type buffer))
(place (name UP4))
)

;;; ======== Transitions Templates and Transitions Instances


(deftemplate trans-1to1
(slot name)
(slot place-in1)
(slot place-out1)
(slot condition)(slot action))

(deftemplate trans-1to2
(slot name)
(slot place-in1)
(slot place-out1)
(slot place-out2)
(slot condition)(slot action))

(deftemplate trans-2to1
(slot name)
(slot place-in1)
(slot place-in2)
(slot place-out1)
(slot condition)(slot action))
```

```
(deffunction cond_tbufUP6bufUP4UP6 (?token)
(neq (member$ UP6 (fact-slot-value ?token done)) FALSE))

(deffunction act_tbufUP6bufUP4UP6 (?token)
(modify ?token (doing_Pb (Next_Problem ?token))))

(deffunction cond_tbufUP4UP6UP7 (?token)
(neq (member$ bufUP4UP6 (fact-slot-value ?token done)) FALSE))

(deffunction act_tbufUP4UP6UP7 (?token)
(modify ?token (doing_Pb (Next_Problem ?token))))

(deffunction cond_tbufUP4bufUP4UP6 (?token)
(neq (member$ UP6 (fact-slot-value ?token done)) FALSE))

(deffunction act_tbufUP4bufUP4UP6 (?token)
(modify ?token (doing_Pb (Next_Problem ?token))))

(deffunction cond_tUP3UP4UP5 (?token)
(neq (member$ UP3 (fact-slot-value ?token done)) FALSE))

(deffunction act_tUP3UP4UP5 (?token)
(modify ?token (doing_Pb (Next_Problem ?token))))

(deffunction cond_tUP5UP6 (?token)
(neq (member$ UP5 (fact-slot-value ?token done)) FALSE))

(deffunction act_tUP5UP6 (?token)
(modify ?token (doing_Pb (Next_Problem ?token))))

(deffunction cond_tbufUP4UP6 (?token)
TRUE)

(deffunction act_tbufUP4UP6 (?token)
(modify ?token (done (insert$ (fact-slot-value ?token done) 1 bufUP4UP6))))

(deffunction cond_tUP2UP3 (?token)
(neq (member$ UP2 (fact-slot-value ?token done)) FALSE))

(deffunction act_tUP2UP3 (?token)
(modify ?token (doing_Pb (Next_Problem ?token))))
```

```
(deffunction cond_tUP1UP2 (?token)
(neq (member$ UP1 (fact-slot-value ?token done)) FALSE))

(deffunction act_tUP1UP2 (?token)
(modify ?token (doing_Pb (Next_Problem ?token))))


(deffacts ped_transitions
(trans-1to1 (name tbufUP6bufUP4UP6)
(place-in1 UP6)
(place-out1 bufUP6bufUP4UP6)
(condition cond_tbufUP6bufUP4UP6)
(action act_tbufUP6bufUP4UP6))

(trans-1to1 (name tbufUP4UP6UP7)
(place-in1 bufUP4UP6)
(place-out1 UP7)
(condition cond_tbufUP4UP6UP7)
(action act_tbufUP4UP6UP7))

(trans-1to1 (name tbufUP4bufUP4UP6)
(place-in1 UP4)
(place-out1 bufUP4bufUP4UP6)
(condition cond_tbufUP4bufUP4UP6)
(action act_tbufUP4bufUP4UP6))

(trans-1to2 (name tUP3UP4UP5)
(place-in1 UP3)
(place-out1 UP4)
(place-out2 UP5)
(condition cond_tUP3UP4UP5)
(action act_tUP3UP4UP5))

(trans-1to1 (name tUP5UP6)
(place-in1 UP5)
(place-out1 UP6)
(condition cond_tUP5UP6)
(action act_tUP5UP6))

(trans-2to1 (name tbufUP4UP6)
```

```
(place-in1 bufUP6bufUP4UP6)
(place-in2 bufUP4bufUP4UP6)
(place-out1 bufUP4UP6)
(condition cond_tbufUP4UP6)
(action act_tbufUP4UP6))

(trans-1to1 (name tUP2UP3)
(place-in1 UP2)
(place-out1 UP3)
(condition cond_tUP2UP3)
(action act_tUP2UP3))

(trans-1to1 (name tUP1UP2)
(place-in1 UP1)
(place-out1 UP2)
(condition cond_tUP1UP2)
(action act_tUP1UP2))

)

;;; ======== Transitions Rules

(defrule rule-trans-1to1
;;token Student
?token <- (Student (where $?token-where))

;;Lugares de entrada
?in1 <- (place (name ?place-in1) (content $?conts1&~nil))

;;Lugares de saída
?out1 <- (place (name ?place-out1))

;;a transição
(trans-1to1 (place-in1 ?place-in1)(place-out1 ?place-out1)
(condition ?cnd) (action ?act))

;;testa se o token está em todos os lugares acima
(test (member$ ?token $?conts1))

;;testa a condição especificada
(test (apply ?cnd ?token))
```

```
=>
(bind $?temp (create$ ?token))

;;cria lista com todos os lugares de entrada
(bind $?ins (create$ ?place-in1))

;;cria lista com todos os lugares de saída
(bind $?outs (create$ ?place-out1))

;;modifica o conteudo de ?in(x) para ficar com a lista de contents,
;; menos o token q foi retirado agora
(modify ?in1 (content (complement$ $?temp $?conts1)))

;;retira do where do token, o(s) lugar(es) de entrada
(modify ?token (where (complement$ $?ins (fact-slot-value ?token where))))

;;coloca no where do token, o(s) lugar(es) de saída
(modify ?token (where (insert$ (fact-slot-value ?token where) 1 $?outs)))

;;pega o conteudo dos lugar(es) de saida
;;se o conteudo não existir, cria, senão, insere o token no final da lista
(bind $?old_cont (fact-slot-value ?out1 content))
(if (or (eq $?old_cont nil) (eq $?old_cont (create$ nil))) then
   (modify ?out1 (content $?temp))
else
   (modify ?out1 (content (insert$ $?temp 2 $?old_cont))))


;;coloca o busy do token como FALSE, para poder disparar
(modify ?token (busy FALSE))

;;Aplica ação
(apply ?act ?token))

(defrule rule-trans-1to2
;;token Student
?token <- (Student (where $?token-where))

;;Lugares de entrada
?in1 <- (place (name ?place-in1) (content $?conts1&~nil))
```

```
;;Lugares de saída
?out1 <- (place (name ?place-out1))
?out2 <- (place (name ?place-out2))

;;a transição
(trans-1to2 (place-in1 ?place-in1)(place-out1 ?place-out1)
(place-out2 ?place-out2) (condition ?cnd) (action ?act))

;;testa se o token está em todos os lugares acima
(test (member$ ?token $?conts1))

;;testa a condição especificada
(test (apply ?cnd ?token))
=>
(bind $?temp (create$ ?token))

;;cria lista com todos os lugares de entrada
(bind $?ins (create$ ?place-in1))

;;cria lista com todos os lugares de saída
(bind $?outs (create$ ?place-out1 ?place-out2))

;;modifica o conteudo de ?in(x) para ficar com a lista de contents,
;;menos o token q foi retirado agora
(modify ?in1 (content (complement$ $?temp $?conts1)))

;;retira do where do token, o(s) lugar(es) de entrada
(modify ?token (where (complement$ $?ins (fact-slot-value ?token where))))

;;coloca no where do token, o(s) lugar(es) de saída
(modify ?token (where (insert$ (fact-slot-value ?token where) 1 $?outs)))

;;pega o conteudo dos lugar(es) de saida
;;se o conteudo não existir, cria, senão, insere o token no final da lista
(bind $?old_cont (fact-slot-value ?out1 content))
(if (or (eq $?old_cont nil) (eq $?old_cont (create$ nil))) then
   (modify ?out1 (content $?temp))
else
   (modify ?out1 (content (insert$ $?temp 2 $?old_cont))))

(bind $?old_cont (fact-slot-value ?out2 content))
```

```
(if (or (eq $?old_cont nil) (eq $?old_cont (create$ nil))) then
   (modify ?out2 (content $?temp))
else
   (modify ?out2 (content (insert$ $?temp 2 $?old_cont))))


;;coloca o busy do token como FALSE, para poder disparar
(modify ?token (busy FALSE))

;;Aplica ação
(apply ?act ?token))

(defrule rule-trans-2to1
;;token Student
?token <- (Student (where $?token-where))

;;Lugares de entrada
?in1 <- (place (name ?place-in1) (content $?conts1&~nil))
?in2 <- (place (name ?place-in2) (content $?conts2&~nil))

;;Lugares de saída
?out1 <- (place (name ?place-out1))

;;a transição
(trans-2to1 (place-in1 ?place-in1)(place-in2 ?place-in2)
(place-out1 ?place-out1) (condition ?cnd) (action ?act))

;;testa se o token está em todos os lugares acima
(test (member$ ?token $?conts1))
(test (member$ ?token $?conts2))

;;testa a condição especificada
(test (apply ?cnd ?token))
=>
(bind $?temp (create$ ?token))

;;cria lista com todos os lugares de entrada
(bind $?ins (create$ ?place-in1 ?place-in2))

;;cria lista com todos os lugares de saída
(bind $?outs (create$ ?place-out1))
```

```
;;modifica o conteudo de ?in(x) para ficar com a lista de contents, menos o token q
(modify ?in1 (content (complement$ $?temp $?conts1)))
(modify ?in2 (content (complement$ $?temp $?conts2)))

;;retira do where do token, o(s) lugar(es) de entrada
(modify ?token (where (complement$ $?ins (fact-slot-value ?token where))))

;;coloca no where do token, o(s) lugar(es) de saída
(modify ?token (where (insert$ (fact-slot-value ?token where) 1 $?outs)))

;;pega o conteudo dos lugar(es) de saida
;;se o conteudo não existir, cria, senão, insere o token no final da lista
(bind $?old_cont (fact-slot-value ?out1 content))
(if (or (eq $?old_cont nil) (eq $?old_cont (create$ nil))) then
   (modify ?out1 (content $?temp))
else
   (modify ?out1 (content (insert$ $?temp 2 $?old_cont))))


;;coloca o busy do token como FALSE, para poder disparar
(modify ?token (busy FALSE))

;;Aplica ação
(apply ?act ?token))

/* Classe que agrega todos os relatorios do q o estudante fez */

(deftemplate PB_Report
  (slot name_Curriculum)
  (slot name_Pb)
  (slot student_id)
  (multislot rep_Exp)
  (multislot rep_Exa)
  (multislot rep_Exe)
)

/*
   Classes de relatório
*/
(deftemplate Exe_Report
```

```
  (slot name_Pb)
  (slot student_id)
  (slot count_views (default 0))
  (slot grade (default 0.0))
  (slot name)
)

(deftemplate Exa_Report
  (slot name_Pb)
  (slot student_id)
  (slot count_views (default 0))
  (slot name)
)

(deftemplate Exp_Report
  (slot name_Pb)
  (slot student_id)
  (slot count_views (default 0))
  (slot name)
)

;;variavel global - objeto de comunicação com o mundo externo ao jess
(defglobal ?*jessComm* = null)

;;importando lib
(import br.ufsc.MathTutor.JessComm.*)

;;definição de template da classe do java bean usado para comunicação assincrona
(defclass JessCommBean OpnStudentJessCommBean)

;;variavel global - bean de comunicação assincrona
(defglobal ?*jessCommBean* = null)

;;regra - se um student estiver em um lugar de problema, e ainda não fez
;;o problema (done), dispara a função de interface
;;a não ser que o lugar seja Begin (não deve fazer nada) ou End

(defrule place
?in <- (place (name ?place-in & ~Begin & ~End) (content $?conts & ~nil) (type proble
?token <- (Student (done $?done) (where $?where) (doing_Pb ?place-in) (busy FALSE))
```

```
;;testa pra ver se o lugar pertence ao where do Student
(test (neq (member$ ?place-in $?where) FALSE))

;;testa para ver se o lugar não pertence ao done do Student
(test (eq (member$ ?place-in $?done) FALSE))
=>

;;indica que não eh para disparar com esse token de novo
(modify ?token (busy TRUE))

;;chama função interface
(interface ?token)

)

;;regra - se um student estiver em no lugar Begin

(defrule placeBegin
?in <- (place (name Begin) (content $?conts & ~nil) (type problem))
?token <- (Student (done $?done) (where $?where) (doing_Pb Begin) (busy FALSE))

;;testa pra ver se o lugar pertence ao where do Student
(test (neq (member$ Begin $?where) FALSE))

;;testa para ver se o lugar não pertence ao done do Student
(test (eq (member$ Begin $?done) FALSE))
=>

;;indica que não eh para disparar com esse token de novo
(modify ?token (busy TRUE))

;;coloca Begin no done deste token
(if (or (eq $?done nil) (eq $?done (create$ nil))) then
   (modify ?token (done (create$ Begin)))
else
   (modify ?token (done (insert$ $?done 1 Begin)))
)

)
;;regra - se uma resposta do estudante for halt, pára tudo e salva
```

```
(defrule answerHalt
?token <- (Student (ID ?id) (doing_Pb ?pb) (answer halt))

=>

;;chama função de "salvamento"
(saveStudent ?token)

;;remove o ?token da rede e todos os objetos relacionados

;;roda queries de busca de objetos relacionados a Reports
;;e retira-os da base de conhec

(bind ?it (run-query getExe_Report ?id ?pb))
(while (?it hasNext)
   (bind ?tok (call ?it next))
   (bind ?rep (call ?tok fact 1))
   (retract ?rep)
)

(bind ?it (run-query getExa_Report ?id ?pb))
(while (?it hasNext)
   (bind ?tok (call ?it next))
   (bind ?rep (call ?tok fact 1))
   (retract ?rep)
)

(bind ?it (run-query getExp_Report ?id ?pb))
(while (?it hasNext)
   (bind ?tok (call ?it next))
   (bind ?rep (call ?tok fact 1))
   (retract ?rep)
)

(bind ?it (run-query getPB_Report ?id ?pb))
(while (?it hasNext)
   (bind ?tok (call ?it next))
   (bind ?rep (call ?tok fact 1))
   (retract ?rep)
)
```

```
;;e por fim, retira o proprio student da base
(retract ?token)

;;manda msg para jessComm
(call ?*jessComm* send (create$ ?id ?pb))
)

;;queries que buscam os ExX_Reports de um Student
(defquery getExe_Report (declare (variables ?stu_id ?pb))
   (Exe_Report (student_id ?stu_id) (name_Pb ?pb))
)

(defquery getExa_Report (declare (variables ?stu_id ?pb))
   (Exa_Report (student_id ?stu_id) (name_Pb ?pb))
)

(defquery getExp_Report (declare (variables ?stu_id ?pb))
   (Exp_Report (student_id ?stu_id) (name_Pb ?pb))
)

;;query que busca os PB_Report de um student
(defquery getPB_Report (declare (variables ?stu_id ?pb))
   (PB_Report (student_id ?stu_id) (name_Pb ?pb))
)

;;regra - se uma resposta do estudante for done, coloca o lugar onde o
;;token se encontra no campo de done

(defrule answerDone
?token <- (Student (ID ?id) (doing_Pb ?pb) (answer done) (done $?done))

=>

;;verifica se o campo done do token está vazio ou não
(if (or (eq $?done nil) (eq $?done (create$ nil))) then
   (modify ?token (done (create$ ?pb)))
else
   (modify ?token (done (insert$ $?done 1 ?pb)))
)

;;coloca resposta como nil
```

```
(modify ?token (answer nil))

)

;;regra - verifica o estado da flag no objeto javabean, se for 1, quer
;;dizer que pode pegar uma resposta
;;
;;IMPORTANTE: o slot ID do token Student deve ser uma String

(defrule answer
?bean <- (JessCommBean (flag 1) (ID ?id))
?token <- (Student (ID ?id) (doing_Pb ?doing))
=>
;;pega o objeto
(bind ?obj (fact-slot-value ?bean OBJECT))

;;modifica o campo answer do Student
(modify ?token (answer (call ?obj getValueAnswer)))

;;se a resposta for done (problema terminado), ou halt (sair)
;;pega os objetos reports
(if (or (eq (fact-slot-value ?token answer) done)
        (eq (fact-slot-value ?token answer) halt)) then
   (bind $?reps (call ?obj getValueReports))

   ;;verifica se tem um report
   (if (and (neq $?reps nil) (neq $?reps (create$ nil))) then

      ;;verifica se jah tem reports, insere ou cria
      (if (or (eq (fact-slot-value ?token report) nil)
              (eq (fact-slot-value ?token report) (create$ nil)) ) then
         (modify ?token (report $?reps))
      else
         (modify ?token (report (insert$ (fact-slot-value ?token report) 1 $?reps)))
      )
   )
)

;;muda a flag, sinalizando q esta resposta jah foi "consumida"
(?obj setFlag 0)
)
```

```
;;regra para inserir novo token Student na rdp

(defrule newToken
?bean <- (JessCommBean (flag 2) (name ?name) (ID ?id))
=>
;;pega o objeto bean
(bind ?obj (fact-slot-value ?bean OBJECT))

;;chama os metodos que devolvem um tipo jess.Value ATOM
(bind ?doing_IU (call ?obj getValueDoing_IU))
(bind ?doing_Pb (call ?obj getValueDoing_Pb))
(bind ?doing_Curriculum (call ?obj getValueDoing_Curriculum))
(bind ?answer (call ?obj getValueAnswer))

;;metodo retorna jess.Value tipo INTEGER
(bind ?bestgrade (call ?obj getValueBestGrade))

;;metodos devolvem jess.Value tipo LIST
(bind $?where (call ?obj getValueWhere))
(bind $?done (call ?obj getValueDone))
(bind $?reps (call ?obj getValueReports))

;;insere o token Student na base de dados
(bind ?stu (assert (Student (ID ?id) (name ?name) (doing_IU ?doing_IU)
    (doing_Pb ?doing_Pb)
                           (doing_Curriculum ?doing_Curriculum)
    (answer ?answer)
                           (bestgrade ?bestgrade) (report $?reps)
                           (where $?where) (done $?done) )))

;;sinaliza objeto bean que os dados jah foram consumidos
(?obj setFlag 0)

;;verifica onde o token deve estar (slot where) e atualiza
;;os fatos conteudos (content) dos lugares
(foreach ?whe $?where

   ;;roda uma query e pega todo(s) lugar(es)
   (bind ?it (run-query getPlace ?whe))
   (while (?it hasNext)
```

```
      ;;pega um lugar
      (bind ?jesstoken (call ?it next))
      (bind ?pl (call ?jesstoken fact 1))

      ;;pega o conteudo deste lugar
      (bind $?contents (fact-slot-value ?pl content))

      ;;se for nil, cria, senão insere
      (if (or (eq $?contents nil) (eq $?contents (create$ nil))) then
         (modify ?pl (content (create$ ?stu)))
      else
         (modify ?pl (content (insert$ $?contents 1 ?stu)))
      )
   )
)
)
;;query que busca o lugar de nome especificado

(defquery getPlace (declare (variables ?X))
   (place (name ?X))
)

;;função interface
;;na verdade, só envia para o jessComm uma msg, e depois uma das regras
;;acima se encarrega de "entregar" a resposta

(deffunction interface (?stu)
   (bind ?name (fact-slot-value ?stu name))
   (bind ?id (fact-slot-value ?stu ID))
   (bind ?curriculum (fact-slot-value ?stu doing_Curriculum))
   (bind ?pb (fact-slot-value ?stu doing_Pb))
   (bind ?iu (fact-slot-value ?stu doing_IU))

   (printout t "================= Interface ================" crlf)
   (printout t "Estudante " ?name " fazendo " ?iu " no problema " ?pb  crlf)

   (call ?*jessComm* ask ?id ?name ?curriculum ?pb)
)

;;função de salvar um Student
(deffunction saveStudent (?token)
```

```
;;pega os campos que serão salvos

(bind ?id (fact-slot-value ?token ID))
(bind ?name (fact-slot-value ?token name))
(bind ?doing_Curriculum (fact-slot-value ?token doing_Curriculum))
(bind ?doing_Pb (fact-slot-value ?token doing_Pb))
(bind ?doing_IU (fact-slot-value ?token doing_IU))
(bind ?bestgrade (fact-slot-value ?token bestgrade))
(bind $?where (fact-slot-value ?token where))
(bind $?done (fact-slot-value ?token done))
(bind $?reports (fact-slot-value ?token report))

;;chama o proc de gravar
(call ?*jessComm* save ?id ?name ?doing_Curriculum ?doing_Pb ?doing_IU ?bestgrade

;;pega a lista de relatorios Exe_Reports
(bind ?it (run-query getExe_Report ?id ?doing_Pb))
(if (?it hasNext) then
   (bind ?tok (call ?it next))
   (bind ?rep (call ?tok fact 1))
   (bind $?names (create$ (fact-slot-value ?rep name)))
   (bind $?grades (create$ (fact-slot-value ?rep grade)))
   (bind $?views (create$ (fact-slot-value ?rep count_views)))

   (while (?it hasNext)
      (bind ?tok (call ?it next))
      (bind ?rep (call ?tok fact 1))
      (bind $?names (insert$ $?names 1 (fact-slot-value ?rep name)))
      (bind $?grades (insert$ $?grades 1 (fact-slot-value ?rep grade)))
      (bind $?views (insert$ $?views 1 (fact-slot-value ?rep count_views)))
   )

   ;;salva
   (call ?*jessComm* saveReport ?id Exercise ?doing_Pb $?names $?grades $?views)
)

;;pega a lista de relatorios Exa_Reports
(bind ?it (run-query getExa_Report ?id ?doing_Pb))
(if (?it hasNext) then
   (bind ?tok (call ?it next))
```

```
      (bind ?rep (call ?tok fact 1))
      (bind $?names (create$ (fact-slot-value ?rep name)))
      (bind $?grades (create$ 0))
      (bind $?views (create$ (fact-slot-value ?rep count_views)))

      (while (?it hasNext)
         (bind ?tok (call ?it next))
         (bind ?rep (call ?tok fact 1))
         (bind $?names (insert$ $?names 1 (fact-slot-value ?rep name)))
         (bind $?grades (insert$ $?grades 1 0))
         (bind $?views (insert$ $?views 1 (fact-slot-value ?rep count_views)))
      )

      ;;salva
      (call ?*jessComm* saveReport ?id Example ?doing_Pb $?names $?grades $?views)
   )

   ;;pega a lista de relatorios Exp_Reports
   (bind ?it (run-query getExp_Report ?id ?doing_Pb))
   (if (?it hasNext) then
      (bind ?tok (call ?it next))
      (bind ?rep (call ?tok fact 1))
      (bind $?names (create$ (fact-slot-value ?rep name)))
      (bind $?grades (create$ 0))
      (bind $?views (create$ (fact-slot-value ?rep count_views)))

      (while (?it hasNext)
         (bind ?tok (call ?it next))
         (bind ?rep (call ?tok fact 1))
         (bind $?names (insert$ $?names 1 (fact-slot-value ?rep name)))
         (bind $?grades (insert$ $?grades 1 0))
         (bind $?views (insert$ $?views 1 (fact-slot-value ?rep count_views)))
      )

      ;;salva
      (call ?*jessComm* saveReport ?id Explanation ?doing_Pb $?names $?grades $?view
   )

)

;;função de escolha de próxima problema a ser feito -
```

```
;;por enquanto, escolhe qquer um que não tenha sido feito

(deffunction Next_Problem (?token)
   (bind $?where (fact-slot-value ?token where))
   (bind $?done (fact-slot-value ?token done))
   (bind $?next_choices (complement$ $?done $?where))

   ;;procura um lugar que não seja de buffer
   (foreach ?pl $?next_choices
      ;;roda uma query e pega o lugar
      (bind ?it (run-query getPlace ?pl))
      (while (?it hasNext)
         ;;pega um lugar
         (bind ?jesstoken (call ?it next))
         (bind ?place (call ?jesstoken fact 1))

         ;;pega o tipo deste lugar
         (bind ?type (fact-slot-value ?place type))

         ;;se for problem, retorna este lugar
         (if (eq ?type problem) then
            (return ?pl)
         )
      )
   )

   ;;se chegou ateh aqui, não tem lugar de problema, retorna nil
   (return nil)

)

;;seta estrategia de disparo para fifo
(set-strategy breadth)
```

Apêndice E

Histórico da Informática na Educação no Brasil

O computador foi introduzido na educação brasileira por meio de universidades − públicas, especialmente − nos anos 50, inicialmente, como ferramenta auxiliar da pesquisa técnico-científica e, a partir da década de 60, na organização administrativa do ensino superior. Nesse período, houve diversos projetos, porém não chegaram ao sistema público de ensino fundamental e médio, permanecendo no campo experimental em universidades, secretarias de educação e escolas técnicas Ministério da Educação.

A Informática na Educação teve início, no Brasil, com a criação, pela Secretaria Especial de Informática − SEI, da comissão Especial n ˚ 01: Informática na Educação : CE-IE, em 1980 FUNTEVE. Nos anos de 1981 e 1982, foram realizados o primeiro e o Segundo Seminário de Informática na Educação, dos quais surgiu o Projeto Brasileiro de Informática na Educação − EDUCOM − em julho de 1983, com o objetivo de realizar estudos e experiências nesse setor visando formar recursos humanos para ensino e pesquisa e criar programas computacionais através de equipes multidisciplinares. Com esse objetivo, a SEI (Comunicado SEI/SS n.º 015/83) solicitou às universidades propostas para a criação de centros-piloto do EDUCOM, selecionados, em dezembro do mesmo ano, os projetos das universidades federais do Rio Grande do Sul (UFRGS), Pernambuco FPe), Rio de Janeiro (UFRJ), Minas Gerais (UFMG) e Universidade estadual de Campinas (UNICAMP). A Informática na Educação tem se desenvolvido e com o advento da internet, onde se concentram as pesquisas atualmente, permite que as ferramentas, antes utilizadas em locais específicos, estejam disponíveis nos mais longínquos pontos geográficos permitindo a difusão do conhecimento. As bases legais da Educação à distância no Brasil foram estabelecidas pela Lei de Diretrizes e bases da Educação Nacional[1].

[1](Lei n.º 9394, de 20 de dezembro de 96), pelo Decreto n.º 2494, de 10 de fevereiro de 1998 publicado no D.O.U. de 11/02/98), Decreto n.º 2561, de 27 de abril de 1998 (publicado no D.O.U. de 28/04/98) e pela Portaria ministerial n.º 301, de 07 de abril de 1998 (publicada no D.O.U. de 09/04/98)

Por vezes confunde-se Ensino à Distância com ensino através da *Web*, mas segundo o *Decreto 2.494, de 10.02.1998*, "Educação a Distância é uma forma de ensino que possibilita a auto-aprendizagem, com a mediação de recursos didáticos sistematicamente organizados, apresentados em diferentes suportes de informação, utilizados isoladamente ou combinados, e veiculados pelos diversos meios de comunicação". O Departamento de Informática na Educação a Distância - DEIED, ligado ao MEC MEC compõe a estrutura da Secretaria de Educação a Distância - SEED suas competências constam do artigo 21 do Decreto nº 4.637, e 21 de março de 2003, a saber:

- planejar e coordenar ações visando a execução de projetos de informática educacional;
- fomentar o desenvolvimento da infra-estrutura de suporte na área de informática junto aos sistemas de ensino nos Estados, municípios e Distrito Federal;
- apoiar o desenvolvimento de tecnologias de informática e a a utilização pelo ensino fundamental, médio e superior e na educação especial;
- realizar estudos e pesquisas visando conhecer a produção nacional e estrangeira na área de informática, voltados para o ensino a distância, em seus diferentes níveis;
- promover o desenvolvimento de pesquisas sobre programas de informática educativa.

O DEIED desenvolve as seguintes grandes ações MEC:

1. implementação, acompanhamento e avaliação das atividades vinculadas ao Programa Nacional de Informática na Educação − ProInfo;
2. desenvolvimento do ambiente digital de aprendizagem − e-ProInfo;
3. Gerenciamento das ações do Centro Experimental de Tecnologia educacional − CETE;
4. projetos de cooperação internacional como o RIVED e o WebEduc, brevemente descritos a seguir.

- RIVED A Rede Internacional Virtual de Educação (RIVED) constitui um projeto piloto de cooperação internacional na América latina. O RIVED, uma iniciativa com o propósito de melhorar o ensino de ciências e Matemática no ensino médio, aproveita o potencial das tecnologias de Informática e da Comunicação. O programa envolve o projeto instrucional de atividades de ensino/aprendizagem, a produção de material pedagógico multimídia, capacitação de pessoal, rede de distribuição de informação, e estratégias de avaliação da aprendizagem e do programa.
- Webeduc É um fórum sobre as novas tecnologias na educação. O objetivo é apoiar projetos de cooperação e de intercâmbio em educação entre a França e o Brasil, nomeadamente na área das tecnologias da Informação e da Comunicação mas também na área do ensino da língua francesa e portuguesa.

Um projeto de EAD é a Universidade Virtual Pública do Brasil, UniRede, um consórcio de 70 instituições públicas de ensino superior cujo objetivo leva em conta democratizar o acesso à educação de qualidade por meio da oferta de cursos à distância nos níveis de graduação, Pós-Graduação, Extensão ou Educação Continuada, o que possibilitou a cooperação entre universidades e escolas técnicas, evitando o isolamento e duplicidade entre suas iniciativas. Entre outros avanços, também desobrigou o pagamento de direitos autorais pela disseminação de metodologias, tecnologias e conteúdos elaborados nas instituições. Embora a data oficial de formação do consórcio da UniRede tenha do o dia 23 de agosto de 2000, quando o Termo de Adesão foi ensinado por representantes de dezenas de instituições, a criação Universidade Virtual Pública do Brasil teve seus primórdios em várias ações anteriores UniRede.

Referências Bibliográficas

ABED. "Associação Brasileira de Educação a Distância". URL `http://www2.abed.org.br/`. Acesso em março 2006.

Abelson, H. e Sussman, G. J. *Structure and Interpretation of Computer Program*. The Mit Press, 1996.

Abrantes, P. *Epistemologia e cognição*. Editora da UNB, Brasília, 1993.

Ainsworth, S. e Fleming, P. "Evaluating authoring tools for teachers as instructional designers". *Computers in Human Behavior*, pp. 131–148, 2006.

Aleven, V., McLaren, B. M., Sewall, J., e Koedinger, K. "The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains". In Ikeda, M., Ashley, K., e Chan, T., editors, *Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS 2006)*, pp. 61–70, Berlin, 2006. Springer Verlag.

Auberger, M. "Student Modeling in Educational Multimedia Titles Using Agents". Technical report, http://aif.wu-wien.ac.at/usr/geyers/, 1998. Acesso em: 18 maio 2000.

Bellifemine, F., Caire, G., Rimassa, G., Poggi, A., Trucco, T., Lhuillier, N., Picault, J., Cortese, E., Quarta, F., e Vitaglione, G. "JADE (Java Agent DEvelopment Framework". Technical report, CSELT ; Motorola Labs ; University of Parma, http://sharon.cselt.it/projects/jade/, 2004.

Berthomieu, B., Ribet, P., e Vernadat, F. "The tool TINA - Construction of abstract state spaces for petri nets and time petri nets". *International Journal of Production Research*, 42(14):2741–2756, 2004.

Bica, F., Silveira, R., e Viccari, R. "ELETROTUTOR III: Uma abordagem Multiagentes.". *IX Simpósio Brasileiro de Informática na Educação/SBIE*, 1998.

Bittencourt, G. "Fundamentos da Estrutura da Informação, Florianópolis: UFSC". http://www.lcmi.ufsc.br/ gb/fei/ Acesso em: 10 julho 2000, 1998a.

Bittencourt, G. *Inteligência artificial:ferramentas e teorias*. Editora da UFSC, Florianópolis, 1998b.

Botafogo, R. e Moss, D. "The MORENA Model for Hypermedia Authoring and Browsing". In *Proc. of the Int. Conf. on Multimedia Computing and Systems*, pp. 42–49, 1995.

Bra, P. D., Aerts, A., Berden, B., e Lange, B. D. "Escape from the Tyranny of the Textbook: Adaptive Object Inclusion in AHA!". In *World Conference on E-Learning in Corporate Government*, pp. 7–11, 2003.

Brusilovsky, P. "Adaptative Hypermedia: From Intelligent Tutoring Systems to Web-Based Education". *Lecture Notes in Computer Science*, 1839:1–7, June 2000. 5th International Conference, ITS 2000, Montreal, Canada.

Brusilovsky, P. "Adaptative Hypermedia". In *User Modeling and User-Adapted Interaction*, number 11 in 1/2, pp. 87–110. Kluwer Academic Publishers, 2001.

Brusilovsky, P. e Peylo, C. "Adaptative and Intelligent Web-based Educational Systems". In *IJAIED: Int. Journal of Artificial Intelligence in Education*, volume 13, pp. 159–172. IOS Press, 2003.

Brusilovsky, P. e Vassileva, J. "Course sequencing techniques for large-scale webbased education". In *International Journal of Cont. Engineering Education and Lifelong Learning*, volume 13, pp. 75–94. Inderscience, 2003.

Cardoso, J., Bittencourt, G., Frigo, L. B., e Pozzebon, E. "Petri Nets for authoring mechanisms". In *XV Simp´osio Brasileiro de Inform´atica na Educação*, pp. 378–387, 2004a.

Cardoso, J., Bittencourt, G., Frigo, L. B., Pozzebon, E., e Postal, A. "MathTutor: A multiagent intelligent tutoring systems". In *1st IFIP Conf. on AI Applications and Innovations -WCC´04*, pp. 22–27, 2004b.

Cardoso, J. e Valette, R. *Redes de Petri*. Editora UFSC, 1997.

CGI. "Common Gateway Interface". URL `http://www.w3.org/CGI/`. Acesso em agosto 2006.

Chen, W. e Mizoguchi, R. *Cognitive Support for Learning - Imagining the Unknown*, capítulo Learner model ontology and learner model agent, pp. 189–200. IOS Press, 2004.

Clancey, W. "Use of MYCIN's rules for tutoring". In Buchanan, B. G. e Shortliffe, E. H., editors, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, capítulo 26, pp. 464–489. Addison-Wesley, MA, 1984.

Colmerauer, A. "Prolog in 10 figures". *Commun. ACM*, 28(12):1296–1310, 1985. ISSN 0001-0782. URL `http://doi.acm.org/10.1145/214956.214958`.

Corcho, ., M. Fernández-López, A. G.-P., e López-Cima, A. "Building Legal Ontologies with METHONTOLOGY and WebODE.". In *Law and the Semantic Web*, volume 3369, pp. 142–157, 2003.

Costa, A. C. P. L. e Bittencourt, G. "Dynamic Social Knowledge: A Comparative Evaluation". In *Proceedings of the International Joint Conference IBERAMIA'2000 (Ibero-American Artificial Intelligence Conference) SBIA'2000 (Brazilian Artificial Intelligence Symposium)*, São Paulo, Brazil, November 19-22 2000. Springer Verlag Lecture Notes in Artificial Intelligence.

Costa, E. B. *Um Modelo de Ambiente Interativo de Aprendizagem Baseado numa Arquitetura Multi-Agentes*. PhD thesis, Universidade Federal da Paraíba, Brasil, 1997.

Coutinho, L. R., Labidi, S., e Jr, G. S. "Groups Formation for Cooperative Learning: a Genetic Algorithm Approach". In *International Conference on Computers and Advanced Technology in Education (CATE'2001)*, Banff, Canada, June 27-29 2001.

Coutinho, L. R., Labidi, S., Jr, G. S., e Teixeira, C. "A Learner Modeling Agent for Cooperative Learning". In *Anais do Simpósio Brasileiro de Informática na Educação (SBIE-2000)*, Maceió, AL, Novembro 8-10 2000.

Cristea, A. "Evaluating Adaptive Hypermedia Authoring while Teaching Adaptive Systems". In *ACM Symposium on Applied Computing (SAC2004)*, pp. 929–934, 2004.

Cristea, A. e Kinshu. "Considerations on LAOS, LAG and their Integration in MOT". In *EdMedia 2003 Conference Proceedings, AACE,D. Lassner and C. McNaught*, pp. 511–518, 2003.

da C. Mora, M., Lopes, J. G. P., Vicari, R. M., e Coelho, H. "BDI Models and Systems: Bridging the Gap". In *ATAL '98: Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages*, pp. 11–27, London, UK, 1999. Springer-Verlag. ISBN 3-540-65713-4.

D'Amico, C. B., Pereira, A. S., Geyer, C. F. R., e Viccari, R. "Adapting Teaching Strategies in a Learning Environment on WWW". In *Proceedings of the WebNet World Conference of the WWW, Internet e Intranet*, Florida, USA, 1998.

de F. Faria, T. "Um ambiente interativo multiagentes para o ensino de estrutura da informação". Master's thesis, Universidade Federal da Santa Catarina, Brasil, 2001.

Demazeau, Y. e Müller, J. P. "Decentralized Artificial Intelligence". In *Proceedings of the 1st European Workshop on Modelling Autonomous Agents in a Multi-Agents World*, pp. 3–13, Cambridge, England, 1989. Elsevier Science.

do S. Da Silva, A. e Hernandez-Dominguez, A. "Uma modelagem baseada em agentes de um Tutor no contexto de uma Classe Virtual Adaptativa.". In *Workshop de Ambientes de Aprendizagem baseados em agentes - www.inf.ufpr.br/sbie99 - Publicação eletrônica*, Curitiba, Paraná, Novembro 1999.

Durfee, E. H. "A Unified Approach to Dynamic Coordination: Planning Actions and Interactions in a Distributed Problem Solving Network". In *COINS Technical Report*, pp. 84–87. Univ. of Massachusetts at Amherst, 1987.

Durfee, E. H. e Rosenschein, J. "Distributed problem solving and multiagent systems: Comparisons and examples". In Klein, M., editor, *Proceedings of the 13th International Workshop on DAI*, pp. 94–104, Lake Quinalt, WA, USA, 1994. URL citeseer.ist.psu.edu/article/durfee94distributed.html.

Eriksson, H. "Using JessTab to Integrate Protege and Jess". *IEEE Intelligent Systems*, 18 (2):43–50, 2003. ISSN 1541-1672. URL http://dx.doi.org/10.1109/MIS.2003.1193656.

Felleisen, M., Findler, R. B., Flatt, M., e Krishnamurthi, S. *How to Design Programs an Introduction to Computing and Programming*. The MIT Press, London, England, 2001.

Ferber, J. *Multi-Agent System, An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Publishers, 1999.

Ferber, J. e Gasser, L. "Intelligence Artificielle Distribué". In *Tutorial Notes of the 11th Conference on Expert Systems and their Applications*, 1991. Avignon'91, France.

Fields, D. e Kolb, M. *Desenvolvendo na Web com Java Server Pages*. Editora Ciência Moderna, Rio de Janeiro, 2000.

FIPA-OS. "FIPA-OS Agent Toolkit". URL http://sourceforge.net/projects/fipa-os/. Acesso em agosto 2006.

Franklin, S. e Graesse, A. "Is it an agent, or just a program? a taxonomy for autonomous agents.". In *Proceedings of the Third International Workshop on Agent Theories,Architecture and Languages*. Springer-Verlag, 1996.

Freitas, F. e Bittencourt, G. "Cognitive Multi-Agent Systems for Integrated Information Retrieval and Extraction over the Internet". In *Proceedings of the International Joint Conference IBERAMIA'2000 (Ibero-American Artificial Intelligence Conference)*, São Paulo, Novembro 2000. Springer Verlag Lecture Notes in Artificial Intelligence.

Friedman-Hill, E. "Jess, The Java Expert System Shell". Technical report, Livermore, CA, http://herzberg.ca.sandia.gov/jess/, 1997.

Frigo, L. B. "MathTutor-Um Ambiente Interativo Multiagente Para O Ensino De Estrutura Da Informação". Master's thesis, Universidade Federal de Santa Catarina, 2002.

Frigo, L. B., Cardoso, J., e Bittencourt, G. "A Method for Modeling Adaptive Interactions in Intelligent Tutoring Systems.". In *IJCEELL special issue on Integrating Intelligent and Adaptive Hypermedia Techniques in Web-Based Education Systems*. Inderscience, 2007. (aceito para publicação).

FUNTEVE. “Um Relato do Estado Atual da Informática no Ensino no Brasil”, 1985. URL http://www.mec.gov.br/seed/default.shtm. Acesso em novembro 2003.

Gasser, L. “Boundaries, identity and aggregation: Plurality issues in multiagent systems.”. In Werner, E. e Demazeu, Y., editors, *Decentralized Artificial intelligence*, pp. 199–212. Elsevier Science Publisher, 1992.

Giraffa, L. M. M. *Uma arquitetura de tutor utilizando estados mentais.* PhD thesis, Porto Alegre: CPGCC/UFRGS, 1999.

Gómez-Pérez, A. “Ontological Engineering: A state of the Art”. *British Computer Society*, 2(3), 1999.

Grimshaw, S., Ainsworth, S., e Underwood, J. “Using an its authoring tool to explore educators. Use of instructional strategies”. *5th International Conference ITS*, pp. 182–191, 2000.

Gruber, T. R. “A translation approach to portable ontology specifications”. *Knowledge Acquisition*, 5(2):199–220, 1993. ISSN 1042-8143. URL http://dx.doi.org/10.1006/knac.1993.1008.

Hix, D. e Hartson, H. *Developing User Interfaces*. Wiley, 1993. URL citeseer.nj.nec.com/hix93developing.html.

JATLite. “Java Agent Template Lite”. Technical report, Stanford University, http://java.stanford.edu/java_agent/html/index2.html, 1997.

JavaCC. “Java Compiler Compiler [tm]”. URL https://javacc.dev.java.net/. Acesso em novembro 2005.

Jennings, N. “Cooperation in Industrial Multi-Agent Systems.”. *World Scientific*, 1994.

Jennings, N. R., Wooldridge, M., e Kinny, D. “A Methodology for Agent-Oriented Analysis and Design”. In *Proc. 3rd Int Conference on Autonomous Agents (Agents-99)*, volume 28, Seattle, 1999.

Jonassen, D. e Wang, S. “The Physics Tutor: Integrating Hypertext and Expert Systems”. In *Journal of Educational Technology Systems*, volume 22, pp. 19–28, 1993.

Kearsley, G. *Artificial Intelligence and Instruction: Applications and Methods*. Addison-Wesley Publishing Company, 1987.

Klassner, F. “Introduction on Artificial Intelligence”. Crossroads 3.1, 1996.

Knezek, G. “Intelligent Tutoring Systems and ICAI”. The Computer Teacher, v.15, 1988.

Labidi, S., Silva, J. C. D., Coutinho, L. R., Costa, N., e de B. Costa, E. “Agent-Based Architecture for a Cooperative Learning Environment”. In *Anais do Simpósio Brasileiro de Informática na Educação (SBIE-2000)*, Maceió, AL, Novembro 8 -10 2000.

Lin, J., Sun, C., e Juang, J. "MathCAL-II Software for Practicing Mathematical Problem Solving". In *Proc. of the Int. Conference on Computers in Education*, pp. 1669–1675, 2004.

Liu, X.-Q., Wu, M., e Chen, J.-X. "Knowledge aggregation and navigation high-level Petri nets-based in e-learning". In *Proceedings International Conference on Machine Learning and Cybernetics*, volume 1, pp. 420– 425, 2002.

Mathnet. "Relatório Técnico 1999/2000". Technical report, LCMI, http://www.das.ufsc.br/mathnet/rel-00/, 2000.

MEC. "Ministério da Educação". URL `http://www.proinfo.mec.gov.br/`. Acesso em novembro 2003.

Microsystems, S. "JAVA", a. URL `http://java.sun.com/`. Acesso em agosto 2006.

Microsystems, S. "JAVA Servlet", b. URL `http://java.sun.com/products/servlet/`. Acesso em agosto 2006.

Ministério da Educação. "Programa Nacional de Informática na Educação". URL `http://www.mec.gov.br/seed/proinfo.shtm`. Acesso em dezembro 2003.

Minsky, M. *A Sociedade da Mente*. Francisco Alves, 1989.

Mizoguchi, R. "Tutorial on Ontological Engineering: Part 1: Introduction to Ontological Engineering.". *New Generation Comput.*, 21(2), 2003a.

Mizoguchi, R. "Tutorial on Ontological Engineering: Part 2: Ontology Development, Tools and Languages.". *New Generation Comput.*, 22(1), 2003b.

Mizoguchi, R. e Bourdeau, J. "Using Ontological Engineering to Overcome Common AI-ED Problems". *International Journal of AI in Education*, pp. 107–121, 2000.

Murray, T. "Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art". *Journal of Artificial Intelligence and Education*, 10:98–129, 1999.

Na, J. e Furuta, R. "Dynamic Documents: Authoring, Browsing, and Analysis. Using a High-Level Petri Net-Based Hypermedia System". In *Proc. of 10th ACM Symposium on Document Engineering (DocEng '01)*, pp. 38–47, New York,NY, November 2001. ACM PRESS.

Nakabayashi, K., Koike, Y., Maruyama, M., Touhei, H., Kato, Y., e Fukuhara, Y. "Architecture of an Intelligent Tutoring System on the WWW". *Proceedings of the 8th World Conference of the AIED Society*, August 1997.

Noy, N. F., Fergerson, R., e Musen, M. "The knowledge model of protege-2000: Combining interoperability and flexibility". In *Proc. of the 2th Int. Conf. on Knowledge Engineering and Knowledge Management*, pp. 17–32, 2000.

Nuzzo-Jones, G., Walonoski, J., Heffernan, N., e Livak, T. "The eXtensible Tutor Architecture: A New Foundations for ITS". *Proceedings of the 12th Artificial Intelligence In Education*, pp. 555–562, 2005.

Palazzo, L. A. M. "Sistemas de Hipermídia Adaptativa". In *Anais do XXII SBC*, 2002.

Park, O. "Functional Characteristics of Intelligent Computer-Assisted Instruction: Intelligent Features". *Educational Technology*, pp. 7–14, 1998.

Parunak, H. V. D. "Applications of Distributed Artificial Intelligence in Industry". In O'Hare, G. M. P. e Jennings, N. R., editors, *Foundations of Distributed AI*. John Wiley & Sons, 1995. URL `citeseer.nj.nec.com/parunak94applications.html`.

Pereira, L. M. "Inteligência Artificial: mito e ciência". *Revista Intelecto*, 5, 2002. Disponível em:www.geocities.com/revistaintelecto/iafm.html Acesso em: 29 de Setembro de 2002.

Postal, A. "Especificação do controle pedagógico em uma ferramenta de autoria para sistemas tutores inteligentes". Master's thesis, Universidade Federal da Santa Catarina, Brasil, 2004.

Pozzebon, E., Cardoso, J., Bittencourt, G., e Hanachi, C. "A Multi-agent Architecture for Group Learning". In *Proceedings of the International Conference e-Society*, volume 1, pp. 60–67. IADIS Press, Dublin - Irland, 2006.

Russell, S. e Norvig, P. *Artificial Intelligence: a Modern Approach*. Prentice Hall Series in Artificial Intelligence, 1995.

Self, J. *Artificial Intelligence and human learning: intelligent computer-aided instruction*. Chapman and Hall, 1988.

Self, J. "Theoretical foundations for intelligent tutoring systems". *Journal of Artificial Intelligence in Education*, 1(4):3–14, 1990.

Sherer, R. e Schlageter, G. *Multi-Agent Approach for Integration of Neural Networks and Expert Systems*. John Wiley e Sons, 1995.

Sherry, L. "Issues in Distance Learning". *International Journal of Educational Telecommunications*, 4(1):337–365, 1996.

Shneiderman, B. *Designing The User Interfaces: Strategies for Effective Human Computer Interaction*. Addisson-Wesley, 1992.

Shortliffe, E. H. *Computer-Based Medical Consultations: MYCIN*. American Elsevier, New York, 1976.

Sibertin-Blanc, C. "High-level Petri nets with data structures". In *European Workshop on Application and Theory of Petri Nets*, pp. 141–170, 1985.

Sichman, J. S., Demazeau, Y., e Boissier, O. "When can knowledge-based systems be called agents". Technical report, Anais do Simpósio Brasileiro de Inteligência Artificial, 1992.

Staab, S., Studer, R., Schnurr, H., e Sure, Y. "Knowledge Processes and Ontologies.". *IEEE Intelligent Systems*, 16(1):26–34, 2001.

Stanford. "Ontolingua", a. URL `http://www-ksl-svc.stanford.edu:5915/&service=frame-editor`. Acesso em setembro 2006.

Stanford. "Protégé", b. URL `http://protege.stanford.edu`. Acesso em março 2006.

Turing, A. M. *Computing machinery and intelligence*. Number 59. Mind, 1950.

UniRede. URL `http://www.unirede.br`. Acesso em dezembro 2003.

V. Aleven, J. Sewall, B. M. M. e Koedinger, K. R. "Rapid Authoring of Intelligent Tutors for Real-World and Experimental Use". In *ICALT '06: Proceedings of the Sixth IEEE International Conference on Advanced Learning Technologies*, pp. 847–851, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2632-2.

Vassileva, J., Deters, R., Geer, J., McCalla, G., Bull, S., e Kettel, L. "Lessons from Deploying I-Help". In *Proceedings International Conference on AI and Education*, Texas, 2001.

Vicari, R. M. e Giraffa, L. M. M. *Sociendades Artificiais: a nova fronteira da inteligência das máquinas*, capítulo Fundamentos dos Sistemas Tutores Inteligentes. Bookman, 2003.

Vignaux, G. "As ciências cognitivas: uma introdução". In por Maria Manuela Guimarães, T., editor, *Coleção Epistemologia e Sociedade*. Instituto Piaget, Lisboa, 1995.

W3C. "OWL", a. URL `http://www.w3.org/TR/2002/WD-owl-ref-20020729/`. Acesso em setembro 2006.

W3C. "RDF", b. URL `http://www.w3.org/RDF/`. Acesso em setembro 2006.

Warendorf, K. e Tan, C. "ADIS - An animated data structure intelligent tutoring system or Putting an interactive tutor on the WWW". *Proceedings of the 8th World Conference of the AIED Society*, August 1997.

Wenger, E. *Artificial Intelligence and Tutoring Systems*. Morgan Kaufmann Publishers, Los Altos, CA, 1987.

Willrich, R., Saqui-Sannes, P., P.Sénac, e Diaz, M. "Multimedia authoring with hierarchical timed stream Petri nets and Java". In *Multimedia Tools Appl.*, volume 16, pp. 7–27, Dordrecht (Holanda), 2002.

Yamane, E. E. "Solução de Problemas na Modelagem e Implementação de Ferramentas de Autoria para Construção de Tutores Inteligentes". Master's thesis, Universidade Federal da Santa Catarina, Brasil, 2006.

Zamberlam, A. O., Giraffa, L. M. M., e Mora, M. C. “Um editor para programação orientada á agentes BDI”. In *II Workshop sobre Ambientes de Aprendizagem Baseados em Agentes*, pp. 481–482, Maceió, 2000. SBC/UFAL. XI SBIE.

Zeus. “Zeus Agent Toolkit”. URL `http://sourceforge.net/projects/zeusagent/`. Acesso em agosto 2006.