UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas de São Carlos

TOOTEMA: UMA FERRAMENTA PARA A CONSTRUÇÃO DE SISTEMAS TUTORES INTELIGENTES EM MATEMÁTICA[1]

Ricardo Hasegawa

Dissertação apresentada ao Instituto de Ciências Matemáticas de São Carlos, Universidade de São Paulo como parte dos requisitos para a obtenção do Título de Mestre em Ciências - Área: Ciências de Computação e Matemática Computacional.

Área de Concentração: Inteligência Artificial

Orientadora:

Profª. Drª. Maria das Graças Volpe Nunes

São Carlos

1995

[1]Trabalho realizado com o auxílio do CNPq.

Era ele quem erguia casas
Onde antes só havia chão
Como um pássaro sem asas
Ele subia com as casas
Que lhe brotavam da mão

O Operário da Construção - Vinicius de Moraes.

Agradecimentos

À Profa. Maria das Graças Volpe Nunes, orientadora e amiga, por me incentivar a enfrentar desafios e ajudar a vencê-los.

Ao Prof. Odelar Leite Linhares por sua eterna dedicação ao ensino.

À minha família por todo amor e carinho recebidos.

Aos professores Agma, Alfredo Hamar, Andreucci, Carmo, Carolina, Chú, Cristina, Dolores, Sandra, Solange e Wagner pela amizade e ajuda durante todos estes anos.

Às secretárias da pós-graduação, Beth e Laura, pela simpatia, carinho, atenção e eficiência com que sempre me atenderam.

Aos funcionários do ICMSC: Ângelo, Carmem, Elien, Gislene, Jacques, Luciano, Maria, Rose, Sandra, Silvana e Sônia, pela colaboração durante todos os meus estudos realizados neste Instituto.

Aos amigos de república: Marquinhos, Mauro e Tchelo, pelo apoio, incentivo e companheirismo em todas as horas.

Aos amigos que direta ou indiretamente, contribuíram com o seu apoio e amizade durante estes anos em São Carlos, em especial: Ana (do Di), Aninha, Célia, Cidinho, Claudete, Claudia (Itautec), Claudinha, Cozin, Craveiro, Cybelle, Delamaro, Di, Jaqueline, Josiane, Lúcia, Marcão, Marisa, Neyva, Nilmara, Pastel, Paty, Renata, Roberta, Rô, Rosana, Sadao, Sandra, Sandrinha (Itautec), Stein, Taisa, Teresa, Titi, Vera e Zero.

Ao CNPq pelo apoio financeiro fornecido durante todo o desenvolvimento deste trabalho.

Índice

Lista de Figuras e Tabelas

Resumo

Neste trabalho é apresentada e discutida uma ferramenta que auxilia a construção de Sistemas Tutores Inteligentes - STIs - no domínio da Matemática, denominada TOOTEMA - TOOl (ferramenta) para Tutores de Ensino de MAtemática. O TOOTEMA é um sistema que tem por objetivo ajudar na tarefa de criar e gerenciar o Módulo do Domínio do ARQTEMA (uma arquitetura genérica para Sistemas Tutores em Matemática), acompanhando o autor (professor) na estruturação de grande quantidade de informações. Ao usuário-autor, a ferramenta de autoria fornece facilidades de organização do material instrucional, provê recursos de representação gráfica, visando a consistência e a qualidade do material, além de reduzir o tempo de construção de STI em subdomínios diferentes e, principalmente, de envolver mais diretamente o autor leigo na criação do sistema.

Abstract

This work presents and discusses a tool that helps building Intelligent Tutoring Systems - ITS - in the Mathematics domain. It is called TOOTEMA and is a system that helps with the task of creating and managing the domain module of ARQTEMA (a generic architecture of Tutoring Systems for Mathematics), allowing the author (teacher) to structure of a large quantity of information. The tool provides facilities to organize the instruction material and supplies resources for graphic representation, aiming to give consistency and quality. Besides these it reduces the time of building an ITS in different domains and, mainly, causes the lay author to get involved in the creation process.

Introdução

Todos sabem que a escola está em atraso com relação aos recentes progressos da tecnologia e a possíveis avanços pedagógicos proporcionados por essa tecnologia. Além disso, a situação se agrava com o aumento do número de alunos, resultado da explosão demográfica. De fato, o ensino está longe de satisfazer a demanda, quer qualitativa, quer quantitativa, da sociedade [CAS87].

A introdução de sistemas computacionais nas escolas pode apresentar diversas vantagens em relação à instrução convencional, desenvolvendo as habilidades cognitivas que proporcionam melhores meios de ensino e aprendizagem, e promovendo o aumento da produtividade, ensinando mais coisas a mais gente, em menos tempo [NUN93].

1.1 Considerações Gerais

A idéia do uso da informática no ensino teve seu início com o advento da "instrução programada". Esta idéia foi usada por Sidney L. Pressey, na década de 20, através do uso de aparatos mecânicos que pudessem oferecer a cada aluno, individualmente, os conhecimentos necessários que seriam assimilados de acordo com o ritmo ou a velocidade de

cada um, e a correção imediata das questões levantadas. Posteriormente, B.F. Skinner, psicólogo norte-americano, reviveu as antigas idéias de Pressey, no início de 1950, ao divulgar em um artigo suas "máquinas de ensinar", que usam o conceito de instrução programada [CAS87].

A sua proposta era apresentada na forma impressa, e nunca se tornou muito popular pelo fato de ser muito difícil a produção do material instrucional. Além disso, os materiais existentes não tinham nenhuma padronização, o que dificultava a sua disseminação. Com o advento do computador e a sua característica interativa, percebeu-se que a produção do material instrucional e a sua apresentação poderia ser realizada com grande flexibilidade, possibilitando uma participação mais ativa do aluno no processo instrucional. Esta interação poderia ser programada de modo a simular um diálogo entre o aluno e o autor do curso, criando um ambiente mais motivador para o aprendizado. Assim, na década de 60, diversos programas de instrução programada foram implementados no computador - nascia a Instrução Auxiliada por Computador ou *Computer-Aided Instruction*, também conhecida como CAI.

Todavia, este tipo de CAI não passava de um livro-texto programável, onde o aluno acompanhava uma seqüência linear imposta pelo autor do material instrucional, e foi denominado por Jaime R. Carbonell de *Ad-hoc Frame-Orienteded* **(AFO)** CAI [NWA90].

Posteriormente, tentando superar a limitação dos programas lineares, Norman A. Crowder, em 1959, propôs o "programa ramificado", conhecido também como programa crowderiano ou intrínseco. Este apresentava algumas diferenças em relação ao AFO-CAI. Os *frames* ou itens do programa ramificado ofereciam muito mais informações que o AFO-CAI. Após a apresentação da informação, o programa colocava uma questão e diversas opções para que o aluno escolhesse a correta. Cada alternativa remetia o aluno a uma página diferente do livro, onde encontrava uma justificativa de seu erro ou seu acerto. Nesta mesma página, o aluno encontrava a indicação da nova página a seguir [CAS87].

Entretanto, a manipulação de grande quantidade do material instrucional neste tipo de sistemas era muito complicada. Na tentativa de resolver este problema, em fins de 1960 e início de 1970, surgiu o Sistema Generativo, ou adaptativo. Neste sistema, o material instrucional e as soluções dos problemas eram gerados pelo computador, trazendo vantagens como a redução drástica de memória, a geração e apresentação de muitos problemas - para um desejado nível de dificuldade - conforme a necessidade do aluno, com melhoria na individualização e *feedback*. Contudo, este sistema ainda trazia a desvantagem de não representar explicitamente todo o conhecimento do aluno, e de não ser capaz de deduzir e planejar ações futuras.

Para suprir estas dificuldades, na década de 70, técnicas de **Inteligência Artificial (IA)** foram introduzidas nos sistemas CAI generativos, e estes evoluíram para os Sistemas Inteligentes de Instrução Assistida por Computador - *Intelligent Computer Assisted Instruction* ou ICAI - e atualmente chamados de **Sistemas Tutores Inteligentes** ou STIs [NWA90].

O STI, entre outras coisas, tem que ter uma "vocação educacional" no sentido mais amplo dessa expressão. Sua característica básica é ser capaz de saber **O QUÊ** ensinar, **PARA QUEM** ensinar e **COMO** ensinar. Os STIs fazem parte da categoria dos Sistemas de Interação Didática capazes de auxiliar na realização de uma tarefa. Estes são programas inteligentes que visam uma interação com alunos de forma a apoiar o sujeito menos experiente na aquisição de conhecimentos, auxiliando também no desempenho de tarefas complexas.

Portanto, os computadores e os *softwares* de ensino vieram criar uma situação pedagógica e cultural totalmente nova, podendo complementar muito bem os materiais instrucionais tradicionais e compartilhar com o professor a tarefa de ensinar.

Entretanto, deve ser ressaltado que o alto custo e a dificuldade de desenvolvimento constituem um limite na produção de STIs, e pouco se conhece sobre ferramentas de apoio para auxiliar na construção deste tipo de sistema.

1.2 Motivação e Objetivos

A fase mais recente das pesquisas de sistemas de ensino envolve a aplicação de técnicas de **IA** para produzir sistemas cada vez mais complexos e inteligentes. Muitos especialistas (por exemplo, engenheiros do conhecimento, psicólogos cognitivos, especialistas em modelagem do estudante e em processamento de linguagem natural) cooperam no projeto e criação destes sistemas.

Como consequência, os professores acabam não tendo um envolvimento direto e poucos acabam tendo a habilidade de fazer contribuições relevantes no desenvolvimento destes sistemas. Assim os resultados das pesquisas tornam-se incrementalmente acadêmicos e sem qualquer relação com aspectos pragmáticos do processo de ensino-aprendizagem [MUR91].

Segundo Anderson [AND88], acima de 50% de nosso esforço é gasto para codificar o conhecimento do domínio. Além disso, muitas das regras para o desenvolvimento de uma boa base de conhecimento podem ser facilmente conferidas. Portanto, é importante investigar como a qualidade do conteúdo a ser ensinado pode ser controlada, não após, mas durante a criação do material a ser ensinado.

As metodologias para a construção de STIs são recentes e ainda estão se firmando, de modo que não há consenso sobre a melhor arquitetura para um STI. Além disso, diferentes domínios parecem requerer estruturas diferentes. Por isso tudo, são raras as propostas, na literatura, de ferramentas para a construção de STI que sirvam ao ensino de domínios quaisquer.

No entanto, dentro de um determinado domínio, é possível se determinar características comuns entre os Sistemas Tutores na literatura. Além disso, a grande maioria dos sistemas implementados até hoje possuem uma arquitetura básica comum, que é constituída de quatro módulos, os quais serão detalhados em capítulos posteriores. Os módulos são:

a) **Módulo do Especialista ou Domínio**: contém um modelo explícito do assunto que se deseja ensinar (conhecimento sobre **O QUÊ** ensinar);
b) **Modelo do Estudante**: contém as informações, a nível detalhado, sobre o quê o aluno sabe e entende do domínio (conhecimento sobre **QUEM** ensinar);
c) **Módulo Tutorial**: contém as estratégias pedagógicas (conhecimento sobre **COMO** ensinar);
d) **Módulo de Comunicação com o Usuário**: é o componente de comunicação do sistema que controla a interação entre o aluno e o sistema (conhecimento sobre **COMO** apresentar o material).

Pode-se encontrar exemplos de sistemas que possuem, além da arquitetura básica, características semelhantes dentro de um determinado domínio. Por exemplo, na tabela 1.1, é possível observar alguns exemplos de sistemas no domínio da Matemática que possuem características semelhantes.

Pode-se, também, citar alguns exemplos de ferramentas para a construção de STIs genéricos que foram implementadas, como por exemplo, IDE [RUS88b] [RUS91], PIXIE [SLE87], BITE-SIZE [BON86], KAFITS [MUR91], entre outros (Ver apêndice I). Entretanto, estas ferramentas estão mais focadas na generalidade do que na usabilidade, pois poucos pesquisadores têm uma análise completa das questões associadas à aquisição e representação de conhecimento associadas a este problema.

O problema de aquisição e representação de conhecimento já existia nos **Sistemas Especialistas**, assim como nas ferramentas para construir Sistemas Especialistas - também conhecidas como *shells* de Sistemas Especialistas. Uma *shell* permite que um desenvolvedor crie uma base de conhecimento sem envolver-se com a programação complexa do mecanismo de controle.

Sistema	Domínio	Base de Conhecimento	Modelo do Estudante	Modelo Tutorial[1]
WEST [BUR79]	Expressões Aritméticas	Representações Baseadas em Regras	Diagnóstico (Diferencial) ou Overlay	Ambiente de Aprendizagem Reativo com método *coach*[2]
BUGGY [BRO78]	Subtração	Redes Procedimentais	Perturbação ou Diagnóstico ou Overlay	Ambiente de Aprendizagem Reativo com Conselheiro
WUSOR [GOL82]	Relações Lógicas	Grafo Genérico	Overlay	Ambiente de Aprendizagem Reativo com método *coach*
QUADRATIC [O'SH82]	Equações Quadráticas	Representações Baseadas em Regras	Overlay	Ambiente de Aprendizagem Reativo com Conselheiro

Tabela 1.1: Comparação entre sistemas com características semelhantes. Apesar de alguns exemplos citados acima possuírem diferentes representações para a base de conhecimento, elas são representadas explicitamente.

Como nas *shells* de Sistemas Especialistas, a construção de STIs só será possível se a base de conhecimento for separada do mecanismo de controle, e o formalismo a ser usado para representar o conhecimento for explícito, pois a ausência de representação explícita torna difícil diagnosticar e corrigir os erros nesse conhecimento.

Por causa disso, e a partir de um modelo genérico de STIs no domínio da Matemática, denominado de ARQTEMA ([TAK93] [TAK94]), que será descrito no capítulo 3, o objetivo da dissertação é especificar uma ferramenta para a construção de STIs com características que seguem este modelo, de forma que o desenvolvedor - um professor talvez - possa construir um novo domínio para o Módulo do Domínio de um STI, sem ter, com isso,

[1] **Ambiente de Aprendizagem Reativo** consiste em fazer o aluno descobrir por si próprio o máximo possível da estrutura da situação.

[2] Ver página 16.

que reconstruir um STI. A idéia é, então, fornecer mecanismos de "troca de domínio" através de uma ferramenta.

O TOOTEMA é, então, um sistema que tem por objetivo ajudar na tarefa de criar e gerenciar a base de conhecimento do ARQTEMA, acompanhar o professor na estruturação e gerenciamento de grande quantidade de informações, fornecendo representações gráficas, facilidades de organização do material instrucional, e permitindo o sequenciamento do material a ser ensinado. Traz vantagens, como a redução de tempo e recursos necessários para desenvolver o conteúdo instrucional, visando a consistência e a qualidade das atividades de aprendizagem. Além disso, visa envolver o professor na criação de STIs.

1.3 Organização do Trabalho

Este trabalho está organizado da seguinte forma:

No Capítulo 1, são apresentadas as bases que motivaram este trabalho e nossos objetivos quanto ao desenvolvimento de ferramentas para a construção de STIs.

No Capítulo 2, são apresentadas a conceituação e as principais características de STIs.

No Capítulo 3, descreve-se o modelo de STI para o domínio da Matemática a ser utilizado no desenvolvimento deste trabalho.

No Capítulo 4, são apresentados e discutidos os requisitos, os recursos utilizados, a estrutura da ferramenta desenvolvida e as características de sua implementação.

No Capítulo 5, é apresentado um exemplo de interação do usuário com o protótipo implementado. Nessa interação são considerados dois tipos de usuários, o professor de Matemática e o programador que conheça a linguagem de programação C.

No Capítulo 6, são apresentadas as contribuições, conclusões e sugestões de trabalhos futuros.

No Apêndice I, são apresentados alguns exemplos de ferramentas para a construção de CAIs e STIs.

Finalmente, no Apêndice II, são apresentadas a hierarquia das classes de objetos que formam a estrutura do TOOTEMA e suas especificações.

Sistemas Tutores Inteligentes

Uma das áreas de pesquisa que mais tem crescido nas últimas décadas é aquela que envolve aplicações de **IA** na Educação. Os STIs são programas de auxílio ao ensino projetados de forma a incorporarem técnicas de **IA** de modo a fazê-los capazes de saber **O QUE** ensinar, **QUEM** ensinar e **COMO** devem ensinar [NWA90].

Neste capítulo são apresentados os princípios básicos de STIs, mostrando suas principais características e sua estrutura básica.

2.1 Sistemas Tutores Inteligentes

Os Sistemas Baseados em Conhecimento, ou simplesmente Sistemas de Conhecimento, são programas de computador que resolvem, ou ajudam a resolver, problemas que requerem inteligência humana.

Diferenciando-se dos sistemas tradicionais de processamento de dados, que processam grandes volumes de informação algoritmicamente, os Sistemas de Conhecimento geralmente examinam um grande número de possibilidades e, em muitos casos, podem construir uma solução dinamicamente.

Outros fatores em que diferem os sistemas de conhecimento dos convencionais são:

a) a maneira como estão organizados;
b) o modo como incorporam o conhecimento;
c) a forma de execução;
d) a impressão que causam aos usuários.

Os STIs constituem uma classe dos Sistemas de Conhecimento que tem por objetivos ensinar conceitos, mostrar resultados e exemplos e explicar seu conhecimento e raciocínio. São sistemas que têm, ainda, que ter capacidade de individualizar o ensino, o que significa ensinar de tal forma que o aluno se sinta como que privilegiado nesse sistema. Portanto, para o sucesso de um STI são necessárias pelo menos as seguintes características [NWA90]:

a) o STI deve assegurar uma clara articulação do conhecimento num domínio limitado;
b) o STI deve ter um modelo de desempenho do aluno, que é mantido dinamicamente e usado para guiar a interação com ele;
c) o projetista - ou o especialista no assunto - define o conhecimento e as regras de inferência, mas não a seqüência da aula, que deve ser derivada pelo programa;
d) o STI deve fornecer diagnóstico detalhado de erros, ao invés de mostrar simplesmente a resposta correta;
e) o aluno deve poder questionar um STI, além de responder a questões formuladas por ele.

Pela sua própria natureza, a pesquisa em STI é uma atividade interdisciplinar que inclui, além do domínio específico, as áreas de **IA**, Psicologia Cognitiva e Educação. A intersecção destas três áreas é denominada de Ciência Cognitiva.

2.2 Estrutura Básica de um STI

Para entender como os STIs diferem dos aplicativos tradicionais, é necessário compreender os componentes de um STI típico (Fig. 2.1). Além da Interface com o Usuário, que permite ao sistema comunicar-se com o usuário, um STI típico possui também um Módulo do Especialista ou Domínio, composto de fatos e regras relacionados às informações necessárias para ensinar o aluno, um Modelo do Estudante, que contém as informações

relativas ao aluno que está sendo ensinado, e um Módulo Tutorial, que usa as informações do Modelo do Estudante a fim de fornecer as estratégias necessárias para uma boa didática.

Uma vez que esses componentes possam ser separados, será muito mais fácil modificar o sistema de modo que todos os mecanismos e métodos anteriores ainda possam continuar a ser usados. Para que isso seja possível, é necessário que cada componente possua o seu próprio mecanismo de controle e sua base de conhecimento, e que a interação entre estes módulos seja feita através do fluxo de controle e de dados.

Esta estrutura típica pode variar em alguns STIs, mas descreve alguns componentes funcionais básicos que estão presentes ou são similares a outros componentes na maioria dos STIs. A Figura 2.1 mostra detalhadamente os elementos da arquitetura básica de um STI, do ponto de vista do usuário do sistema.

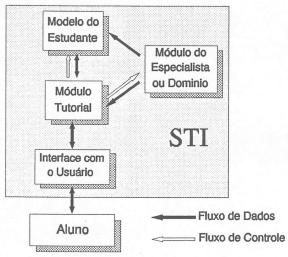


Figura 2.1: Arquitetura Básica de um STI.

2.2.1 Módulo do Especialista ou Domínio

O Módulo do Especialista ou Domínio, ou simplesmente Módulo do Domínio, compreende o conhecimento do especialista - contém fatos e regras de um determinado domínio que será ensinado ao aluno. Este conhecimento pode ser representado de várias formas, dependendo do tipo de conhecimento a ser manipulado [NIL82].

Em conhecimentos declarativos e teóricos, como por exemplo, Geometria ou Física, a representação utilizada é, geralmente, declarativa, enfatizando os aspectos estáticos do conhecimento, isto é, fatos sobre objetos, situações e suas relações e fatos gerais sobre o mundo. Para representar este tipo de conhecimento, usualmente são utilizadas redes semânticas e a metodologia utilizada para a sua aquisição é dividida em três fases: [BIE91]

1) **Determinar os objetos a serem incluídos no domínio:** dependendo do domínio, os nós que representam os objetos podem ser do tipo textual, visual (imagens gráficas, sequências de animações, etc.) ou até sons (sintetizadores de voz). Os objetos podem ser tão simples como uma única palavra ou tão complexos como um livro. É importante a variedade de objetos que os nós podem representar, pois torna o esquema de representação das redes semânticas mais versátil.
2) **Decidir como os objetos se relacionam entre si:** o próximo passo é determinar os tipos de ligações que possam existir entre os nós, isto é, não apenas decidir quais objetos se relacionam entre si, mas também a natureza exata das relações. As relações podem ser de pai para filho, entre irmãos ou qualquer outro tipo de associação. Como no caso dos objetos, a variedade de interpretações das ligações aumenta a flexibilidade da rede semântica.
3) **Verificar quais relações estão corretas:** assim que a rede tenha sido construída, percorrem-se todos os seus caminhos para verificar o que se pode capturar através das relações entre todos os seus objetos. Dependendo da complexidade e precisão do domínio, pode ser necessário o refinamento da rede ou repetir o processo diversas vezes para obter um modelo exato.

Em domínios procedimentais, como por exemplo, Linguagens de Programação, o conhecimento acerca do mundo é formulado em termos de uma seqüência de transformações, codificadas em procedimentos, que dizem como o conhecimento deve ser manipulado. O conhecimento procedimental é tipicamente explicativo: explica como fazer uma certa tarefa, como diagnosticar um problema ou recomendar uma ação. Para incorporar o conhecimento procedimental em um sistema, três passos são recomendados: [BIE91]

1) **Estabelecer as metas:** decidir sobre os possíveis objetivos e resultados do sistema é um excelente ponto de partida para o processo de transferência do conhecimento procedimental, pois a identificação destes resultados ajuda a focalizar as especificações do problema.
2) **Estabelecer os fatos:** tendo decidido as possíveis metas do sistema, o próximo passo envolve escolher os fatos ou critérios que irão servir para descriminar as possíveis recomendações.
3) **Estabelecer as relações entre as metas e os fatos:** uma vez determinados os resultados e fatores pertinentes do sistema, é necessário descrever como estes fatos e resultados se relacionam entre si.

Existem diversas técnicas que são utilizadas para representar o conhecimento de um domínio em STIs. Nos primeiros tutores, o conhecimento era representado apenas no modelo *black-box*, isto é, destinados a produzir resultados consistentes com os resultados que correspondem à realidade, porém sem tentar imitar os processos utilizados pelos seres humanos. Por exemplo, o tutor SOPHIE I [BUR79] pode responder a inúmeras questões a respeito de Eletrônica. Entretanto, por utilizar técnicas numéricas de simulação, o tutor não é capaz de explicar o porquê da resposta. Pelo fato de tais explicações serem importantes na aquisição de conhecimento, trabalhos mais recentes têm focado os modelos denominados *glass-box* - ou "modelos articulados" - que tentam representar tanto o conhecimento como os processos que parecem ser aqueles utilizados por especialistas humanos quando estes estão resolvendo algum problema, fornecendo possibilidades mais ricas nas explicações aos alunos. Como exemplos de sistemas que utilizam este modelo, pode-se citar BUGGY [BRO78] e GUIDON [CLA83].

O modelo *glass-box* parece ser o mais útil, uma vez que pode ser usado tanto no processo de avaliação como na determinação das habilidades básicas necessárias para resolver um problema. O processo de determinação das habilidades básicas é obtido observando-se a resolução de problemas executada pelo especialista e anotando as habilidades que são usadas. O processo de avaliação das habilidades básicas consiste em gerar o espaço de comportamentos alternativos das melhores resoluções, obter a resposta do aluno e comparar aquele espaço com esta resposta. Entretanto, o processo de avaliação requer mais computação e robustez para gerar o espaço de comportamentos alternativos, tornando o modelo *glass-box* ineficiente [BUR79].

Como a implementação do modelo *black-box* não possui as restrições dos algoritmos semelhantes ao raciocínio humano, ele pode ser considerado potencialmente mais eficiente e, além disso, mais útil na avaliação da resposta do aluno. Entretanto, como as habilidades

utilizadas para gerar as resoluções ótimas não são semelhantes às habilidades utilizadas pelo aluno, o modelo *black-box* não pode ser utilizado diretamente na determinação das habilidades. Isto levanta a possibilidade de combinar um eficiente e robusto modelo *black-box*, na avaliação das habilidades, com um modelo *glass-box* pouco eficiente, na determinação das habilidades.

A eficiência computacional não é a única razão do desenvolvimento da interação entre o modelo *black-box* e o modelo *glass-box*. O modelo *black-box* utilizado na avaliação necessita apenas ser estendido com as partes incompletas do modelo *glass-box*, que são necessárias para detectar características críticas ou tutoriais das respostas produzidas pelo modelo *black-box*. Por outro lado, o modelo *glass-box* não precisa ser capaz de produzir a resolução completa de si próprio, precisa apenas trabalhar *backward* a partir da solução produzida pelo modelo *black-box*, para determinar as características tutoriais importantes [BUR79] [BUR92].

2.2.2 Modelo do Estudante

A eficiência e eficácia de um STI no ensino depende muito de sua estratégia para detectar, representar e manipular o conhecimento sobre o aluno a quem ele está ensinando. Para tanto, um STI precisa de técnicas que consigam, da melhor forma, realizar esta estratégia.

Estas técnicas são o que se chama de modelagem do estudante, e o componente do STI que reúne essas técnicas e possibilita esta estratégia é denominado de Modelo do Estudante. Portanto, o Modelo do Estudante procura detectar, representar e manipular o estado atual do conhecimento do aluno.

Espera-se que, com a utilização do Modelo do Estudante, o STI possa inferir aspectos não observáveis do comportamento do aluno. Assim, uma inferência pode produzir uma interpretação das ações do aluno e também levar a uma reconstrução do conhecimento que originou estas ações; pode fornecer conselhos e explicações específicos para cada aluno, e gerar instruções dinamicamente, ao invés de apresentar instruções pré-definidas durante o processo de aprendizagem.

A própria interação com o tutor já leva o aluno a alterar seu estado cognitivo ao longo do tempo, além da interação cotidiana com o ambiente, que também o influencia [VIC92]. Isso significa que o Modelo do Estudante deve ser capaz de representar tanto os aspectos do conhecimento como do desenvolvimento do aluno.

O objetivo da pesquisa sobre modelagem do estudante é obter mais informações sobre o estado cognitivo do aluno através da interação com este, a fim de fornecer uma instrução mais individualizada, e, além disso, experimentar diferentes regras tutoriais para determinar

como e quando o sistema tutor deve utilizar e atualizar o Modelo do Estudante, bem como o que deve ser representado nesse modelo.

Modelagem do Estudante

Um Modelo do Estudante deve suportar uma descrição consistindo de informações sobre desempenho, motivação, dificuldades e facilidades que caracterizam o aluno. Atualmente existem diversos tipos de Modelos de Estudante, que são classificados segundo suas características de modelagem. Considerando como os modelos interpretam as características do aluno, estes podem ser classificados em modelo *overlay*, modelo de perturbação, modelo de diagnósticos e modelo procedimental [VAS90].

O **modelo *overlay***, sugerido por Goldstein, pode ser visto como um subconjunto da base de conhecimento do domínio (Fig. 2.2). Para tanto, a representação do conhecimento no Modelo do Estudante e no Módulo do Domínio têm que ser compatíveis.

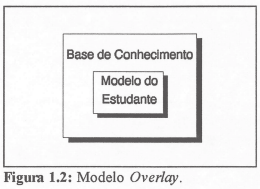


Figura 1.2: Modelo *Overlay*.

Isto permite que o estado de conhecimento do aluno seja comparado com o Módulo do Domínio e a instrução então será influenciada pelas porções do modelo que se mostram "fracas". O aluno é dito ser fraco em uma habilidade ou conceito quando este se desvia das soluções propostas pelo especialista.

Contudo, o comportamento incorreto ou subótimo nem sempre resulta do conhecimento incompleto. Pode também ser decorrente de concepções incorretas na mente do aluno. Portanto, um Modelo do Estudante também deverá fornecer representações explícitas das concepções incorretas do aluno para futuras remediações.

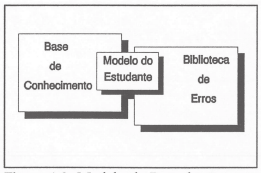


Figura 1.3: Modelo de Perturbação.

O **modelo de perturbação** - ou **modelo de *bugs*** - não apenas detecta quando um erro ocorre, mas analisa os conceitos errados e identifica os fatores que levaram à resposta errada. Além de ser um subconjunto da base de conhecimento, o

Modelo do Estudante deve consultar uma biblioteca de erros, para descrever possíveis erros cometidos pelo aluno, suas conseqüências e as contradições que o levaram a realizá-lo (Fig. 2.3). Entretanto, estes modelos ainda são limitados principalmente pela estrutura da biblioteca de erros [VIC92]. O sistema BUGGY [BRO78] e PROUST [JOH87] são exemplos de sistemas que utilizam esta abordagem.

O **modelo de diagnósticos**, também conhecido como **modelo diferencial**, compara o trabalho realizado pelo aluno com o desempenho de um especialista [BUR79]. A "diferença" encontrada na comparação deve fornecer hipóteses sobre o que o aluno não conhece ou não domina. A figura 2.4 ilustra esse modelo.

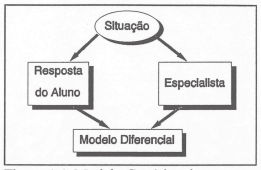


Figura 1.4: Modelo Genérico do processo de modelagem diferencial [WAC88].

Os sistemas WEST [BUR79], SOPHIE III, GUIDON, MACSYMA-ADVISOR [GEN82] são exemplos de sistemas que utilizam este modelo. Todos estes sistemas utilizam um Sistema Especialista para fazer suas diagnoses [WIL88]. Por exemplo, SOPHIE III utiliza um Sistema Especialista no domínio de Circuitos Eletrônicos como auxílio para isolar erros hipotéticos no comportamento de alunos, e GUIDON foi construído sobre o Sistema Especialista MYCIN para diagnóstico médico.

O **modelo procedimental** contém os caminhos supostamente utilizados pelo aluno na resolução de um determinado problema sugerido pelo sistema, isto é, contém os resultados intermediários de uma ação de um especialista no domínio, executando-a de modo similar ao do aluno. Para que seja possível o sistema avaliar o aluno, são incorporadas ao Módulo do Domínio algumas técnicas especiais para possibilitá-lo imitar o modo do aluno solucionar seus problemas [VAS90].

Estes modelos quase sempre utilizam o conhecimento pré-estabelecido do domínio e de erros previamente observados e interpretados por vários alunos. O domínio é utilizado como um parâmetro de comparação.

O que distingue os modelos *overlay* dos modelos procedimentais é o fato de os modelos *overlay* poderem usar comparações com formas genéricas de representação do conhecimento, enquanto os modelos procedimentais utilizam e comparam o mecanismo de resolução do problema com formas de representações mais apuradas. Deste modo, os modelos procedimentais têm melhores capacidades de diagnosticar do que os modelos *overlay*. Modelos procedimentais podem também ser vistos como modelos de perturbação se eles utilizarem

algoritmos, divididos em pequenas porções distintas, correspondendo a partes específicas do conhecimento, que podem ser executadas correta ou incorretamente pelo aluno. Como exemplo de sistema que utiliza o modelo procedimental, pode-se citar o DEBBUGY [VAS90] - seu Modelo de Estudante também pode ser classificado como modelo de perturbação.

2.2.3 Módulo Tutorial

O Módulo Tutorial é o componente do STI que planeja e governa a interação com o aluno. Contém estratégias de ensino que são selecionadas e combinadas dinamicamente de acordo com as reações dos alunos. As estratégias constituem conhecimentos sobre como ensinar, ou seja, sobre como gerar, a partir das informações diagnosticadas, monitoradas e analisadas, uma seqüência tática de ensino capaz de apresentar um determinado tópico a um determinado aluno. Esta seqüência pode ser formada pelos seguintes tipos de táticas: descrição de um conceito ou estrutura conceitual; explanação de uma estrutura causal; definição de objetos; concretização de conhecimento abstrato; apresentação de questões e/ou problemas; comentários sobre as ações do aluno, entre outros.

Considerando que um STI visa melhorar o desempenho de alunos no processo de aprendizagem, é neste módulo que as diferentes formas de interação com o aluno são definidas. Visa atender aspectos tais como o nível de conhecimento e o desempenho do aluno (fornecidos pelo Modelo do Estudante), para diferenciá-lo de outros usuários do sistema, permitindo uma interação mais detalhada ao usuário menos experiente e, ao usuário mais experiente, saltar etapas.

Segundo Kearsley [KEA87], é importante que um STI adapte suas estratégias de ensino para individualizar o processo de aprendizagem. Entretanto, esta adaptação só será possível se o conhecimento contido no Módulo Tutorial for representado explicitamente, permitindo que o desenvolvedor modifique e melhore as estratégias de ensino. A representação explícita do conhecimento também possibilita que algumas estratégias sejam reutilizadas em outros domínios.

No Módulo Tutorial, o controle da iniciativa, isto é, a maneira como se dará o gerenciamento da iniciativa no controle do aprendizado, pode variar entre:

a) **controle total do sistema** sobre o processo de aprendizagem. O sistema monitora todas as atividades realizadas pelo aluno, o qual apenas responde às questões formuladas pelo sistema;

b) **controle total do aluno** sobre o processo de aprendizagem. O aluno tem a possibilidade de controlar totalmente o sistema. Este tipo de controle pode ser encontrado, por exemplo, em sistemas de treinamento (*Coach Systems*), e

c) **controle de iniciativa mista**. O sistema e o aluno revezam-se no controle do processo de aprendizagem, isto é, ambos podem perguntar e responder questões.

Para muitos autores ([BUR79] [BRE88] [BRO82] [CLA83] [GOL82] [KIM82] [MIL82] [RIC89] [VIC92] [WEN87]), o sucesso das estratégias de ensino depende da definição das seguintes questões:

a) Quando interromper o aluno? Que razões justificam interromper o curso de raciocínio ou aprendizagem do aluno?

b) O que dizer? Esta questão desdobra-se em:
   1) seleção do(s) tópico(s) a ser apresentado(s);
   2) ordenação dos tópicos, se houver mais de um.

c) Como dizer? Esta é provavelmente a questão mais difícil. Não há soluções gerais concretas por falta de teorias pedagógicas suficientemente detalhadas.

Nos atuais STIs, as estratégias de ensino para especificar a apresentação do material instrucional são basicamente representadas por dois métodos: o método socrático e o método *coach* [HEG90] [PAR87].

O **método socrático** conduz o aluno ao longo de uma linha de raciocínio que deverá conseqüentemente levar a uma solução correta - ou representação correta. O sistema WHY [STE82], por exemplo, questiona sobre locais onde ocorrem fortes tempestades, diagnostica passos errados no conhecimento do aluno e informa a este aluno sobre os passos corretos. Similarmente, no sistema SCHOLAR é utilizado o estilo socrático de diálogo tutorial. O sistema tenta primeiro diagnosticar as *misconceptions* do aluno e, então, apresenta alguns conceitos que irão forçar o aluno a ver seu próprio erro.

O **método *coach*** fornece ao aluno um ambiente no qual ele desempenha uma atividade, tal como jogos de computador, a fim de aprender habilidades relacionadas e habilidades de resolução de problemas genéricos. O objetivo do programa é dar ao aluno lazer e propiciar aprendizado como uma conseqüência da brincadeira. Dois bons exemplos de sistemas *coach* são o WEST [BUR79] e o WUSOR [GOL82]. Em ambos os sistemas, um Modelo do Estudante é formado pela comparação do comportamento do aluno com a de um especialista. WUSOR utiliza um especialista articulado e os conselhos derivam do "rastreio"

do especialista. No WEST, os seus conselhos vêm de conhecimentos locais, associados com fraquezas particulares.

Muitos tutores humanos interrompem demasiadamente seus alunos, por falta de tempo ou paciência, e isso priva o desenvolvimento de importantes habilidades cognitivas que os permitem detectar e usar seus próprios erros. Portanto, neste tipo de ambiente, é melhor que o aluno descubra por si só o máximo possível da estrutura da situação. Quando o aluno realiza um movimento errado, ele tem a chance de descobrir o jogo por si próprio, com o sistema apenas fornecendo conselhos que o ajudam a superar a fraqueza observada no Modelo do Estudante.

2.2.4 Módulo de Comunicação com o Usuário

Quanto mais o aluno se sentir inserido no seu ambiente de aprendizagem, mais rica será a sua experiência, e esta inserção pode ser facilitada através da interação com uma interface de múltiplos meios. Entretanto, mesmo quando a interface do usuário propicia um alto grau de fidelidade em modelar algum sistema, este será de pouco uso se os usuários tiverem dificuldades de se comunicar com ele ou de compreendê-lo. Portanto, a escolha do meio de comunicação mais adequado ao tópico a ser ensinado e a eficácia da comunicação são problemas relevantes.

Grande parte do potencial, da flexibilidade e da utilidade característicos de um STI, provém dos recursos do Módulo de Comunicação com o Usuário. É através destes recursos que o sistema tutor pode exercer duas de suas principais funções [VIC92]:

a) a apresentação do material instrucional e

b) a monitoração do progresso do aluno.

Além do aluno, outro usuário que não se deve esquecer no projeto de um STI, é o professor. Como se sabe, a interação entre o professor e o STI é diferente da interação entre o aluno e o STI. Neste caso, o sistema não tem o dever de ensinar, mas de expor e explicar o funcionamento e a estrutura lógica do sistema, bem como fornecer informações sobre o desempenho dos alunos que interagem com o sistema. Assim sendo, esta interação teria com funções [TAK94]:

a) permitir que o professor verifique e faça alterações, se necessário, no sistema, adequando-o às suas necessidades de ensino. Para isto, seria necessário que o

sistema se "mostrasse" ao professor, apontando os pontos passíveis de alteração e a forma (explicação) de como ele pode ser alterado, e

b) servir como fonte para avaliações dos alunos pelo professor. A partir das informações do Modelo do Estudante (apresentado através de gráficos, tabelas, textos, entre outros), o professor teria meios de avaliar tanto o aluno como o sistema.

Diferentes abordagens têm sido propostas para melhorar a comunicação entre STI e seu usuário. Entre elas pode-se citar o processamento de linguagem natural. Entretanto, tem se evitado este tipo de abordagem, pelo fato da compreensão e geração de linguagem natural serem, até hoje, um dos grandes desafios da **IA**.

Mas pode-se citar exemplos de STIs que utilizam alguns mecanismos de linguagem natural, por exemplo: a comunicação do SCHOLAR é limitada a sentenças simples e compreensão de respostas corretas; GUIDON compreende sentenças simples e uma lista de comandos e SOPHIE utiliza mecanismos de análise semântica. Enquanto, outras abordagens vêm utilizando cada vez mais os recursos oferecidos pelos novos avanços tecnológicos, como novos recursos gráficos, animações, sons, entre outros.

Os sistemas **hipertexto/hipermídia** propiciam um meio que tem enriquecido a apresentação do material instrucional, já que podem oferecer uma variedade de recursos - texto, material gráfico, recursos de vídeo, animação, som, etc - aliada à possibilidade de percorrer o material de maneira vinculada à semântica do conteúdo ("navegando sobre o domínio").

Dado que a interface deve introduzir o aluno no domínio, esta deveria fornecer variações de acordo com as habilidades de diferentes alunos. Através do caminho percorrido na base de conhecimento, os sistemas hipertexto/hipermídia podem monitorar o progresso do aluno, ou seja, obter informações sobre o desenvolvimento do aluno, e assim podem habilitar ou desabilitar certos caminhos na base de conhecimento, adequando as informações da base ao nível e necessidade do aluno [NUN93].

Embora apenas superficialmente, descreve-se abaixo um conjunto parcial de recursos desejáveis para um STI e que servem de suporte à comunicação usuário/sistema:

a) lidar com diversos estilos de interação, talvez com a capacidade de passar de um estilo para outro, de acordo com a escolha do usuário permitindo-lhe uma experiência mais rica;

b) apresentar dados em diversos formatos e meios;

c) ser fácil de usar, o que significa que ele deve minimizar o número de ações necessárias para uma comunicação com o tutor;

d) ser altamente interativo e fornecer muitas informações sobre os estados intermediários da resolução do problema do aluno;

e) propiciar respostas rápidas, ou pelo menos permanecer dentro de limites aceitáveis de tempo de resposta.

2.3 Conclusões Parciais

Nos STIs a representação do conhecimento possui uma grande influência na capacidade de explicação dos tutores e na facilidade de alteração, bem como na competência de diagnosticar os seus alunos. Assim sendo, é desejável que o conhecimento do domínio de ensino seja representado explicitamente.

O conhecimento pedagógico é uma questão muito importante a ser considerada para que o STI não prive o desenvolvimento do aluno. Muitos STIs têm representado seu conhecimento pedagógico através de regras de produção, pelo fato de facilitar a alteração do conhecimento.

O Modelo do Estudante é necessário para determinar as causas dos erros cometidos pelo aluno e para possibilitar a individualização do ensino. Assim sendo, foi mostrada uma classificação dos Modelos do Estudante existentes, focalizando as capacidades de diagnóstico desses modelos.

O Módulo de Comunicação com o usuário conseguiu um avanço significativo a partir do surgimento dos sistemas hipertexto/hipermídia. Conforme a afirmação de Rickel [RIC89], as pessoas retêm 25% do que ouvem, 45% do que vêem e ouvem, e 70% do que vêem, ouvem e fazem, portanto há ter fortes argumentos a favor de interfaces que incluam não apenas texto, mas também imagem e som, permitindo que o usuário tenha opções de interação com o STI.

Neste capítulo foi apresentada uma introdução a STIs, concentrando-se na definição da arquitetura básica de um STI típico, na qual a ferramenta a ser proposta neste trabalho estará apoiada.

3

ARQTEMA: Uma Arquitetura Genérica para STIs em Matemática

Em seu artigo "*Structuring Mathematical Knowledge*", E. Michener [MIC78] propõe um modelo de representação do conhecimento matemático através de redes envolvendo três categorias do conhecimento: conceitos, resultados e exemplos. Este modelo é a base do Módulo do Domínio do ARQTEMA, uma arquitetura genérica para STIs desenvolvida por Takehara [TAK94].

Neste capítulo é apresentado o modelo de Michener e a arquitetura ARQTEMA, sobre a qual foi desenvolvida a *shell* de STIs denominada TOOTEMA.

3.1 O Modelo de Michener

Examinando como os matemáticos usam e explicam seu conhecimento, Michener concluiu que existem, no mínimo, três categorias de informações necessárias para representar o conhecimento matemático: *conceitos*, *resultados* e *exemplos*.

A categoria de *conceitos* representa definições e princípios matemáticos. A categoria de *resultados* contém os aspectos lógicos tradicionais da matemática, isto é, os teoremas e suas provas, e a categoria de *exemplos* contém o material ilustrativo.

Além da classificação em categorias, Michener estabeleceu classes epistemológicas, baseada no fato de que os itens da teoria desempenham funções distintas e que, portanto, existe uma divisão taxonômica para esses itens com relação a seu papel na nossa compreensão e às atribuições especiais que desempenham de forma única.

3.1.1 A Categoria de Conceitos

Os conceitos podem ser estruturados por uma relação de *ordem pedagógica*, onde se estabelece que um conceito A deve ser introduzido (apresentado) antes que um conceito B. Algumas vezes esta relação reflete o fato de que o conceito A faz parte da definição do conceito B; outras vezes, reflete preferências na ordem de exposição dos conceitos (opção didática). Em Álgebra Linear, por exemplo, alguns autores definem um autovalor de uma matriz A em termos da equação $Av = v$, enquanto outros o definem como raiz da equação característica $\det(A - I) = 0$. Logo, duas abordagens diferentes, respectivamente, geométrica e algébrica. Para a primeira definição, o conceito de autovalor é mais intuitivo, sendo necessário um pré-conhecimento sobre autovetores. Já na segunda abordagem, não necessariamente é preciso saber o que é um autovetor, mas é preciso conhecimento sobre determinantes e polinômios.

A categoria de conceitos possui três classes epistemológicas: *definições*, *mega-princípios* e *contra-princípios*.

A classe de definições fornece as definições contidas na teoria tratada.

Os mega-princípios fornecem núcleos de conhecimento na forma de fortes sugestões ou heurísticas. Por exemplo, "matrizes simétricas são boas" é um mega-princípio da Álgebra Linear. Este mega-princípio é, de certa forma, um resumo de muitos resultados que mostram que matrizes simétricas são diagonalizáveis e bem-condicionadas.

Contra-princípios são avisos que indicam possíveis fontes de erros ou

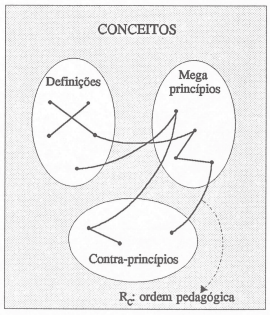


Figura 3.1: A categoria de conceitos.

problemas. Por exemplo, "não dividir por zero" é um Contra-Princípio bem familiar. A Figura 3.1 ilustra a categoria de conceitos.

3.1.2 A Categoria de Resultados

Os resultados podem ser estruturados pela relação de *dedução lógica*, na qual "A precede B" significa que o resultado A é usado para provar o resultado B.

A categoria de resultados possui duas classes epistemológicas, denominadas *resultados básicos* e *resultados culminantes*.

Resultados básicos estabelecem propriedades elementares, mas importantes, de conceitos, enquanto que resultados culminantes são aqueles para os quais a teoria se direciona, ou seja, são os pontos principais da teoria. Um resultado básico da Álgebra Linear seria, por exemplo: λ é um autovalor da matriz A se, e somente se, $\det(A - \lambda I) = 0$. Um resultado culminante no estudo de polinômios seria, por exemplo, o Teorema Fundamental da Álgebra.

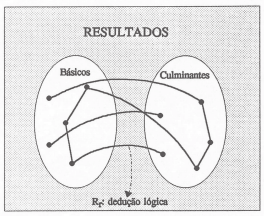


Figura 3.2: A categoria de resultados.

A Figura 3.2 ilustra a estrutura da categoria de resultados tendo a dedução lógica como relação entre seus itens.

3.1.3 A Categoria de Exemplos

Os exemplos podem ser organizados pela relação de *derivação fabricacional*, na qual "A precede B" significa que o exemplo A é usado para construir o exemplo B. Por exemplo, a partir dos números naturais, pode-se gerar os inteiros Z (pela inclusão de números negativos), dos quais pode-se gerar os racionais Q (pela formação dos quocientes), dos quais pode-se construir os números reais R (completando com a Seqüência de Cauchy).

A categoria de exemplos possui quatro classes epistemológicas, denominadas *exemplos iniciais*, *exemplos de referência*, *exemplos modelos* e *contra-exemplos*.

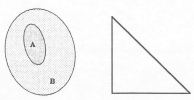


Figura 3.3: Modelo para inclusão de conjuntos e triângulo retângulo.

Exemplos iniciais são aqueles que auxiliam a introduzir um novo assunto, motivando suas definições e resultados básicos.

Exemplos modelos são situações-padrão que sugerem a essência de um resultado ou conceito. Eles indicam situações genéricas que, geralmente, devem ser adequadas às especificidades do problema em questão. Exemplos modelos para a idéia de inclusão de conjuntos e triângulo retângulo são mostrados na Figura 3.3.

Exemplos de referência fornecem um nó comum através do qual vários resultados e conceitos são ligados. Portanto, são exemplos referidos várias vezes durante o desenvolvimento de uma teoria. Por exemplo, na teoria dos números, os inteiros são sempre referenciados para facilitar o entendimento.

Contra-exemplos são muito conhecidos e mostram que uma asserção não é verdadeira. Assim como os contra-princípios, os contra-exemplos nos advertem sobre os limites das propriedades de uma teoria. A Figura 3.4 ilustra a categoria de exemplos.

Vistas em detalhes as três categorias que representam uma teoria matemática, pode-se fazer algumas considerações: exemplos modelos, mega-princípios e resultados culminantes representam os itens mais importantes de uma teoria; contra-exemplos e contra-princípios mostram os limites de uma teoria; resultados básicos e exemplos iniciais são pontos de entrada para

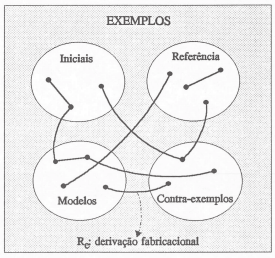


Figura 3.4: A categoria de exemplos.

introduzir uma teoria. Em adição a estas analogias, pode-se considerar relações específicas entre itens em diferentes categorias, como é mostrado a seguir.

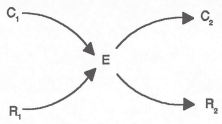
3.1.4 O relacionamento entre as categorias

No modelo proposto por Michener, a representação de um item também considera os itens fora da categoria a qual ele pertence. A "idéia dual" realça as relações entre os três espaços de representações. Estes são considerados "espaços epistemológicos duais" de cada outro. Assim, cada uma das três categorias (conceitos, exemplos, resultados) possui itens duais nas outras duas categorias.

Por exemplo, seja C um item da categoria de conceitos, R um item da categoria de resultados e E um item da categoria de exemplos:

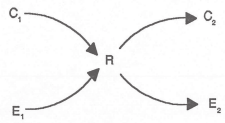
Um exemplo (E) possui como itens duais:

1) os conceitos ($\mathbf{C_1}$) e resultados ($\mathbf{R_1}$) necessários para sua discussão ou fabricação, e
2) os conceitos ($\mathbf{C_2}$) e resultados ($\mathbf{R_2}$) motivados por ele.



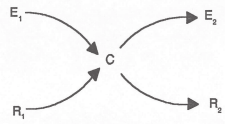
Um resultado (R) possui como itens duais:

1) os exemplos ($\mathbf{E_1}$) que o motivam e os conceitos ($\mathbf{C_1}$) necessários para ele e para sua prova, e
2) os conceitos ($\mathbf{C_2}$) e exemplos ($\mathbf{E_2}$) derivados dele.



Um conceito (C) possui como itens duais:

1) os exemplos ($\mathbf{E_1}$) que o motivam e os resultados ($\mathbf{R_1}$) que preparam sua base, e
2) os exemplos ($\mathbf{E_2}$) que o ilustram e os resultados ($\mathbf{R_2}$) que provam coisas sobre ele.



A Figura 3.5 ilustra, através das relações duais, as ligações entre as três categorias de conhecimento. Cada relação dual pode ser de um tipo (ilustra, prepara, deriva, constrói, motiva, entre outras).

Por simplificação, denota-se o dual de um item I por D(I). Em particular, os exemplos duais de I (onde I é um resultado ou conceito) por E(I); os conceitos duais de I (onde I é um resultado ou exemplo) por C(I); e os resultados duais de I (onde I é um conceito ou exemplo) por R(I). Portanto, o dual de um exemplo E é D(E) = R(E) ∪ C(E). Analogamente, D(C) = E(C) ∪ R(C), onde C é um conceito, e D(R) = E(R) ∪ C(R), onde R é um resultado.

Por outro lado, dois itens *relacionam-se* via a idéia dual (~) se eles compartilham itens duais:

$$A \sim B \Leftrightarrow D(A) \cap D(B) \neq \varnothing$$

Mais especificamente,

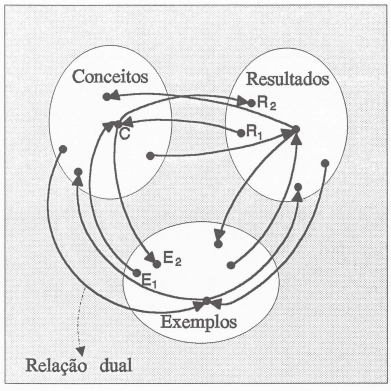


Figura 3.5: O relacionamento entre as três categorias.

1) são conceitualmente duais (C~) se compartilham conceitos:

$$A\ C\sim B \Leftrightarrow C(A) \cap C(B) \neq \varnothing$$

2) são ilustrativamente duais (E~) se compartilham exemplos:

$$A\ E\sim B \Leftrightarrow E(A) \cap E(B) \neq \varnothing$$

3) são dedutivamente duais (R~) se compartilham resultados:

$$A\ R\sim B \Leftrightarrow R(A) \cap R(B) \neq \varnothing$$

Dois itens são *equivalentes duais módulo um item* se, e somente se, este item é comum no conjunto dual de ambos, isto é:

$$A \equiv \text{dual } B \bmod C \Leftrightarrow C \ D(A) \cap D(B)$$

Esta relação é muito útil pelo fato de mostrar de que maneiras certas informações, que aparentemente não se relacionam, se associam.

Uma relação muito forte é a de *equivalência dual*, definindo que dois itens são equivalentes duais se, e somente se, os seus itens duais forem iguais, isto é:

$$A \equiv \text{dual } B \Leftrightarrow D(A) = D(B)$$

Neste caso, o item A pode ser substituído por B em qualquer situação e vice-versa.

Dadas as características do modelo de Michener, nota-se a possibilidade de se representar o Módulo do Domínio de um STI (na área matemática) através dele. O ARQTEMA constitui uma arquitetura que faz uso deste modelo.

3.2 ARQTEMA: Arquitetura para STIs

ARQTEMA é uma arquitetura de STI desenvolvida para o ensino de domínios na área Matemática [TAK94]. A arquitetura geral é composta pelos quatro módulos pertencentes à arquitetura básica de um STI: Módulo do Domínio, Modelo do Estudante, Módulo Tutorial e Módulo de Comunicação, discutidos no capítulo dois.

3.2.1. O Módulo do Domínio

O Módulo do Domínio tem a forma de uma rede de conhecimento complexa que é baseada no modelo de Michener acrescido de mais duas categorias e de um outro nível. O conhecimento é representado de maneira explícita através de uma Rede de Registros. O Módulo do Domínio é uma estrutura dividida em dois níveis: *Nível Tópicos* e *Nível Básico* (Fig. 3.6).

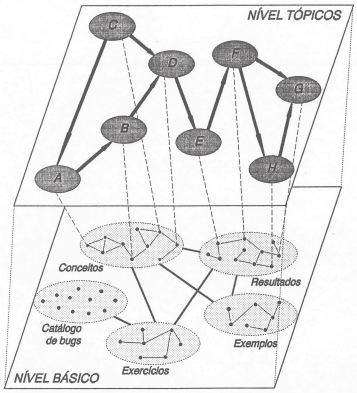


Figura 3.6: Representação Gráfica do Módulo do Domínio do ARQTEMA.

O Nível Tópicos é uma rede formada por tópicos da teoria tratada. A cada tópico está relacionado um conjunto de itens do domínio, estabelecendo uma sub-rede no domínio. Esta sub-rede determina a meta do tutor, isto é, ao ensinar um tópico, o tutor deverá percorrer todos os nós da sub-rede para concluir sua aula.

As relações entre os tópicos é uma relação de precedência, onde os tópicos seguem uma ordem pedagógica tal qual a de pré-requisitos.

Esse meta-nível possibilita que se mude as relações de precedências sem, contudo, alterar o domínio. Isto é, o conhecimento explícito da teoria é único, mas as relações entre os tópicos pode ser mudada assim como o conjunto de itens do domínio associado a cada um deles. Desse modo, pode-se pensar na utilização do sistema de modo a possibilitar o estudo por um professor, sobre suas idéias e a aplicação de seu conhecimento pedagógico.

O Nível Básico possui o conhecimento dividido em cinco categorias: as três propostas por Michener (conceitos, resultados e exemplos), mais a categoria de *exercícios* e o *catálogo de bugs*.

As categorias de conceitos, resultados e exemplos possuem uma estrutura de dados similar. Basicamente, a estrutura de dados de um conceito contém:

```
IDENTIFICADOR: número que identifica o conceito.
NOME: nome dado ao conceito.
CLASSE: classe a que pertence o conceito (definiçao, mega-principio ou
  contra-principio).
CLASSIFICAÇAO DE SUA IMPORTANCIA NA TEORIA: variando entre 1 e 4,
  indica a importância do item na teoria, respectivamente, de menor
  para maior importância.
TEXTO: definiçao do conceito.
PONTEIROS PARA SEUS SUCESSORES E PREDECESSORES: indicam a posiçao
  pedagogica do conceito dentro de sua categoria.
PONTEIROS DUAIS: indicam seus itens duais.
PONTEIROS PARA EXERCICIOS: indicam quais exercicios estao relacionados
  a este conceito dentro da classificaçao: básico, médio, avançado.
```

As estruturas de dados tanto de um resultado quanto de um exemplo não diferem muito da estrutura acima, observando-se as particularidades próprias. Por exemplo, a classe epistemológica de um resultado não será a mesma que a de um conceito ou um exemplo. Aliás, este item - exemplo - não tem o último campo, pois não se relaciona com a categoria de Exercícios.

Um conceito ou resultado tem ponteiros para os exercícios divididos em três tipos: básico, médio e avançado. Isto indica que para um determinado conceito, os exercícios podem ser classificados como de fácil, média e difícil resolução, respectivamente. No entanto, um exercício considerado difícil para um conceito pode ser considerado médio ou fácil para outro.

Com relação às categorias exercícios e catálogo de *bugs*, como elas foram introduzidas para fins tutoriais, não possuem as mesmas características das categorias do modelo de Michener. Por exemplo, não se aplica a relação dual aos itens destas duas categorias e nem tampouco a relação entre seus itens segue uma ordem pedagógica, de dedução lógica ou de derivação fabricacional.

Na categoria de exercícios encontram-se os problemas e exercícios relativos à teoria tratada. Os elementos desta categoria estão estruturados em classes que compõem tipos de resolução similares. Por exemplo, exercícios que propõem a resolução de equações polinomiais podem ser classificados numa mesma classe. Mas, cada qual tem uma única identificação.

A relação desta categoria com a de conceitos e resultados é uma relação de *envolvimento na resolução*. Ou seja, um exercício relaciona-se com os conceitos e/ou resultados que ele utiliza em sua resolução.

Por ser genérico, o Módulo do Domínio do ARQTEMA não permite a resolução de problemas baseado em seu conhecimento. A resolução de exercícios é feita, por enquanto, através de múltipla escolha. Esta forma simplifica o trabalho de detecção de *bugs*, pois associado a cada escolha incorreta há um *bug* pré-definido.

O catálogo de *bugs* contém informações sobre os possíveis *bugs* da teoria tratada. Esses *bugs* são específicos a cada teoria, precisando-se, então, averiguá-la para determinar quais e como são os *bugs* associados a ela.

No ARQTEMA os *bugs* relacionam-se com os exercícios que os detectam e com outros *bugs* que podem ser, também, causadores dos erros cometidos. Isto é, existe um *bug* mais provável relacionado à resposta incorreta, mas pode acontecer de ele não ser exatamente o causador do erro e, sim, um *bug* co-relacionado. Existe uma relação que identifica isto e permite ao tutor atacar o problema de diversas formas com o objetivo de sanar o problema. Existe também a informação de quais são os possíveis itens conceituais envolvidos no *bug*.

3.2.2 O Modelo do Estudante

O Modelo do Estudante no ARQTEMA é formado por um conjunto de parâmetros que caracterizam quantitativamente e qualitativamente o aluno. Os parâmetros que compõem o Modelo do Estudante na arquitetura ARQTEMA incluem:

1) Nome do Aluno (Identificação):

O nome do aluno é um dado que o sistema usa para armazenamento e recuperação do Modelo do Estudante em arquivo. Além do que, o tutor usa essa informação para se comunicar de forma personalizada com o aluno.

2) Número da Sessão, Tópicos Abordados por Sessão, Lista de Exemplos e Informações sobre Exercícios por Tópico Abordado:

O Número da Sessão, n, representa a n-ésima vez em que o aluno utilizou o sistema. Relacionados a cada sessão, têm-se os Tópicos Abordados (tópicos que foram apresentados ao aluno) durante seu período. Por exemplo, João em sua terceira sessão viu os tópicos Tópico-6, Tópico-7 e Tópico-8.

A cada tópico abordado está vinculada uma lista dos exemplos que o aluno viu e um campo com informações sobre os exercícios que o aluno fez (vazio, se o aluno não quis ser avaliado). Neste campo, tem-se uma lista dos exercícios que o aluno fez e o nível final atribuído ao aluno por seu desempenho nos exercícios.

O Número da Sessão é atualizado a cada nova sessão, mas Tópicos Abordados por Sessão, Lista de Exemplos e Informações sobre Exercícios por Tópico Abordado são atualizados ao final de cada tópico.

3) Conceitos Supostamente Vistos:

Neste capítulo, conceito significa a idéia de informação da teoria como um todo, referindo-se tanto aos conceitos quanto aos resultados de Michener.

Quando um aluno entra no sistema pela primeira vez, ele escolhe o tópico inicial de sua primeira sessão. Se o tópico escolhido não for o primeiro, significa, para o tutor, que o aluno acredita conhecer os tópicos anteriores e, conseqüentemente, os conceitos a ele relacionados. Conceitos Supostamente Vistos são, então, aqueles conceitos que o tutor supõe que o aluno sabe devido ao fato de ele (aluno) não ter iniciado seus estudos por tais conceitos.

4) Conceitos Vistos Anteriormente/Número da Sessão/Tópico:

Conceitos Vistos Anteriormente são aqueles conceitos que foram apresentados ao aluno em sessões anteriores. Sabem-se quais tópicos foram vistos em determinada sessão. Porém, também é importante saber exatamente quais e como foram dados os conceitos, visto que o percurso pela rede de conceitos e resultados pode diferir de um aluno para outro. O número da sessão permite a visão de como o aluno percorre por si próprio a teoria. Isto é, supondo que o aluno queira ver um determinado conceito, não diretamente relacionado ao tópico que ele está vendo, então tem-se no Modelo do Estudante esta informação.

5) Conceitos Vistos/Número da Sessão/Tópico:

Conceitos Vistos são aqueles conceitos atualmente em estudo pelo aluno, isto é, todos os conceitos apresentados ao aluno na sessão que está se desenvolvendo. A partir do momento que o aluno deixa o sistema, os conceitos vistos passam a ser conceitos vistos anteriormente, ficando este campo vazio no modelo do aluno. Este tipo de diferenciação deve-se ao fato de que um conceito que já foi tratado em outra sessão pode ser, por exemplo, esquecido pelo

aluno. Já, se o conceito está sendo tratado atualmente pelo sistema, a possibilidade de esquecimento é, de certa forma, remota, sendo mais provável, por exemplo, uma confusão por parte do aluno. Sabendo se o conceito foi dado em outras sessões ou na atual, o tutor pode seguir caminhos diferentes no tratamento de um erro, por exemplo. A cada novo conceito dado, o sistema atualiza esta informação acrescentando-o à lista de conceitos dados.

6) Exemplos Vistos:

Exemplos Vistos são os exemplos que são dados ao aluno enquanto ele está vendo um determinado tópico. Existe a possibilidade de a cada conceito ou resultado, o aluno ter acesso aos exemplos, sendo que isto precisa ser armazenado para que o sistema saiba quais os exemplos que já foram mostrados com relação ao tópico em andamento. Ao final de cada tópico, estas informações são atualizadas no parâmetro (2), sendo colocadas na Lista de Exemplos, e este parâmetro passa a ser nulo novamente.

7) Exercícios/Número da Sessão/Tópico:

Os exercícios são a forma pela qual o tutor pode tentar avaliar o aluno. Neste campo do Modelo do Estudante serão armazenados todos os exercícios dados ao aluno. Através da identificação, tem-se uma lista de exercícios aos quais estarão relacionadas informações sobre qual a sessão e qual o tópico em que foram aplicados, número do exercício aplicado, se acertou ou não, se houve ou não revisão e, em caso positivo, qual exercício resolvido foi mostrado.

O campo de exercícios será atualizado a cada vez que o aluno fizer uma sessão de exercícios.

8) *Bugs* Detectados/Número da Sessão/Tópico:

Bugs detectados são aqueles que são determinados através dos exercícios que o aluno erra. Este parâmetro é uma lista de *bugs* identificados contendo informações sobre em que sessão e tópico foram identificados e quais os exercícios que os revelaram.

Os *bugs* identificados são utilizados para que o tutor verifique se o aluno já cometeu o mesmo erro anteriormente. Isto poderia significar que o aluno está com deficiência em algum conceito anterior que não foi suprida totalmente. Então ele ataca a deficiência relacionada ao conceito anterior antes de fazê-lo com o atual. Este parâmetro é atualizado a cada *bug* identificado.

3.2.3 O Módulo Tutorial

O Módulo Tutorial no ARQTEMA é dado por um sistema de regras de produção estruturadas [NIL82][RIC83]. De acordo com o contexto, regras são acionadas para dar continuidade a sessão.

Existe uma fase inicial do sistema tutor onde o aluno, após escolher o domínio de ensino que quer estudar, entra com seu nome para que o sistema possa descobrir se ele já utilizou alguma outra vez este mesmo tutor (sistema/domínio escolhido). Quando o aluno digita seu nome, uma função de busca de Modelos de Estudante é acionada.

Se o Modelo do Estudante não é encontrado (aluno é novato), o tutor pede ao novo aluno que escolha um tópico inicial para sua primeira aula — seqüência de tópicos dada pelo Nível Tópicos. Assim, após escolher o tópico inicial de sua primeira sessão, o sistema cria e atualiza o Modelo do Estudante com a informação de que os conceitos pertinentes aos tópicos anteriores ao escolhido são supostamente vistos e inicia o sistema de regras.

Caso o aluno não seja novato, o tutor irá mostrar a ele seu histórico. Através das informações contidas no Modelo do Estudante, o tutor informa quantas sessões o aluno já teve, quais tópicos foram dados em cada sessão e o desempenho obtido por ele em cada tópico. Assim, o aluno pode escolher por continuar - o que significa seguir a ordem dada no Nível Tópicos - ou iniciar sua sessão por outro tópico qualquer.

Se o aluno decide continuar, o próximo tópico (segundo a ordem do Nível Tópicos e a informação do último tópico visto por ele) é o tópico inicial da nova sessão - iniciando o sistema de regras. Caso contrário, a mesma seqüência de tópicos dada a um novato é mostrada ao aluno, porém existe a informação de quais tópicos ele já viu. Quando fizer isto, o sistema iniciará o sistema de regras pelo tópico escolhido e, após ter dado o tópico, armazenará no Modelo do Estudante todas as informações obtidas na revisão dos tópicos tal como se fosse um tópico ainda não estudado. Caso o aluno escolha um tópico que foi supostamente visto, o sistema de regras é acionado e o tutor, após ter dado o tópico, atualiza o campo de Conceitos Supostamente Vistos retirando os conceitos relacionados ao tópico dado, pois a partir de então, serão Conceitos Vistos. E, por fim, se o aluno escolher como tópico inicial da sessão um tópico que está situado à frente do que seria aconselhado (segundo o Nível Tópicos e o Modelo do Estudante), o sistema atualiza o Modelo do Estudante com a informação de que os conceitos pertinentes aos tópicos entre o último visto e o que será dado são supostamente vistos, e o sistema de regras é acionado para dar o tópico escolhido.

Uma sessão consiste, basicamente, em mostrar ao aluno os conceitos e resultados específicos do tópico em estudo, permitir que ele veja exemplos e que faça exercícios.

Os conceitos e resultados são determinados através da sub-rede do domínio relacionado ao tópico escolhido. Após a escolha do tópico, o sistema determina em que ordem serão apresentados os conceitos e resultados, preparando sua aula. As relações pertinentes a cada classe, mais as relações duais, permitem que o tutor determine um percurso diferente, caso necessário, para um outro aluno (isto é, o tutor pode apresentar os conceitos e resultados em ordens diferentes para alunos diferentes).

3.2.4 O Módulo de Comunicação

O Módulo de Comunicação do ARQTEMA não possui, por enquanto, uma representação explícita do conhecimento sobre a forma de apresentação do material instrucional tal qual sugerido na arquitetura básica de um STI.

O primeiro protótipo de STI em construção, que se utiliza do ARQTEMA, tem uma interface construída através do Andrew Toolkit (ATK). Este é um sistema que auxilia os programadores na construção de programas complexos, com multimídia, para a comunicação com o usuário [BOR90][HAN88]. Este foi um dos fatores mais importantes na escolha de tal ferramenta.

O aluno pode interagir com o protótipo, através do teclado e do mouse, selecionando itens em menus, manipulando ícones, janelas, *buttons*, entre outros recursos de uma interface gráfica.

Esta interação conta, também, com explicações fornecidas pelo INTEMA [MAL93] [MAL94], um submódulo de interação destinado a processar perguntas/respostas do/para o usuário.

3.3 Conclusões Parciais

Neste capítulo foram apresentados o modelo de Michener e o ARQTEMA: arquitetura genérica para STIs no domínio da Matemática. Como pode ser observado, o modelo de Michener é uma forma de representar e manipular conhecimentos no domínio da Matemática. Ele oferece riqueza de representação e precisão suficientes para especificar uma base de

conhecimento para matemática. O ARQTEMA faz uso deste modelo para construir o Módulo do Domínio que compõe a arquitetura genérica para STIs no domínio da Matemática.

A generalidade proporcionada pelo ARQTEMA limitou algumas características desejáveis em um STI, como, por exemplo, a resolução de problemas baseada em seu próprio conhecimento. Contudo, é possível incorporar à arquitetura um módulo que resolva problemas da teoria - um resolvedor de problemas. Acoplando este módulo ao ARQTEMA, obtém-se um tutor mais eficiente, pois ele permite que o aluno possa, por exemplo, entrar com sua própria resposta que seria analisada ao invés de escolher uma das alternativas propostas pelo tutor. Isto faz com que o aluno fique livre em relação às respostas que deseja dar.

No capítulo seguinte será apresentado o projeto TOOTEMA e sua arquitetura, descrevendo a estrutura e a função de cada um dos módulos que a compõem.

4

TOOTEMA: Uma Ferramenta para a Construção de STIs em Matemática

No capítulo anterior, foi descrito o modelo de Michener, que permite representar qualquer teoria matemática, e uma extensão para seu uso em STIs, cujo domínio seja a Matemática. Em consideração a esse fato, é possível projetar e desenvolver ferramentas - denominadas *shells* - que permitam a troca de subdomínios (lógica matemática, geometria, álgebra, etc.) a serem ensinados. Para tal, esta ferramenta utilizaria o núcleo básico de um STI no domínio da Matemática e possibilitaria o preenchimento da base de conhecimento com o subdomínio escolhido.

Neste capítulo, serão apresentados os requisitos e as especificações necessárias para construir uma *shell* de STI no domínio da Matemática, denominada TOOTEMA, que utiliza a arquitetura do ARQTEMA e que tem por objetivo auxiliar na tarefa de criar e desenvolver novos subdomínios.

4.1 Introdução e Objetivos

Atualmente os STIs são construídos por especialistas, que têm um bom conhecimento dos princípios e técnicas de **IA**, e que possuem profundo conhecimento do domínio a ser inserido na base de conhecimento do sistema. Entretanto, a idéia por trás de uma *shell* é que o usuário não precisa conhecer detalhes de implementação de STIs para ser capaz de construir um.

A idéia de utilizar ferramentas para construir sistemas intensificou-se a partir do desenvolvimento de Sistemas Especialistas. As *shells* normalmente são construídas via extração e generalização do mecanismo de controle e do esquema de representação de um Sistema Especialista existente. EMYCIN, para citar um exemplo apenas, é uma *shell* derivada do MYCIN [CLA88] - um sistema de diagnóstico para doenças sanguíneas raras. As *shells* dão ao problema uma estrutura melhor e são muitos úteis se o problema a ser resolvido tiver características semelhantes às do Sistema Especialista original. Ou seja, EMYCIN é muito eficaz na modelagem de problemas de diagnósticos, pois este era o principal objetivo do MYCIN.

Os STIs, por se tratarem de programas semelhantes aos Sistemas Especialistas, onde a base de conhecimento refere-se a um domínio específico a ser aprendido, também podem ter seu desenvolvimento baseado numa ferramenta *shell*, com algumas particularidades. Por exemplo, o sistema GUIDON [CLA83] [CLA88] explora as vantagens e limitações do MYCIN, com sua arquitetura baseada em regras.

Com o uso dessas ferramentas, não é necessária a preocupação com técnicas de programação mais complexas, o que normalmente ocorre no desenvolvimento de sistemas desse porte. Essas ferramentas visam também obter a redução dos custos de produção e tempo de desenvolvimento, bem como a melhoria da qualidade e confiabilidade do *software* produzido.

Usuários de *shells* para Sistemas Especialistas, tais com EXPERT, UNITS e EMYCIN, implementam uma base de conhecimento para um novo domínio e não um sistema completo [RIC83]. O mesmo pode ocorrer com as *shell* para STIs, trocando a base de conhecimento no Módulo do Domínio para um novo domínio.

O presente trabalho tem por objetivo especificar e implementar uma *shell* que ofereça estrutura básica de um STI para o domínio matemático, de modo a permitir ao usuário - um professor, por exemplo - definir um subdomínio a ser ensinado, através de uma interface simples e informativa, deixando o desenvolvedor livre de detalhes de implementação, e que garanta um bom nível de qualidade.

4.2 Requisitos do ARQTEMA

Em qualquer projeto de sistema, é relevante fazer um levantamento das características desejadas para o sistema. No caso de um STI, pode-se citar entre outros requisitos:

a) **Representação explícita do conhecimento**: para o sistema ser capaz de explicar o próprio comportamento e para facilitar alterações na base de conhecimento.
b) **Conhecimento procedimental parametrizado**: o sistema deve fornecer facilidades para permitir readaptação quando alterado, sem que isto exija do responsável pela alteração da base de conhecimento conhecimento do mecanismo interno de funcionamento do STI.
c) **Modularidade**: este requisito evita que os componentes de um STI fiquem altamente acoplados, ou seja, que os códigos dos componentes fiquem espalhados por várias partes do sistema.

O ARQTEMA possui a estrutura e a maleabilidade acima mencionadas; seu conhecimento no Módulo do Domínio tem a forma de uma rede de conhecimento complexa, baseada no Modelo de Michener apresentado no capítulo anterior. Esta rede de conhecimento pode ser vista como uma forma estruturada de representação do conhecimento, que permite uma certa independência do domínio através do formalismo existente no modelo, ou seja, este modelo permite representar e manipular conhecimentos no domínio da Matemática.

O Modelo do Estudante, por estar relativamente ligado ao Módulo do Domínio, também foi projetado para permitir readaptação a outros domínios. Ele é formado por parâmetros qualitativos, escolhidos através de observações feitas sobre as necessidades primordiais do sistema, sempre com o objetivo de possibilitar a individualização do ensino.

O conhecimento no Módulo Tutorial é representado através de regras de produção estruturadas, que podem ser utilizadas para modelar qualquer procedimento calculável. Estas regras foram desenvolvidas para que a aula ministrada pelo sistema cubra todos os itens necessários, segundo o modelo. Além da facilidade de modelagem, as regras de produção permitem que novas regras possam facilmente ser acrescentadas para satisfazer novas situações sem perturbar o resto do sistema, e são uma boa maneira de modelar os tipos de ações inteligentes fortemente movidas a dados, pois, à medida que novas entradas são acrescidas ao Modelo do Estudante, o comportamento do sistema modifica-se. [RIC83]

A interface do ARQTEMA é uma interface gráfica composta por menus e janelas, como mostra a Figura 4.1, e foi implementada no **Andrew ToolKit** [BOR90] [HAN88].

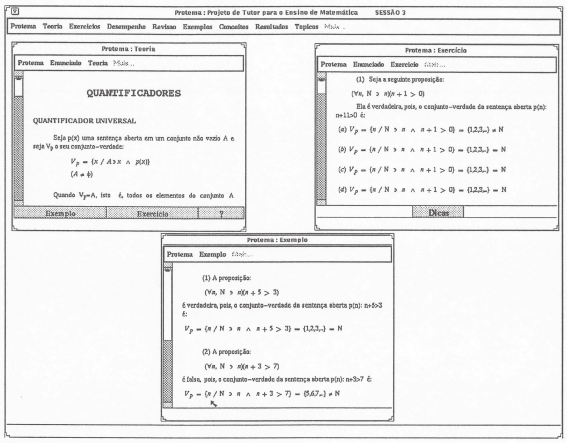


Figura 4.1: Tela de atividades do PROTEMA.

4.3 Recursos Utilizados

Da mesma forma que existem muitas considerações de projeto no desenvolvimento de um STI, também existem muitas questões de implementação, tais como a escolha da linguagem de programação, a otimização de desempenho, a minimização de tempo e o custo de desenvolvimento, entre outras.

Qualquer linguagem pode ser utilizada para desenvolver o código executável de qualquer STI. Não é necessária uma linguagem de **IA**. Entretanto, algumas linguagens fornecem rapidez e facilidades no desenvolvimento de STIs, enquanto outras fornecem melhores desempenhos. Algumas linguagens são padronizadas e disponíveis em diversas máquinas, tanto de pequeno como de grande porte. Algumas linguagens necessitam de um interpretador, enquanto outras são compiladas.

Considerando as questões acima mencionadas e os requisitos do ARQTEMA, a implementação do protótipo do TOOTEMA também foi realizada utilizando a linguagem de programação C e o **Andrew ToolKit**. C é uma linguagem relativamente de baixo nível, da qual eficientes compiladores são amplamente disponíveis. Entretanto, a linguagem C não

fornece bom suporte para programação de interfaces do usuário e para tipos abstratos de dados.

O **Andrew User Interface System** possui diversas características que auxiliam a suprir estas desvantagens. Ele fornece um ambiente orientado a objeto, onde cada objeto é dinamicamente carregado, permitindo a diminuição do tempo de desenvolvimento, um pré-processador que permite abstrações de alto nível para serem compilados com códigos eficientes em C, um mecanismo de herança, uma rica biblioteca de objetos, bem como facilidades que permitem a construção de interfaces. Também é extremamente portável, podendo rodar em vários ambientes que rodem sobre o X11, e ser transferido para vários sistemas operacionais (Telmat, Linux, Solaris, SGI, entre outros) e plataformas (IBM, SPARCstation, HP, Dec, Apollo, SCO e PC).

Portanto, o **Andrew** é um sistema que auxilia os programadores a construírem programas complexos de multimídia [BOR90] [BOR91]. Ele foi projetado com o objetivo de dar estimulo, principalmente, a quatro áreas da educação universitária [MOR86]:

a) no desenvolvimento de CAIs, através da utilização de estações gráficas de trabalhos;
b) na criação e uso de novas ferramentas;
c) na comunicação através do uso de novas técnicas de multimídia e
d) no acesso à informação.

Algumas vezes é desejável usar o computador para fazer apresentações de materiais como se faz com os livros. O computador tem a vantagem de oferecer animações e outras interações com o aluno. A apresentação pode ser controlada pelo autor e o aluno pode ser questionado para garantir a compreensão de algum conceito antes de prosseguir com a apresentação. Desta maneira, uma apresentação controlada do material é chamada de aula.

Construir aulas com as tradicionais linguagens de programação pode ser uma tarefa tediosa. Imagens devem ser construídas através de primitivas gráficas de mais baixo nível, um seqüenciamento tem que ser manipulado em todos os seus detalhes e a interação com o usuário tem que ser implementada para cobrir uma grande diversidade de entradas.

A essência dos programas multimídia é que eles permitem a livre mistura de vários meios de representação e apresentação da informação. Assim, texto, gráfico, animação, som, entre outros, são vistos como diferentes meios de representação e possuem mecanismos próprios para tratá-los. Por exemplo, o **Andrew** inclui um editor genérico de objetos, chamado **EZ**, para editar textos que, além de conter diversas fontes de letras, pode conter imagens *raster*, tabelas, desenhos, equações, animações simples, entre outros. Esses objetos também podem conter outros objetos, incluindo textos. As Figuras 4.2 e 4.3 mostram telas editadas e manipuladas pelo **EZ** com diversos tipos de meios.

No **Andrew**, as classes dos objetos são criadas para serem dinamicamente carregadas, ou seja, as classes de objetos são carregadas na memória algum tempo depois do programa principal ter sido inicializado. Nos programas convencionais em C, todos os códigos são ligados por um *linker*, que cria um único programa executável contendo todas as instruções necessárias para executá-lo. Entretanto, em um programa dinamicamente carregável, os módulos são carregados em memória conforme a sua necessidade.

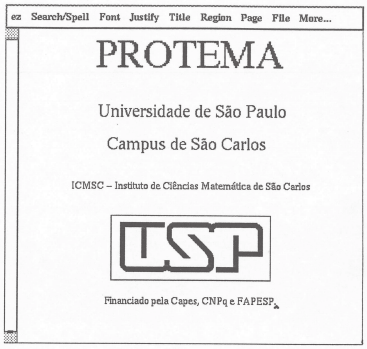


Figura 4.2: Texto com a imagem **raster** (Logotipo da USP) criado pelo **EZ**.

Isto acarreta diversos benefícios. Primeiro, o programador não perde seu tempo esperando o seu programa ser "linkado". Segundo, a carga dinâmica permite o compartilhamento de códigos, ou seja, cada objeto dinamicamente carregável pode ser utilizado por diversos programas sem que necessite ser recompilado. Terceiro, a carga dinâmica torna o programa mais fácil de expandir e alterar, ou seja, o programa pode ser modificado de maneira significativa sem que seja necessário a recompilação dos códigos básicos que formam a biblioteca de objetos do **Andrew**. Finalmente, este esquema possibilita um ganho de tempo de execução em muitas versões do sistema operacional UNIX.

Existem também aspectos negativos da carga dinâmica. O mais notável deles é o perigo de versões incompatíveis, ou seja, quando as especificações de alguma classe são alteradas, todos os objetos que utilizam esta classe devem ser recompilados. Um outro possível problema ocorre quando dois objetos com o mesmo nome são criados em diretórios diferentes, o que torna impossível utilizar os dois objetos em um mesmo processo.

Uma diferença entre a programação orientada a objetos e a tradicional está na facilidade de fornecer uma plataforma útil para construir protótipos rapidamente. Tradicionalmente, a tarefa de escrever um grande programa utiliza a abordagem *top-down*,

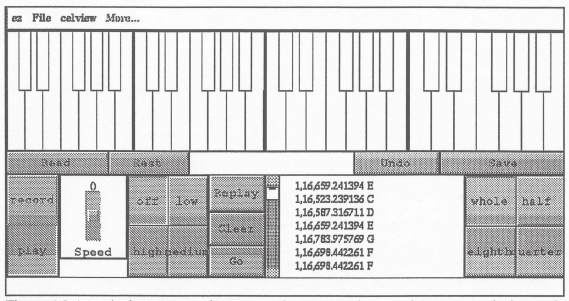


Figura 4.3 Através do **Arbcon**, criou-se um piano que emite som das notas musicais e pode ser utilizado pelo editor **EZ**.

na qual o programa é construído de maneira unificada, enquanto a programação orientada a objetos pode ser descrita como uma abordagem *bottom-up*, no sentido que primeiro se constrói e testa os componentes antes de determinar a melhor maneira de combiná-los para formar o programa. A programação orientada a objetos permite que grandes programas sejam decompostos em programas pequenos e independentes, cada qual descrevendo o comportamento de algum objeto e interagindo entre si através da troca de mensagens.

Todos esses recursos foram úteis para a especificação e o desenvolvimento do TOOTEMA. Além disso, as características do **Andrew**, incrementadas com as do TOOTEMA, permitem que o desenvolvedor elabore e desenvolva eficientes bases de conhecimento. Por exemplo, na elaboração de cada aula, o desenvolvedor poderá trabalhar interativamente com a *shell*, editando os textos, gráficos, animações no Editor de Documentos e assim tornar a apresentação mais rica.

No entanto, o TOOTEMA foi o primeiro software desenvolvido no ICMSC-USP sobre o **Andrew**, o que exigiu um esforço adicional na implementação da ferramenta, principalmente pelo fato de terem sido utilizados recursos complexos de multimídia que requerem bastante familiarização.

4.4 Estrutura da Ferramenta

Com base no estudo dos requisitos e funcionalidade apresentados por algumas *shells* encontrada na literatura (ver apêndice I), foram definidos os componentes do

TOOTEMA, e foi feita uma descrição que refletisse a funcionalidade e os principais requisitos desejados.

A *shell* é dividida em cinco componentes principais (mostrados na Fig. 4.4): Interface com o Usuário (Professor e Programador), Editores Gráfico e de Documento, *Browser* , Simulador e Montador, que são discutidos a seguir.

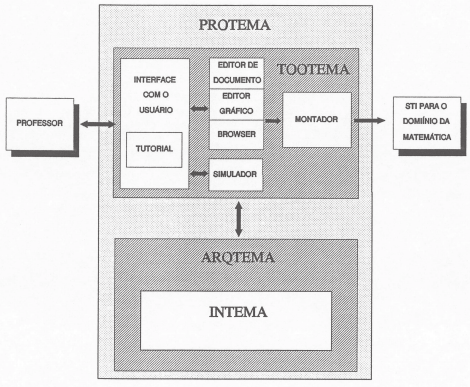


Figura 4.4: Componentes da ferramenta.

4.4.1 Interface com o Usuário - Professor

Espera-se que os principais usuários das *shells* para STIs sejam os professores e os desenvolvedores de cursos. Portanto, para desenvolver um STI eficiente, uma interface com o usuário não deve ser apenas fácil de usar, mas bastante flexível para poder envolver seu usuário na sua tarefa. Com isto em mente, foram considerados quatros requisitos para a construção da *shell*:

1. Possibilitar o desenvolvimento da base de conhecimento em um ambiente bastante familiar, com o objetivo de tornar o desenvolvimento o mais atraente e produtivo possível. Os editores de textos multimídia e os editores gráficos oferecem a maneira mais

básica e direta de inserir e alterar o conhecimento no Módulo do Domínio de um STI, independentemente da forma como este conhecimento esteja sendo representado. Os editores do TOOTEMA, que serão melhor detalhados na seção 4.4.3, têm uma característica de integração que permite que o professor se mova de maneira rápida por entre as fases de testes, depurações e edições, enquanto desenvolve uma parte da base de conhecimento[1].

2. Algumas ferramentas adotam uma abordagem orientada a menus na aquisição de conhecimento e entrada de dados. A vantagem de um sistema baseado em menus é que ele impede que o usuário provoque inconsistências ou altere indevidamente a base de conhecimento. Por outro lado, esta abordagem pode restringir a potencialidade de usuários experimentados. No TOOTEMA, parte do processo de aquisição de conhecimento pode ser orientada a menus. Por exemplo, o professor pode fazer a ligação entre os nós utilizando a opção "Ligar Itens" do submenu do menu principal "Gráfico", que então lhe oferece as opções de nós candidatos, ou alternativamente o *mouse*, selecionando em um nó e arrastando o *mouse* até o segundo nó a ser ligado.

3. Uma outra importante característica que se deve considerar para a interface com o usuário é a habilidade da ferramenta em mostrar visualmente a lógica e a estrutura da base de conhecimento. Esta característica é crítica no desenvolvimento do Módulo do Domínio, pelo fato de descrever como uma aula será apresentada ou como uma consulta será conduzida. No TOOTEMA, além do Editor Gráfico que permite a visualização da estrutura da rede de conhecimento, existe o Simulador de aula. Nele o professor pode visualizar como um determinado tópico do domínio será apresentado a um suposto aluno. A sua utilização permite que se altere e corrija o conteúdo dos nós e as relações entre os nós do tópico durante a simulação.

4. Finalmente, uma interface deve oferecer informações que ajudem o usuário a resolver grande parte de seus problemas. O Tutorial é um dos componentes da interface do TOOTEMA que tem por objetivo fornecer informações sobre como utilizar, da melhor maneira, todos os componentes da *shell* (Fig. 4.5). Ele mostra, através de exemplos, como especificar um determinado conhecimento para construir a rede de conhecimento. Além disso, dá ao usuário desta *shell* suporte para localizar facilmente informações relevantes, e adicionar, eficientemente, novas informações em harmonia com as convenções existentes.

[1] A base de conhecimento compreende não apenas a rede de conhecimento (Módulo do Domínio), mas as regras tutoriais (Módulo Tutorial) e as informações sobre o aluno (Modelo do Estudante).

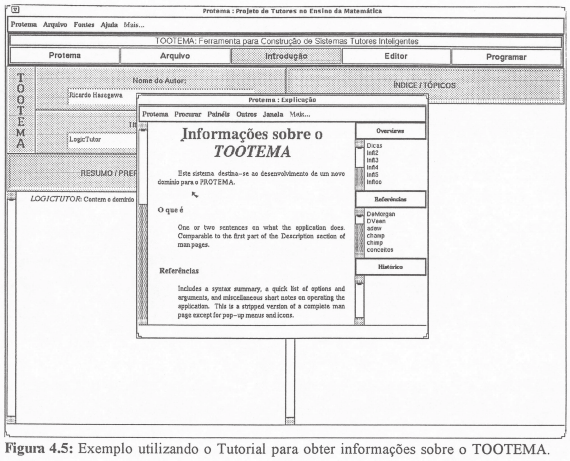


Figura 4.5: Exemplo utilizando o Tutorial para obter informações sobre o TOOTEMA.

4.4.2 Interface com o Usuário - Programador

O TOOTEMA provê uma interface projetada especialmente para usuários com conhecimento em linguagem de programação C.

Interfáceis amigáveis têm a reputação de serem de difícil desenvolvimento. Pensando nisso, foi desenvolvida uma maneira do programador desenvolver seus aplicativos sem grandes dificuldades, utilizando, por exemplo, a biblioteca de objetos do ambiente, bem como os procedimentos criados pelo próprio programador. Isto é possível graças à interface com o programador, que utiliza, entre outras coisas, uma ferramenta do **Andrew**, denominada **Arbcon**.

O **Arbcon** permite que um programador selecione objetos entre os diversos tipos de objetos da biblioteca da *shell*, e agrupe-os para criar a interface de uma nova aplicação. Todo esse processo é realizado utilizando apenas as facilidades oferecidas pela interface.

Após o programador criar a interface da sua aplicação, é possível gerar o código fonte do protótipo na linguagem C, bem com o arquivo de especificação para definir a classe dessa aplicação.

A ferramenta permite que sejam feitas mudanças na aplicação. Assim, se o programador decidir fazer novas adições de objetos, ou trocar um objeto por outro, a ferramenta fará as atualizações do código fonte e o tornará compatível com a nova coleção de objetos. Para isso, o código fonte gerado contém muitos comentários da forma:

```
/* codigo do usuario comeca aqui para exemploCallBack */
/* codigo do usuario termina aqui para exemploCallBack */
```

É entre estes comentários que todo o código escrito pelo programador deve ser adicionado. O **Arbcon** utiliza estes comentários para determinar quais códigos ele deve preservar quando criar uma nova versão do código. Isto é necessário para que o sistema saiba quais foram os códigos gerados pelo sistema e quais foram acrescentados pelo usuário. Assim, se o programador decidir adicionar um novo objeto, o **Arbcon** irá mover o velho código em C para um arquivo de *backup* e irá adicionar todo o código escrito pelo programador para o novo código fonte.

O principal objetivo da interface com o programador é permitir que qualquer programador possa enriquecer o PROTEMA com aplicações bastante complexas, mas sem a necessidade de conhecer o complicado funcionamento do paradigma que envolve todo o sistema.

Esta interface é dividida em três partes importantes: **Arbcon**, Janela de trabalho e Editor de código para a linguagem C. (Fig. 4.6)

Com ela é possível que o programador crie aplicações com interfaces gráficas, através de técnicas de orientação a objetos e multimídia, e faça os arranjos e as ligações de diversos objetos dentro das aplicações.

4.4.3 Editores e *Browser*

Os usuários do TOOTEMA precisam de suporte para localizar, adicionar e alterar informações na base de conhecimento. Além disso, alguns componentes que permitam o usuário tomar decisões durante a implementação do sistema se fazem necessários. A seguir, são descritos os componentes de edição que permitem ajudar o usuário a resolver estes problemas.

De uma maneira geral, existem duas formas para armazenagem e recuperação de informações numa base de conhecimentos:

a) através de edições e consultas diretas à base de conhecimentos, utilizando-se, por exemplo, um mecanismo capaz de exprimir as características da informação

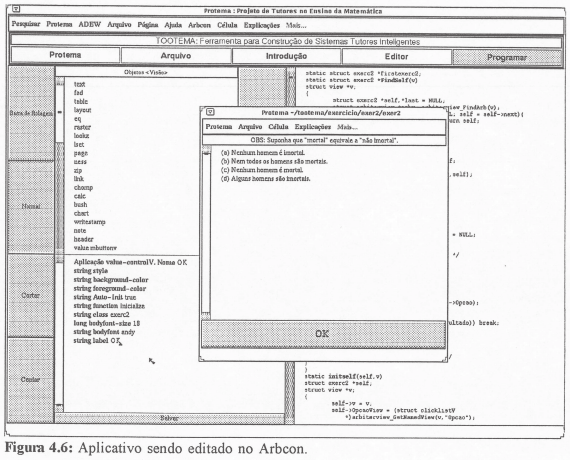


Figura 4.6: Aplicativo sendo editado no Arbcon.

a ser acrescentada ou procurada, e

b) através da navegação sobre a estrutura organizacional da base à procura de informações com características ainda não muito bem estabelecidas.

Esses dois métodos não são exclusivos, podendo ser utilizados conjuntamente, de forma a se complementarem. Por exemplo, a partir de uma primeira tentativa de consulta, começa-se a percorrer a base à procura de conceitos, exemplos, resultados, exercícios e *bugs* mais específicos, ou, da mesma forma, a partir de uma primeira visualização das características encontradas em conceitos, exemplos, resultados, exercícios e *bugs* observados durante um processo de navegação, opta-se pela definição de uma consulta mais específica. Os principais componentes que desempenham estas funções são os Editores e o *Browser*.

Os Editores

Os Editores são basicamente dois: o Editor de Documentos e o Editor Gráfico. O Editor de Documentos é um editor multimídia, que permite que as informações na rede de

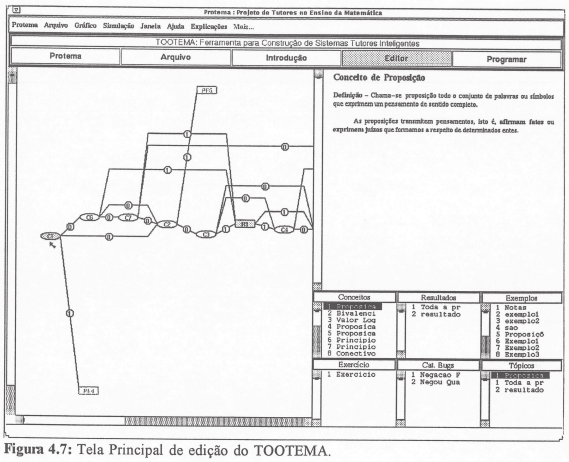


Figura 4.7: Tela Principal de edição do TOOTEMA.

conhecimento possam ser criadas e modificadas na forma de texto, imagem digitalizada, animação, entre outros. A Figura 4.7 mostra um nó-documento sendo inserido na rede de conhecimento.

O Editor de Documentos não só é um editor multimídia. Ele também pode ser utilizado para criar e modificar a estrutura da rede de conhecimento. Esta é uma das características que permitem visualizar a informação contida em cada nó da rede de conhecimento, bem como alterar o seu tamanho. A quantidade de informação contida em cada nó foi uma preocupação retirada dos conceitos de hipertexto, pois muitos autores de hipertexto utilizam os nós para expressar um simples conceito ou idéia.

O hipertexto obriga o autor a modularizar as suas idéias em unidades, para que essas idéias possam ser referenciadas em qualquer parte do texto. Além disso, o autor, também, deve levar em conta o fato de que o nó do hipertexto, ao contrário do parágrafo de texto, tem que ser uma unidade rigorosa, para ter uma coesão com os nós vizinhos. Alguns sistemas de hipertexto (Notecards, por exemplo) permitem que os nós sejam vistos como um único e grande nó. Entretanto, estes sistemas têm demonstrados que os limites

ao redor dos nós são quase sempre discretos e as vezes requerem um esforço extra para serem encontrados.

O processo de identificar uma unidade semântica, como uma idéia ou conceito, em uma unidade sintática, como um parágrafo ou nó de um hipertexto, não é exclusivo ao hipertexto. Nos textos tradicionais, as convenções são livres para modularizar os textos em parágrafos. Esta liberdade é aceitável pelo fato dos parágrafos terem um pequeno efeito sobre o fluxo de leitura. Os limites dos parágrafos são fornecidos apenas para dividir o texto e dar ao olho um ponto de referência. Assim, decisões sobre a distribuição das sentenças entre os parágrafos nem sempre são críticas, enquanto que no hipertexto elas podem causar um ocultamento de informação. [CON87]

Assim, como a rede de conhecimento é semelhante à estrutura de hipertexto, o Editor de Documentos possibilita que o desenvolvedor primeiro escreva suas idéias e conceitos na forma tradicional, ou seja, seqüencialmente e estruturados na forma de livro, formando um único e grande nó. Depois, seccione este grande nó em outros menores, e reestruture até formar a rede de conhecimentos. Este processo pode ser repetido para cada novo nó criado (Fig. 4.8).

Além disso, o Editor de Documento permite que os diversos objetos da biblioteca e aplicativos criados pelo **Arbcon** sejam apresentados e executados por ele. Esta característica possibilita que o desenvolvedor crie diversos tipos de exercícios para o mesmo conceito ou resultado. Por exemplo, no domínio de Lógica, um exercício poderia comparar as fórmulas proposicionais dadas pelo aluno com alguma fórmula equivalente. Para tanto, o desenvolvedor criaria uma interface no **Arbcon** e acrescentaria o algoritmo de equivalência ao código gerado.

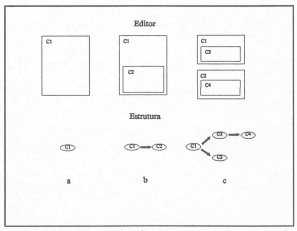


Figura 4.8: Exemplo de como estruturar a rede de conhecimento usando o Editor de Documentos.

As funções do Editor Gráfico foram criadas para facilitar o desenvolvimento da rede de conhecimento. A visualização da rede de conhecimento ajuda a formar uma imagem mental clara e correta da estrutura e funções de um domínio, ou seja, com a visualização gráfica pode-se identificar a estrutura interna da rede em desenvolvimento, seus principais caminhos, associações, medidas de complexidade, entre outros. Além de poder identificar imediatamente anomalias como a falta de estruturação.

Além disso, o Editor Gráfico estimula o desenvolvedor com algumas facilidades, tais como, a utilização do *mouse* e menus para criar e alterar os nós e as ligações da rede de conhecimento. Assim, se o desenvolvedor selecionar com o *mouse* qualquer região da janela do Editor Gráfico e alterar a estrutura da rede de conhecimento, o grafo por ele alterado é instantaneamente redesenhado na janela do editor com a melhor forma que este possa apresentá-lo. Outra facilidade é a dimensão do desenho poder ser redefinida, conforme a necessidade do desenvolvedor.

Durante o desenvolvimento do Editor Gráfico, preocupou-se, também, com os aspectos estéticos do diagrama por ele desenhado. Alguns aspectos estéticos são requisitos básicos, como a minimização dos cruzamentos de arcos e da área ocupada pelo desenho.

A estratégia adotada para o roteamento dos arcos levou em conta algumas de suas características ligadas principalmente ao seu tipo. Arcos curtos têm roteamento trivial e para os outros tipos de arcos foi definido um algoritmo que determina as opções possíveis para o posicionamento e escolhe a melhor.

Muitos algoritmos de posicionamento de nós de grafos utilizam uma estratégia de redefinir o posicionamento de nós já posicionados à medida que é requisitado mais espaço para os subgrafos desses nós; essa abordagem desperdiça tempo pois é necessário reposicionar todos os antecessores de um nó que teve a sua posição alterada; na abordagem utilizada no Editor Gráfico esse problema é eliminado através do uso da variável denominada escopo; com essa variável, pode-se determinar diretamente as posições dos nós sem haver repetição de cálculos. Além disso, os grafos gerados tendem a ser equilibrados pois devido à estratégia para o cálculo e uso da variável escopo os subgrafos mais densos tendem a ficar mais internos à estrutura do grafo.

Normalmente, mais de um aspecto estético é considerado ao mesmo tempo em aplicações reais, podendo um desenho adequar-se perfeitamente a um aspecto estético, mas ser reprovado com relação a outro. Por exemplo, alguns exemplos mostram que a simetria é mais importante que o não cruzamento dos arcos, porque a simetria é uma característica importante das estruturas.

O Editor Gráfico também impõe algumas restrições, como por exemplo, toda vez que o desenvolvedor criar uma ligação que forme ciclos, o Editor o alerta sobre o ciclo e não conclui a operação. E se o desenvolvedor inadvertidamente "clicar" três vezes com o botão direito do *mouse* sobre um nó ou ligação (operação para destruir o nó ou ligação), o Editor Gráfico envia uma mensagem perguntando se ele realmente deseja concluir a operação.

A janela do Editor Gráfico (retângulo maior com dupla barra de rolagem à esquerda do Editor de Documentos e das listas de itens, na Figura 4.7) que ocupa quase a metade da tela, pode, como todas as janelas do TOOTEMA, ter seu tamanho redefinido movendo suas bordas com o **mouse**.

O *Browser*

O *Browser* é um recurso particular de interface, criado para facilitar a interação entre usuários e a rede de conhecimento. As facilidades fornecidas pelo *Browser* permitem que se trabalhe com grande quantidade de informações sem se perder. Ele é utilizado para executar buscas exploratórias e permitir visualização dos conceitos, exemplos, resultados, exercícios e *bugs* de uma rede de conhecimento, bem como seus inter-relacionamentos.

Por ser uma alternativa para fazer busca formal e parametrizada, o *Browser* é utilizado para auxiliar aqueles que não estão familiarizados com a estrutura da rede de conhecimento. Esse uso tutorial ajuda o desenvolvedor a encontrar informações que ele necessita, bem como os termos que são usados em um determinado domínio.

Existem diversos sistemas orientados a textos, como por exemplo, hipertexto, que utilizam buscas exploratórias. Um dos primeiros sistemas, o ZOG e sua versão comercial - KMS, foi projetado para ser uma interface de propósito geral [AKS88]. Sua estrutura é organizada na forma de árvore e composta por unidades de informação, denominadas *frames*. As buscas são realizadas caminhando-se entre os *frames* até encontrar a informação desejada. As vantagens do ZOG estão na facilidade de uso que requer pouco treinamento, e na rapidez de execução. A desvantagem provém da falta de capacidade gráfica que mostre ao usuário a sua localização no contexto de toda a informação armazenada na estrutura.

Baseando-se nesta vantagem, o *Browser* foi projetado com o objetivo de ser facilmente utilizado. Por exemplo, buscar, criar, alterar, visualizar ou apagar as informações de cada item da rede de conhecimento e as variáveis neles definidas são funções que podem ser realizadas, bastando que o desenvolvedor, com o *mouse*, indique qual o item a ser selecionado, tanto na lista de itens, como no diagrama que contém os nós que representam os itens na janela do Editor Gráfico.

Quando o usuário utiliza o Editor Gráfico para disparar as funções do *Browser*, é possível visualizar a localização deste item em relação a toda informação armazenada na rede de conhecimento e visualizar o seu conteúdo na janela do Editor de Documentos para ser inspecionado ou alterado.

4.4.4 Simulador

O Simulador tem por objetivo mostrar ao desenvolvedor como uma dada aula será ministrada pelo sistema. Com ele, é possível ver a coesão entre os diversos itens da rede de conhecimento que forma a aula, ou seja, a ordenação topológica da rede de conhecimento. Para tanto, ele utiliza as regras tutorias e o procedimento de preparação de

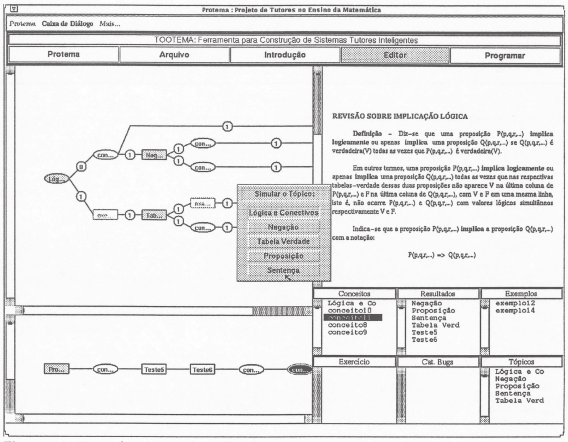


Figura 4.9: Exemplo mostrando o Simulador em operação.

aula, encontrados no Módulo Tutorial, e algumas informações contidas no Modelo do Estudante, tais como: nível de desempenho do aluno, conceitos supostamente vistos, conceitos vistos anteriormente e conceitos vistos, e a rede de conhecimento do Módulo do Domínio.

Para fazer a simulação não é preciso que toda a rede de conhecimento esteja implementada, mas apenas a estrutura do tópico a ser simulado, ou seja, os nós e ligações que pertencem ao escopo do tópico. Como a simulação é realizada para um suposto aluno, a única informação a ser inserida no Modelo do Estudante é o nível de desempenho do aluno. Essa informação é a única que pode ser alterada durante a simulação, pois o sistema cria e atualiza todas as outras informações do Modelo do Estudante.

As interações com o Simulador são realizadas através da janela desenvolvida especificamente para ele. Nessa janela é possível ter uma visualização gráfica da ordenação topológica da rede de conhecimento. Cada iteração é realizada selecionando-se no ultimo nó ordenado topologicamente (Fig. 4.9).

Portanto, todas as telas foram projetadas para que o desenvolvedor, usando componentes sensíveis ao *mouse* e com poucos comandos, seja capaz de editar, recuperar,

alterar ou simplesmente inspecionar os conceitos, resultados, exemplos, catálogo de *bugs* e exercícios. A interface de edição tem fácil acesso tanto ao *Browser* como ao Simulador, permitindo que o desenvolvedor visualize a estrutura da rede de conhecimento e veja as relações entre os itens da teoria e as regras tutoriais.

A possibilidade de simular uma aula consiste em um dos aspectos inovadores deste trabalho, por permitir ao professor simular uma possível aula para um suposto aluno, enquanto aquele constrói a base de conhecimento em um subdomínio da Matemática. Outro fato inovador é de não terem sido encontrados mecanismos semelhantes de simulação que fornecessem tal facilidade em nenhuma ferramenta pesquisada durante a revisão bibliográfica.

4.4.5 Montador

A função do Montador é montar os procedimentos reusáveis e os elementos armazenados na base de conhecimento necessários para formar o programa. A estes é necessário acrescentar o Modelo do Estudante e qualquer mecanismo de inferência necessário para um determinado subdomínio.

Os procedimentos reusáveis serão quase todos os procedimentos utilizados no sistema ARQTEMA, pelo fato de alguns destes procedimentos serem específicos deste sistema. Para tanto, existe uma biblioteca, que além de conter os procedimentos do ARQTEMA, contém outros procedimentos reusáveis que não estão disponíveis neste sistema. É possível, também, que o desenvolvedor programador enriqueça esta biblioteca, acrescentando novos procedimentos reusáveis específicos de um subdomínio.

Algumas das características do Montador foram herdadas do **Andrew**. Uma delas é a que permite inserir qualquer aplicação, criada pelo **Arbcon**, na base de conhecimento através do Editor de Documentos, e assim, criar uma base de conhecimento multimídia. No exemplo da Figura 4.10 é possível ver um texto contendo uma calculadora criada no **Arbcon**.

4.5 Avaliação Parcial da Ferramenta

A utilização do **Andrew** fez com que o TOOTEMA, bem como o PROTEMA de um modo geral, incorporasse todas as suas características. Assim, o PROTEMA é um sistema multimídia portável para diversos ambientes e plataformas. Além disso, o **Andrew** contribuiu no desenvolvimento de todo o sistema, impondo uma metodologia orientada a objetos e tornando todo o sistema modular, o que facilita as tarefas de desenvolver, manter,

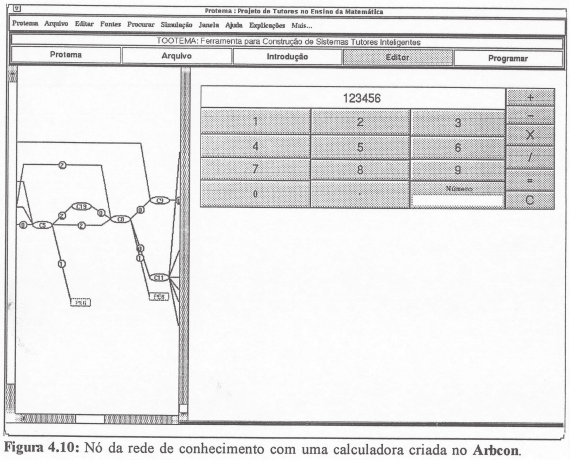


Figura 4.10: Nó da rede de conhecimento com uma calculadora criada no **Arbcon**.

modificar e estender o mesmo.

Durante o desenvolvimento do TOOTEMA, uma das grandes preocupações do trabalho foi com a interface do usuário. Diversas "*guidelines*" sobre interface do usuário, que devem ser consideradas para que uma ferramenta tenha um bom desempenho na usabilidade ([AKS88] [MUR92b]), foram consideradas durante o decorrer deste trabalho, a saber:

a) Tanto quanto possível, antecipe as capacidades e as limitações cognitivas de usuários típicos.

Para não sobrecarregar o usuário com muitos conceitos, deve-se ter um ambiente simples, direto e fácil de usar. Por exemplo, a interface foi desenvolvida para que grande parte da interação seja feita através do *mouse* e os seus três botões. Além disso, o TOOTEMA é capaz de fazer, a partir do item introduzido pelo usuário, algumas ligações automaticamente com os itens anteriormente digitados e que formam a rede de conhecimento, bem como tem a caraterística de dificultar operações inadequadas.

b) Permita movimentos e interações rápidas entre o teste e a modificação.

Quanto mais rápida for a resposta do sistema, melhor será a interação com o usuário. Durante o teste do TOOTEMA, observou-se que a resposta do sistema é dada em tempo bastante aceitável.

c) Forneça ao usuário um modelo cognitivo claro e refinado da estrutura básica.

Se o usuário desejar, é possível, através do Editor Gráfico e do *Browser*, visualizar a rede de conhecimento de múltiplas maneiras, e ver as relações entre os itens na rede de conhecimento. O Editor Gráfico também permite alterações das ligações e o *Browser* permite investigações a procura de uma melhor coesão entre as informações contidas nos itens da rede.

d) Ajude o usuário a se localizar e a "navegar" dentro da base de conhecimento.

Um dos problemas clássicos dos sistemas de hipertexto/hipermídia é a desorientação. Isso acontece quando a quantidade de informação contida na base de conhecimento é muito grande. Esse problema é resolvido no TOOTEMA através do Editor Gráfico e do *Browser*, que permitem ao usuário formar uma imagem mental clara e correta da estrutura lógica da base de conhecimento e assim reduzir a desorientação.

e) Forneça assistências e referências *on-line*.

O TOOTEMA possui um tutorial que tem por finalidade fornecer informações sobre possíveis problemas que o usuário possa ter.

Portanto, o ambiente de interação do TOOTEMA, para se criar um novo subdomínio para STIs, é formado de janelas, ícones, menus, gráficos, gravuras, entre outros elementos. As ferramentas com as quais o usuário interage incluem os Editores, *Browser*, **Arbcon** e Tutorial, com os quais o desenvolvedor se comunica através de menus, ícones, preenchimento de campos vazios, movimentos com o *mouse*, etc.

5

Exemplos de Interação com a Ferramenta

Neste capítulo são apresentados alguns exemplos que mostram a forma de interação entre o usuário e a ferramenta TOOTEMA. Tem-se como meta fornecer uma visão geral do funcionamento da ferramenta.

O exemplo utilizado está no domínio da Lógica, que corresponde a um trabalho desenvolvido durante a especificação da arquitetura do PROTEMA [TAK94]. Para facilitar o entendimento, é utilizado um conjunto reduzido de dados, ou seja, a rede de conhecimento que forma o exemplo deste capítulo foi catalogada e organizada conforme o modelo proposto por Michener e abrange apenas um tópico do domínio da Lógica Matemática.

Como visto no capítulo quatro, a interface do TOOTEMA foi desenvolvida para interagir com dois tipos de usuários. Dessa forma, são apresentados exemplos de interação com o usuário professor e com o usuário programador.

A seguir será ilustrada a interação do usuário professor com o ambiente, utilizando o exemplo no domínio de Lógica.

5.1 Interação com o Usuário Professor

Ao ativar o sistema, é apresentada, ao usuário professor, a janela que contém uma barra de título, uma barra de menus e a janela com o logotipo animado do projeto PROTEMA (Fig. 5.1).

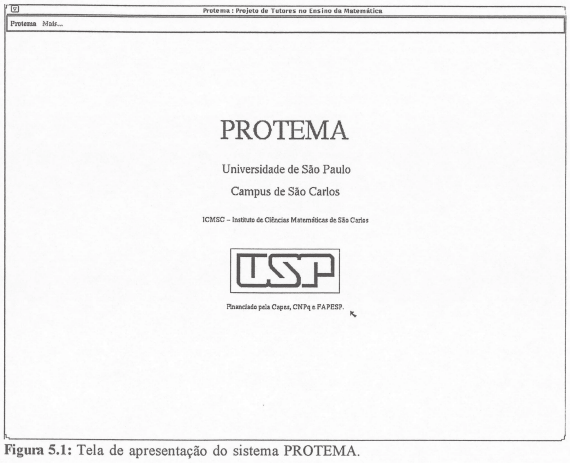


Figura 5.1: Tela de apresentação do sistema PROTEMA.

Posteriormente aparece a tela que possui a palavra PROTEMA, dois botões e uma barra de botões. O dois botões permitem a escolha entre o TOOTEMA e o TEMA - Tutor para o Ensino de domínios da MAtemática. A barra de botões apresenta três botões com os rótulos "Informações", "Voltar", e "Sair". O botão "Informações" tem por função fornecer informações sobre a janela em uso. O botão "Voltar" permite voltar para a última tela vista. Finalmente, o botão "Sair" permite o professor interromper a execução e finalizar o sistema.

Para o professor utilizar o TOOTEMA, basta selecionar o botão TOOTEMA com o botão esquerdo do *mouse*. Após esta operação, a tela que aparece é semelhante à anterior, sendo que, no lugar da palavra PROTEMA, aparece a frase "TOOTEMA Ferramenta para

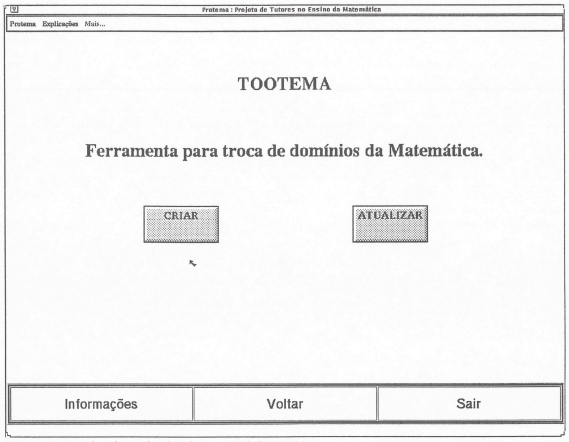


Figura 5.2: Primeira tela do sistema TOOTEMA.

troca de domínios da Matemática", e dos dois botões permitem a escolha entre "Criar" um novo domínio ou "Atualizar" um domínio existente no sistema (Fig. 5.2). As telas utilizadas são sensíveis ao contexto, ou seja, sempre que o cursor do *mouse* mudar de formato em determinadas regiões, é possível obter informações clicando nessas regiões. Por exemplo, se o professor selecionar através do *mouse* na palavra TOOTEMA, é possível ele obter algumas informações sobre esta ferramenta.

Como o objetivo deste exemplo é ilustrar a criação de um novo domínio, o professor opta pelo botão Criar. Se fosse escolhido o botão Atualizar, o sistema apresentaria uma lista com vários domínios implementados e que poderiam ser atualizados. A única diferença entre este botão e o de Criar está no fato de que o segundo não carrega nenhuma informação na base de conhecimento.

Após clicar no botão de criação de um novo domínio, o professor visualiza a janela mostrada na Figura 5.3. Esta janela contém uma barra de botões, dois campos de preenchimento, um editor de notas e um índice de tópicos. A barra de botões tem por finalidade facilitar o acesso aos diversos ambientes de trabalho. Os dois campos de preenchimento são utilizados para introduzir os nomes dos autores do novo domínio e o

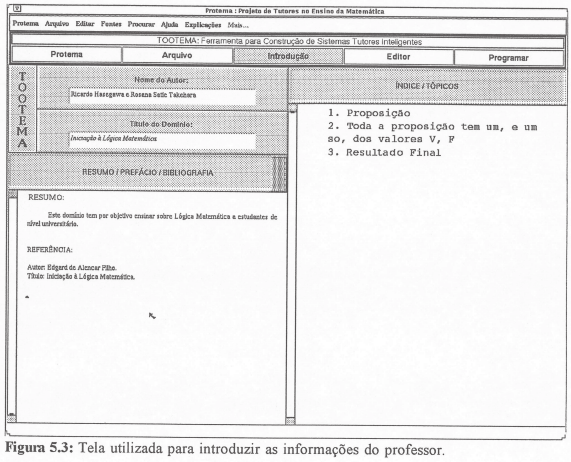


Figura 5.3: Tela utilizada para introduzir as informações do professor.

título do domínio. O editor de notas pode ser utilizado para introduzir algumas informações sobre o novo domínio, tais como, as fontes de informações consultadas - referências a livros, professores, entre outros - utilizadas para construir o domínio, algum resumo e seus objetivos. As informações têm por objetivo ajudar outros usuários (professores) quando estes desejarem atualizar a rede de conhecimento. Finalmente, o índice de tópicos é uma opção que pode ser utilizada para montar ou visualizar apenas a estrutura do nível tópicos. Mesmo que não seja utilizado, o sistema atualiza automaticamente a visualização do índice após cada alteração da rede no nível tópicos.

Como o exemplo é reduzido e seu escopo alcança apenas um tópico, o nível tópicos contém apenas três itens tópicos, sendo que o primeiro e o terceiro itens tópicos são utilizados para delimitar o escopo do exemplo e apontam para o primeiro e ultimo itens da rede de conhecimento, respectivamente. Já o segundo item corresponde ao escopo do tópico a ser ensinado pelo sistema. Neste exemplo, o professor criou o índice com o nome dos dois primeiros itens tópicos, preencheu os dois campos com o seu nome e o título do domínio, e escreveu as referências bibliográficas consultadas para criar este domínio, como mostra a Figura 5.3, e deixou a inserção do terceiro item tópico para o Editor Gráfico.

A seguir, ele selecionou o botão Editar, da barra de botões, para chamar a tela de edição da rede de conhecimento. Esta tela é composta, além dos menus e da barra de botões, do Editor Gráfico, do Editor de Documentos e das Listas de Itens.

Assim, como os dois primeiros itens tópicos já foram criados pelo professor, o Editor Gráfico vai conter o diagrama que representa os dois itens tópicos e uma ligação entre eles. Cada item da rede de conhecimento é representado no Editor Gráfico por uma figura geométrica que o identifica. Por exemplo, os itens conceitos e exercícios são representados pela elipse, diferenciando-se entre si pela cor; os itens resultados e exemplos são representados pelo retângulo, também diferenciando-se entre si pela cor, e o item catálogo de *bugs* é representado pela união do retângulo com a elipse. Além disso, cada item no diagrama é identificado por uma letra e um número criado pelo próprio sistema ou pelo nome fornecido pelo usuário para cada item, conforme o desejo do usuário. As figuras que representam os resultados e os conceitos também podem ter o seu interior destacado quando estes forem itens tópicos.

No diagrama do exemplo, o primeiro item tópico é um conceito, isto é explicado pelo fato de uma aula introdutória no PROTEMA sempre iniciar com um conceito, e o

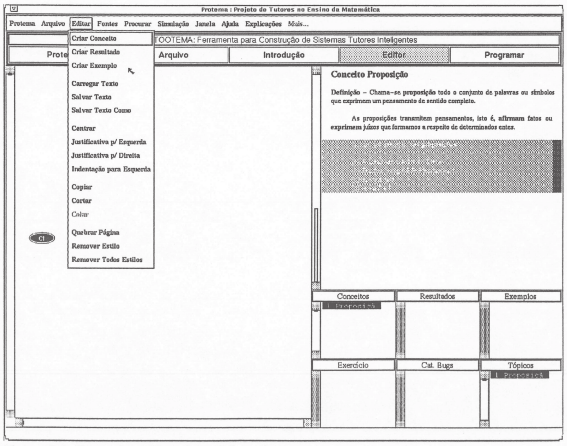


Figura 5.4: Exemplo utilizando o Editor de Documentos para criar um item.

segundo item tópico é um resultado, por este ser o objetivo do tópico a ser ensinado. A ligação entre eles foi criada automaticamente pelo sistema, para facilitar a visualização no diagrama.

Ao visualizar o diagrama o professor resolveu preencher o item conceito "Proposição". Assim, ele selecionou com o botão esquerdo do *mouse* sobre a elipse que representa este conceito na tela do Editor Gráfico e na tela do Editor de Documentos, escreveu a definição desse conceito e utilizou o Editor de Documentos para criar o exemplo desse conceito. Para isso, ele escreveu no mesmo nó o conceito e o exemplo, e depois criou a sua estrutura, marcando o bloco correspondente ao exemplo e selecionando na opção "Criar Exemplo" no menu "Editar" (Fig. 5.4). Isto faz com que o sistema pergunte o nome que será dado ao item e a sua classe, como mostra a Figura 5.5a e 5.5b.

Figura 5.5: Caixas de diálogos utilizadas pelo sistema para obter o nome e a classe de um item exemplo da rede de conhecimento.

Para criar o próximo item conceito, o professor optou por primeiro criar a sua estrutura e depois escrever a sua definição. Assim, ele selecionou com o botão esquerdo do *mouse* sobre o conceito "Proposição" no diagrama do Editor Gráfico e no menu "Gráfico" escolheu a opção "Criar item" e "Conceito". Como no item exemplo, isto faz com que o sistema pergunte pelo nome do item e a sua classe. Após responder a estas perguntas, o professor resolveu preencher este item executando os mesmos passos de quando preencheu o item conceito "Proposição". O que muda em relação ao anterior é que ao invés de criar um exemplo, o professor resolveu criar outro item conceito.

Assim que os dois itens conceitos foram criados, o professor resolve ligar o item conceito **C2** "Princípio do Terceiro Excluído" com o item resultado **R1** "Toda proposição tem um, e um só, dos valores V, F" através do menu "Gráfico", das opções "Ligar Itens" (Fig. 5.6), "CONCEITO" (Fig. 5.7a), "Princípio do Terceiro Excluído" (Fig. 5.7b) e "Toda proposição tem um, e um só, dos valores V, F" (Fig. 5.7c). Na caixa de diálogo da Figura 5.7a o sistema apresenta apenas as categorias de conceito, resultado, exemplo e exercício, pois a categoria de catálogo de *bugs* sempre será um nó folha.

Existe outra maneira, através do *mouse*, de fazer as ligações entre os itens da rede de conhecimento. Supondo que o professor resolva usá-la para ligar o item conceito **C1** "Princípio da Não Contradição" com o item resultado **R1**, mas acidentalmente ele seleciona com o botão esquerdo do *mouse* sobre a figura que representa o item resultado e arrasta

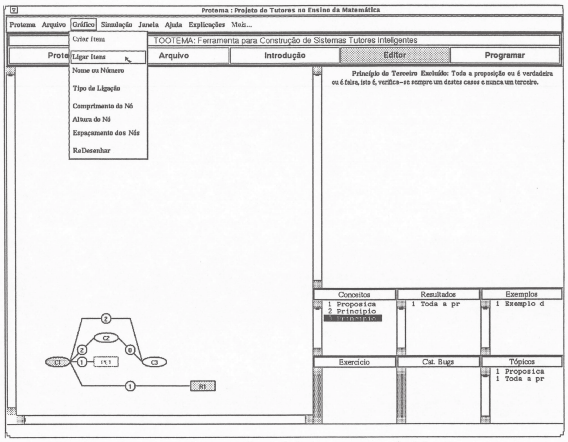


Figura 5.6: Exemplo mostrando como ligar dois itens atraves do Menu.

o *mouse* com o botão pressionado até a figura que representa o item conceito, ele recebe uma mensagem de erro (Fig. 5.8), advertindo que esta operação pode causar um ciclo na rede de conhecimento.

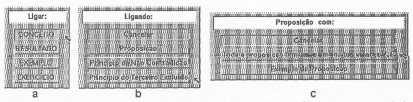


Figura 5.7: Caixas de diálogos utilizadas para fazer uma ligação entre dois itens através do Menu.

Após esta operação ser executada corretamente, o professor pode atualizar o tipo das duas ligações que criou. Mas antes, é preciso que ele defina os tipos de ligações através do menu "Gráfico" e da opção "Tipo de Ligações". Isto faz com que apareça uma caixa de diálogo mostrando as opções com os tipos de ligações, clicando em cada opção, com exceção da primeira e da segunda, é possível trocar o nome dado a cada tipo de ligação.

Feitos as alterações, o professor pode atualizar o tipo de ligação das duas ligações selecionando as com o botão esquerdo do *mouse* e em uma das opções que aparece na caixa de diálogo.

Figura 5.8: Advertência do sistema após o professor cometer um erro.

A seguir o professor resolve destruir a ligação entre o item conceito **C0** "Proposição" e o item resultado **R1**. Isto também pode ser feito de duas maneira: através da opção "Destruir Itens" no menu "Protema", ou através do *mouse*, apertando três vezes com o botão direito sobre a ligação. Independentemente da maneira como a operação é executada, no final, o sistema sempre pergunta se o professor quer realmente destruir a ligação. As mesmas operações são válidas para se destruir um nó da rede de conhecimento, com a diferença dessa operação ser mais complexa pelo fato de, além de destruir o item, é necessário destruir todas as ligações presas a ele, e de se não existir, criar as ligações entre os pais do item e os filhos do item.

E assim, depois de executar e combinar diversas operações semelhantes a estas, o professor foi capaz de criar uma rede de conhecimento semelhante à mostrada na Figura 5.9. Todas as telas podem ser ampliadas ou diminuídas deslocando-se as suas bordas. Por exemplo, se o professor não quiser aumentar ou diminuir a área do diagrama, pode-se ainda optar pela utilização das barras de rolagem que são as barras posicionadas à esquerda e abaixo da área do diagrama.

5.2 Interação com o Usuário Programador

A interface com o programador permite criar aplicativos que utilizam os objetos da biblioteca da ferramenta, tais como botões, barras de rolagem, menus, texto, entre outros, sem o conhecimento do paradigma de programação por trás do Andrew. O único requisito é algum conhecimento da linguagem de programação C.

Através do Arbcon, é possível criar códigos fontes em C dos aplicativos usando os mecanismos de orientação a objetos do Andrew. O exemplo abaixo corresponde a um possível exercício no domínio de Lógica e demonstra como um aplicativo simples pode ser criado, acrescentando-se algumas linhas de código pelo usuário (Fig. 5.10).

Ao inicializar a tela do programador, é possível acessar tanto o Arbcon como o Editor de códigos fontes em C. O Arbcon é composto pelas chaves "Barra de rolagem/Normal" e "Ligar/Novo", pelos botões "Cortar", "Copiar" e "Salvar", uma lista de aplicativos, e uma tela de mensagens. (Fig. 5.11)

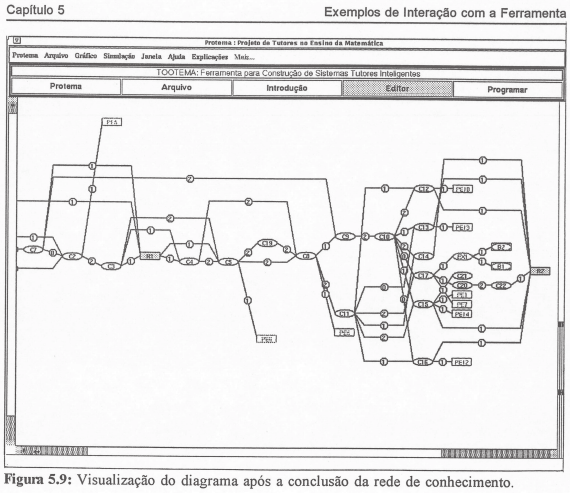


Figura 5.9: Visualização do diagrama após a conclusão da rede de conhecimento.

A chave "Barra de rolagem/Normal" é utilizada para se criar aplicativos com ou sem barra de rolagem (normal). A chave "Ligar/Novo" é utilizada para adicionar um objeto filho ou inicializar um novo aplicativo. Os botões "Cortar", "Copiar" e "Salvar" são utilizados para retirar, duplicar e salvar as configurações dos objetos no aplicativo. A lista de aplicativo lista todos os objetos disponíveis na biblioteca do TOOTEMA. E finalmente, a tela de mensagens é utilizada pelo sistema para mostrar o estado em que se encontra o sistema ou para obter informações que servem para configurar cada objeto que é selecionado.

Para ilustrar o funcionamento do Arbcon, será criado um exercício de múltiplas alternativas, composto de um objeto **clicklistV**, utilizado para apresentar a lista de alternativas e retornar uma chamada à função "xxxCallBack" quando se seleciona uma das alternativas, um **controlV**, utilizado para finalizar o exercício, e um **lset** utilizado para dividir a janela e apresentar o seu *layout* (Fig. 5.10).

Para criar este exercício, os seguintes passos são necessários:

1) Na lista de aplicativos do Arbcon, selecione o objeto **lset** e no menu Arbcon a opção "Nova Janela". Isto faz com que apareça uma janela com seu interior todo preto,

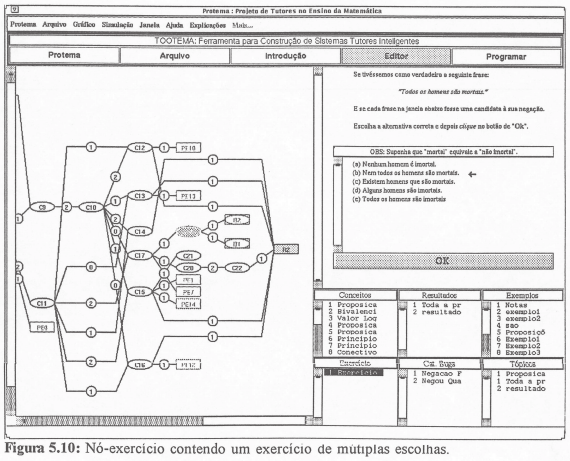


Figura 5.10: Nó-exercício contendo um exercício de mutiplas escolhas.

onde será criado o novo aplicativo.

2) O próximo passo é dividir essa janela em dois retângulos, escolhendo a opção "Dividir na Horizontal" no menu "lset" dessa janela, e apertando o botão esquerdo do *mouse* na posição desejada.

3) Após dividir a tela em dois retângulos, na lista de aplicativos escolhe-se um dos objetos que vai compor o aplicativo, na janela do aplicativo o retângulo onde será colocado o objeto, e no menu "lset" a opção "Colar". Caso o objeto escolhido necessite de alguns atributos, esses serão apresentados na tela de mensagens e deverão ser preenchidos pelo usuário para que o objeto seja corretamente incorporado ao novo aplicativo.

Neste exemplo, os objetos **clicklistV** e **controlV** ficarão no retângulo superior e inferior da janela, respectivamente, e seus atributos serão preenchidos como mostram as Figuras 5.11 e 5.12.

4) Depois de colocar esses objetos nos respectivos retângulos e salvar essa configuração em um arquivo, o usuário deve ativar a opção "Criar Código Fonte" do menu "ADEW" (Fig.5.12). Essa opção cria quatro arquivos: um arquivo de especificação com a extensão .ch, um arquivo de implementação com a extensão .c, um Imakefile e um

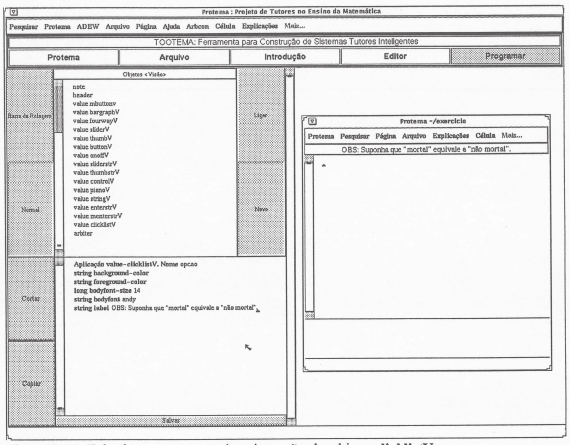


Figura 5.11: Tela de mensagem após a inserção do objeto **clicklistV**.

shellscript.

Os arquivos de especificação e implementação contém os códigos fontes do aplicativo e os arquivos Imakefile e *shellscript* são utilizados para compilar e criar o código executável do programa.

Mas antes de compilar é necessário alterar o arquivo de implementação para que o aplicativo funcione adequadamente. Assim, ao carregar o arquivo de implementação no Editor de códigos fontes, é preciso encontrar a função "OpcaoCallBack" que, como já foi dito, é chamada pelo objeto **clicklistV** toda vez que se seleciona uma das alternativas da lista. Esta função é mostrada a seguir:

```
static void OpcaoCallBack(self,val,r1,r2)
struct exercicio *self;
struct value *val;
long r1,r2;
{
if(r2 == value_OBJECTDESTROYED) {
        if(self->opcao == val) self->opcao = NULL;
}
{
/* codigo do usuario comeca aqui para OpcaoCallBack */
/* codigo do usuario termina aqui para OpcaoCallBack */
}
```

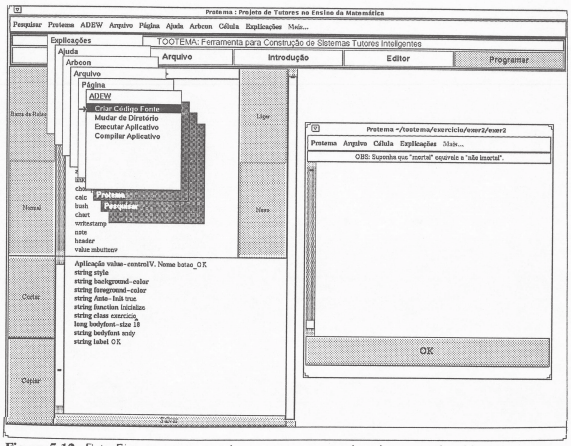



Figura 5.12: Esta Figura mostra a tela de mensagem após a inserção do objeto **controlV** e a opção que gera o código fonte.

}

Onde:

self é um ponteiro para o novo objeto criado pelo usuário;

val é um ponteiro para o objeto que foi alterado;

r1 é uma variável que indica o objeto que chamou esta função, ou seja, se neste exemplo existisse outro objeto **cliclistV**, ambos poderiam compartilhar a mesma função "OpcaoCallBack";

r2 é uma variável usada para informar que algum objeto foi destruído. Conforme pode ser observado acima, o código gerado verifica se algum objeto foi destruído e, em caso afirmativo, atribui NULL ao ponteiro do objeto destruído.

Para acessar e atribuir os valores dos objetos, é necessário conhecer os métodos de cada objeto. As descrições desses objetos e seus métodos podem ser encontradas no Tutorial. Neste exemplo são utilizados apenas dois métodos: "value_GetString(val)", que

retorna o item da lista que foi escolhido e "ColocaRespostaMEstudante(val)", que atualiza o campo de exercícios no Modelo do Estudante.

Assim, usando os dois métodos, é possível acrescentar a seguinte linha de código ao arquivo:

```
/* codigo do usuario comeca aqui para OpcaoCallBack */

    char *resultado;
    int i;

        resultado = value_GetString(self->Opcao);
        if(!resultado) return;
        /* Localizar o resultado na lista de opcoes. */
        for(i = 0; i < 5; i++)
          if(!strcmp(alternativa[i], resultado)) break;
        ColocaRespostaMEstudante(i);

/* codigo do usuario termina aqui para OpcaoCallBack */
```

Para este exemplo é preciso também definir e inicializar a lista de alternativas "char alternativa[int]", atribuir essa lista ao objeto **clicklistV** através do método "value_SetStringArrayAndSize(valueob, lista, tam)" e chamar o Modelo do Estudante no construtor do exemplo através do método "InicializeMEstudante()".

5.3 Exemplo de Simulação

Supondo que o professor queira fazer a simulação de um aluno supostamente fraco para o tópico que acabou de criar, os seguintes passos são recomendados:

1) No menu "Simulação" escolha a opção "M.Estudante" e escolha "Fraco";

2) Novamente no menu "Simulação" escolha a opção "Tópicos" e escolha o tópico "Toda proposição tem um, e um só, dos valores V, F". Isto faz com que o sistema diminua a tela do Editor Gráfico, apresente a tela do Simulador com o primeiro item a ser ordenado topologicamente, e mostre o seu conteúdo na tela do Editor de Documentos (Fig. 5.13);

3) Os próximos passos consistem em "clicar" nos últimos nós que vão sendo mostrados na tela do Simulador até aparecer a mensagem "Fim da Simulação" (Fig. 5.14 até 5.16).

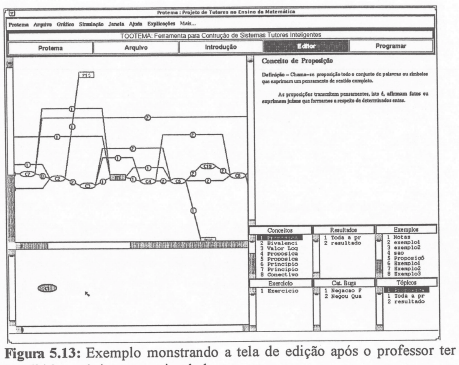


Figura 5.13: Exemplo monstrando a tela de edição após o professor ter escolhido o tópico a ser simulado.

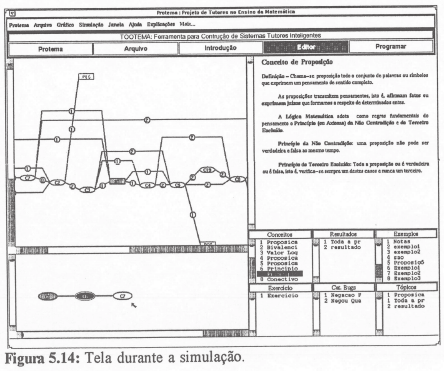


Figura 5.14: Tela durante a simulação.

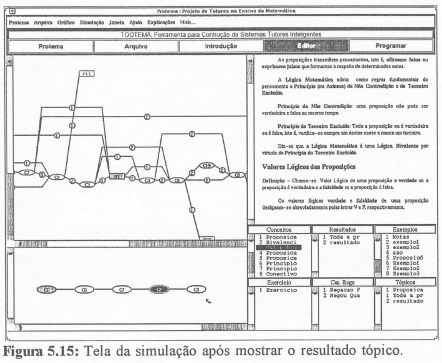


Figura 5.15: Tela da simulação após mostrar o resultado tópico.

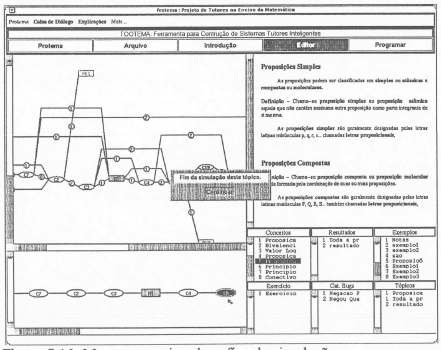


Figura 5.16: Mensagem avisando o fim da simulação.

5.4 Conclusões Parciais

Neste capítulo foi apresentado um pequeno exemplo no domínio de Lógica.

Para melhor visualizar a ferramenta TOOTEMA, foram também apresentados dois exemplos de interação, um com o usuário professor e outro com o usuário programador, que utilizam todas as janelas do protótipo da ferramenta.

No exemplo com o usuário programador, foi criado um aplicativo com poucas linhas de códigos acrescentadas pelo usuário e que pode ser utilizado no PROTEMA através do TOOTEMA.

A rede de conhecimento criada para mostrar a interação com o usuário professor também foi utilizada para demonstrar o funcionamento do Simulador.

Muito trabalho ainda deve ser feito para que este protótipo se torne uma ferramenta eficaz de auxílio no processo de desenvolvimento de materiais de ensino. Alguns destes trabalhos são propostos no capítulo seguinte.

6

Conclusões

Este capítulo conclui a dissertação, destacando as contribuições alcançadas e apresentando as conclusões finais e sugestões para a continuidade do trabalho.

6.1 Contribuições do Trabalho

TOOTEMA é um sistema que auxilia na tarefa de criar e gerenciar a base de conhecimento do ARQTEMA. Ele acompanha o professor na estruturação e gerenciamento de grande quantidade de informações, fornecendo representações gráficas, facilidades de organizar o conteúdo do material e permitindo o seqüenciamento do material a ser ensinado. Alem disso, o TOOTEMA permite que o professor analise, navegue, inspecione e manipule a estrutura do ARQTEMA, sem a necessidade de se inteirar dos detalhes da implementação. O objetivo do TOOTEMA é reduzir o tempo e recursos necessários para desenvolver o conteúdo instrucional, enquanto melhora a consistência e a qualidade das atividades de aprendizagem, deixando a cargo do professor a decisão de criar e organizar o material de ensino.

Projetar e criar instruções, especialmente para cursos extensivos e sofisticados, são tarefas complexas de gerenciamento do conhecimento. O professor deve determinar o conteúdo a ser coberto, decidir como apresentá-lo e preparar o material a ser ensinado.

Na literatura é possível encontrar diversos processos para se criar e organizar o material de ensino[HUM93][MUR92b][RUS91], os quais podem diferenciar entre si, dependendo do domínio do conhecimento. Ao analisar estes processos, foi possível definir um modelo genérico de autoria que possui as fases de:

1- **Criação e coleta do material a ser ensinado:** Envolve identificar os principais conceitos a serem ensinados e os principais tópicos a serem cobertos. Referências são identificadas e materiais preliminares do curso são criados. Exemplos de possíveis produtos desta fase podem incluir um conjunto de definições do conceito; um conjunto de exemplos; um conjunto de explicações; ilustrações; etc. Isto envolve alguma forma de classificação do material por tópicos.

2- **Contextualização da informação:** Assim que os principais tópicos do domínio tenham sido identificados, o material deve então ser contextualizado, isto é, relacionado aos outros assuntos que os estudantes tenham estudado ou irão estudar. A separação desta fase da anterior permite que o núcleo do material possa ser adaptado para diferentes tipos e níveis de estudantes.

3- **Identificação das atividades de aprendizagem:** Esta fase corresponde a projetar um conjunto de exercícios/atividades dos estudantes relacionados com os tópicos. Exemplos incluem questões a serem perguntadas e problemas a serem resolvidos. As atividades de aprendizagem podem ser organizadas por tópicos ou por tipos e podem ser ordenadas por fatores como o nível de dificuldade.

4- **Identificação das características dos estudantes:** Os indivíduos podem ter estilos de aprendizagem radicalmente diferentes. A identificação dos padrões de aprendizagem dos indivíduos pode resolver, entre outras coisas, o efeito de estilos alternativos de apresentação ou explicação.

5- **Identificação dos métodos de avaliação:** Os meios pelos quais o curso vai ser avaliado devem ser identificados. Isto inclui avaliação formal, que pode contribuir para uma classificação total do estudante, e avaliação informal, que pode ser usada para fornecer informações sobre o desempenho do estudante durante o curso. As avaliações informais podem ser realizadas ao examinar os resultados das atividades, permitindo que o professor altere a estrutura e a abordagem para ensinar um determinado tópico. As estruturas e abordagens poderão então ser adaptadas para vários tipos de estudantes.

6- **Apresentação do material:** Aqui os tópicos são organizados em um conjunto de aulas. Cada aula deverá consistir de uma ou mais atividades de aprendizagem. Estas aulas podem ser adaptadas de acordo com as características do estudante.

7- **Refinamento do projeto:** Avaliar a eficiência da base de conhecimento e modificar as porções que se mostraram ineficientes. A avaliação do sistema envolve muito mais do que simplesmente criar novas questões de teste, ou seja, além de tentar numerosas combinações possíveis de respostas dos estudantes, devemos obter informações através de entrevistas e comentários dos estudantes durante a tentativa de uso do sistema.

Em muitos casos, a simples determinação do conteúdo necessita de muitas pesquisas e análises, principalmente pelo fato do conhecimento sobre o assunto poder ser apresentado de diversas formas e ser alterado diversas vezes, o que pode torná-lo conflitante. Além disso, cada uma destas fases especificam um conjunto de tarefas a serem exercidas pelo professor e algumas destas fases podem utilizar naturalmente o computador como ferramenta de apoio.

Durante o processo de autoria da base de conhecimento, o professor tem a seu dispor todos os componentes do TOOTEMA para utilizar em todas as fases do processo. Por exemplo, o Editor de Documentos é utilizado na fase de **criar e coletar o material a ser ensinado**, e o Editor Gráfico e o *Browser* são utilizados na fase de **contextualização da informação** impondo restrições, tais como as de pré-requisitos, ordem parcial, entre outras. Para isso o ARQTEMA fornece uma rede de nós que pode ser livremente estruturada através do Editor Gráfico e *Browser*

Para a fase de **identificação das atividades de aprendizagem** e **identificação das características dos estudantes**, é utilizado o Simulador, que permite ao professor verificar se o material preparado está adequado, ou se deve ser reformulado através de nova interação com o sistema.

Na fase de **identificação dos métodos de avaliação**, o TOOTEMA permite ao professor criar diversos tipos de exercícios que serão úteis para diagnosticar o desempenho do aluno, ou seja, determinar se o processo de aprendizagem obteve sucesso.

Para ajudar na fase de **apresentação do material**, o TOOTEMA possui uma rica biblioteca de objetos utilizada pelo Montador e que permitem a construção de interfaces. Parte desta biblioteca é herdada do Andrew.

E, finalmente, a fase de **refinamento do projeto** será realizada após a conclusão da primeira versão do projeto. Uma das características do ARQTEMA é possuir duas interfaces: uma para o estudante e outra para o professor. Assim, a interação professor-sistema-estudante permite um refinamento da base de conhecimento embutida no sistema.

Além de tudo isso, o professor poderá recorrer ao Tutorial para aprender a usar o sistema e ver exemplos que mostram a realização de todas as fases.

6.2 Conclusões e Sugestões de Trabalhos Futuros

A importância de um suporte ao professor é indiscutível. Uma das mais promissoras idéias é utilizar técnicas de Inteligência Artificial, embora todo o esforço nesta área esteja sendo aplicada no suporte ao aluno.

Assim, acreditamos que é importante iniciar um processo onde o professor familiarize-se com tecnologias emergentes, seja capaz de analisar criticamente o papel destas tecnologias no processo educacional e conheça o processo de elaboração de programas educacionais no computador, para que ele possa conceber projetos que considerem as potencialidades e limitações desta tecnologia.

Portanto, a capacidade do professor para desenvolver, selecionar e explorar os programas adequados ao contexto específico é que dará a devida dimensão ao uso dos computadores na educação, não só porque facilitará as tarefas de ensino, mas, principalmente, porque poderá facilitar e ampliar a aprendizagem de seus alunos.[CAS87]

Este trabalho procurou mostrar que o suporte ao professor pode permitir aliar as técnicas de IA a uma teoria instrucional para a produção de materiais instrucionais mais efetivos. No TOOTEMA, o professor tem o suporte para participar de toda a fase de processo de autoria da base de conhecimento de um STI.

Os trabalhos de pesquisa que dão continuidade e complementam aspectos apresentados neste trabalho são vários. A seguir são listadas alguns deles:

a) Desenvolvimento de novos mecanismos que permitam alterações e ampliação das regras de produção do Módulo Tutorial, para melhor adaptar estas regras a novos domínios. Durante alguns testes com o Simulador, as regras de produção mostraram-se insuficientes para criar as aulas.

b) Desenvolvimento de componentes que permitam inserir novas informações no INTEMA ([MAL93a], [MAL93b]), e assim torná-lo mais eficiente, gerando explicações específicas de novos domínios.

c) Uma melhoria da interface da ferramenta seria a representação gráfica da rede de conhecimento separada através dos níveis e categorias, permitindo uma melhor visualização e compreensão da estrutura proposta por Michener. Por exemplo, se o autor desejasse ver a relação entre os itens da categoria de conceitos, a ferramenta mostraria apenas o diagrama que forma o grafo dessa

categoria, assim facilitaria a visualização de detalhes, bem como a detecção de possíveis falha na estruturação da rede de conhecimento.

d) Uma outra possível melhoria da ferramenta seria a modelagem do autor. A modelagem do estudante permite aos STIs monitorar o progresso e o desempenho de um estudante, analisar suas respostas e formar uma imagem de sua compreensão do conteúdo instrucional. Os sistemas aplicam então regras tutoriais para determinar a forma de auxiliar o estudante no entendimento da matéria. O mesmo método poderia ser utilizado para o modelo do autor: os sistemas monitorariam o progresso e o desempenho do professor, analisariam suas entradas, e gerariam uma imagem do professor. Os sistemas poderiam, então, aplicar técnicas instrucionais, bem como de programação para guiar o professor até a conclusão do produto.

e) Avaliação do TOOTEMA como ferramenta de autoria. Uma avaliação cuidadosa da ferramenta é necessária para que se possa estimar sua eficiência e verificar sua eficácia diante de usuários potenciais, como professores de várias disciplinas da Matemática. Neste sentido, dois projetos de Iniciação Científica estão sendo iniciados a fim de realizar esta avaliação.

f) Desenvolvimento de novas interfaces que permitam que o autor crie uma base de conhecimento em qualquer subdomínio da Matemática, sem o auxílio do programador, diminuindo assim a necessidade da utilização da interface do usuário programador.

g) Migração para outras plataformas. O **Andrew ToolKit** possui outras versões, além da versão para estação de trabalho da SUN, que rodam nas plataformas IBM, HP, PC, entre outros. A migração, por exemplo, para a plataforma PC permitirá que o sistema PROTEMA possa ser acessível a mais pessoas, pelo fato dessa plataforma ser bem difundida e de seu custo de aquisição ser menor em relação a outras plataformas.

h) Extensão para ambiente em rede seria de grande valia para o PROTEMA, pois vários alunos poderiam interagir entre si e o professor, trocando idéias ou dúvidas. O professor também poderia acessar o Modelo do Estudante de um determinado aluno enquanto este interage com o sistema e, se desejar, fazer alterações da base de conhecimento sem que o aluno perceba.

Bibliografia

Bibliografia Citada

[AIK89] AIKEN, R.M. *The Impact of Artificial Intelligence on Education: Opening New Windows.* In: MARÍK, V.; STEPANKOVÁ, O. & ZDRÁHAL, Z., ed. Lecture Nodes in Artificial Intelligence: Artificial Intelligence in Higher Education. Prague: Springer-Verlag, October, 1989, n. 451, pp. 1-13.

[AKS88] AKSCYN, R.M.; McCRACKEN, D.L.; YODER, E.A. *KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations.* Communications of the ACM, 31(7):820-835, 1988.

[AND88] ANDERSON, J.R. *The Expert Module.* In: POLSON, M.C. & RICHARDSON, J.J. eds. Foundations of Intelligent Tutoring Systems, London: Lawrence Erbaum, 1988, pp. 21-53.

[BIE91] BIELAWSKI, L. & LEWAND, R. *Intelligent Systems Design: Integrating Expert Systems, Hypermedia, and Database Technologies.* Willey Professional Computing, 1991, 302p.

[BON86] BONAR, J.; CUNNINGHAM, R.; SCHULTZ, J. *An Object-Oriented Architecture for Intelligent Tutoring Systems.* In: Proceedings OOPSLA'86, p.269-276, September, 1986.

[BOR90] BORENSTEIN, N.S. *Multimedia Applications Development with the Andrew Toolkit.* New Jersey: Prentice Hall, 1990, 310p.

[BOR91] BORENSTEIN, N.S. & THYBERG, C.A. *POWER, Ease of Use and Cooperative Work in a Practical Multimedia Message System.* Int. J. Man-Machine Studies, 34:229-259, 1991.

[BRE88] BREUKER, J. *Coaching in Help Systems.* In: SELF, J. ed. Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction, London: Chapman & Hall, 1988, pp. 310-337.

[BRO78] BROWN, J.S. & BURTON, R.B. *Diagnostic Models for Procedural Bugs in Basic Mathematical Skills.* Cognitive Science, 2:155-192, 1978.

[BRO82] BROWN, J.S.; BURTON, R.R.; DEKLEER, J. *Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I,II and III.* In: SLEEMAN, D. & BROWN, S. eds. Intelligent Tutoring Systems, New York: Academic Press, 1982, pp. 227-282.

[BUR79] BURTON, R.R. & BROWN, J.S. *An Investigation of Computer Coching for Informal Learning Activities.* Int. J. of Man-Machine Studies, 11:5-24, 1979.

[BUR88] BURNS, H. & CAPPS, C.G. *Foundations of Intelligent Tutoring Systems: An Introduction.* In: POLSON, M.C. & RICHARDSON, J.J. eds. , Foundations of Intelligent Tutoring Systems, London: Lawrence Erbaum, 1988, pp. 1-19.

[BUR92] BURGER, M.L. & DESOI, J.F. *The Cognitive Apprenticeship Analogue: A Strategy for using ITS Technology for the Delivery of Instruction and as a Research Tool for the Study of Teaching and Learning.* Int. J. of Man-Machine Studies, 36:775-795, 1992.

[CAS87] CASTRO, A.D.; CARVALHO, A.M.P.; COLOTTO, C.A.; CRUZ, E.C.; LIMA, G.C.N.; CINTRA, M.A.; PARRA, N. & BALZAN, N. C. *Didática para a Escola de 1o e 2o Graus.* PARRA, N. coord., 9ª edição, Ed. Pioneira, São Paulo, 1987, 215p.

[CLA83] CLANCEY, W.J. *GUIDON.* Journal of Computer-Based Instruction, 10(1,2):8-15, 1983.

[CLA88] CLANCEY, W.J. & BOCH, C. *Representing Control Knowledge as Abstract TASKs and Metarules.* In: BOLC, L. & COOMBS, M.J. eds. Expert System Applications, 1988, pp. 1-77.

[CON87] CONKLIN, J. *Hypertext: An Introduction and Survey.* Computer, September, 1987, pp. 17-41.

[GEN82] GENESERETH, M.R. *The Role of Plans in Intelligent Teaching Systems.* In: SLEEMAN, D. & BROWN, S. eds. Intelligent Tutoring Systems, New York: Academic Press, 1982, pp. 137-155.

[GOL82] GOLDSTEIN, I. *The Genetic Graphic: A Representation for the Evolution of Procedural Knowledge.* In: SLEEMAN, D. & BROWN, S. eds. Intelligent Tutoring Systems, New York: Academic Press, 1982, pp. 51-77.

[HAN88] HANSEN, W.J. *The Andrew Environment for Development of Educational Computing.* Comput. Educ., 12(1):231-239, 1988.

[HEG90] HEGAZI, O.M. & RADY, H.A.K. *Intelligent Tutoring Systems.* The Egyptian Computer Journal, Cairo Univ., 18(2): 93-112, 1990.

[HOL87] HOLYOAK, H.P. *Cognitive psychology.* In: SHAPIRO, S.C.; ECKROTH, D. & VALLASI, G. eds. Encyclopedia of Artificial Intelligence. Wiley & Sons - Interscience Publication, 1987, pp. 115-120.

[HUM93] Hume, T. *Isssues in the Design of An Authoring System for Educational Hypermedia Applications.* In: Proceedings *AI Tools and the Classroonm: Theory into Practice.* Seventh International PEG Conference, 1993. pp. 140-153.

[JOH87] JOHNSON, W.L. & SOLOWAY, E. *PROUST: An Automatic Debugger for Pascal Programs.* In: KEARSLEY, G. ed. Artificial Intelligence and Instruction - Applications and Methods, California: Addison-Wesley, June, 1987, pp. 49-67.

[KEA87] KEARSLEY, G. *Overview.* In: KEARSLEY, G. ed. Artificial Intelligence and Instruction - Applications and Methods, California: Addison-Wesley, June, 1987, pp. 3-10.

[KEL93] KELLY, A.E.; SLLEMAN, D.H. & GILHOOLY, K.J. *Artificial Intelligence in Education: Using State Space Search and Heuristics in Mathematics Instruction.* International Journal of Man Machine Studies, v. 38, pp. 725-746, 1993.

[KIM82] KIMBALL, R. *A Self-improving Tutor for Symbolic Integration.* In: SLEEMAN, D. & BROWN, S. eds. Intelligent Tutoring Systems, New York: Academic Press, 1982, pp. 283-307.

[MAC88] MACMILLAN, S.A.; EMME, D.; BERKOWITZ, M. *Instructional Planners: Lessons Learned.* In: PSOTKA, J.; MASSEY, L.D. & MUTTER, S.A. eds. Intelligent Tutoring Systems - Lessons Learned, New Jersey: Lawrence Erlbaum, 1988, pp. 229-256.

[MAL93] MALTEMPI, M.V. *INTEMA: Um gerador de explicações pra sistemas tutores inteligentes.* Mini-dissertação apresentada ao ICMSC-USP, São Carlos, 1993, 66p.

[MAL94] MALTEMPI, M.V. & NUNES, M.G.V. *INTEMA: An Explanation Generator for Intelligent Tutoring Systems.* In: Proceedings of the XI Brazilian Symposium on Artificial Intelligence, Fortaleza, October, 1994, pp. 457-469.

[MIC78] MICHENER, E. *Structuring Mathematical Knowledge.* In: Massachusetts Inst. of Technology - DM. Cambridge, Massachusetts, 1978, 24p.

[MIL82] MILLER, M.L. *A Structured Planning and Debugging Environment for Elementary Programming.* In: SLEEMAN, D. & BROWN, S. eds. Intelligent Tutoring Systems, New York: Academic Press, 1982, pp. 119-135.

[MOR86] MORRIS, J.H.; SATYANARAYANAN, M.; CONNER, M.H.; HOWARD, J.H.; ROSENTHAL, D.S.H.; SMITH, F.D. *ANDREW: A Distributed*

Personal Computing Environment. Communications of the ACM, 29(3):184-201, 1986.

[MUR91] MURRAY, T. & WOOLF, B.P. A *Knowledge Acquisition Tool for Intelligent Computer Tutors. SIGART Bulletin*, vol. 2, no. 2, pp. 09-21, 1991.

[MUR92a] MURRAY, T. & WOOLF, B.P. *Results of Encoding Knowledge with Tutor Construction Tools.* In: Proccedings of the Tenth National Conference on Artificial Intelligence (AAAI-92), San Jose, July, 1992, pp. 17-23.

[MUR92b] MURRAY, T. & WOOLFf, B.P. *Tools for Teacher Participation in ITS Design.* In FRASSON, C.; GAUTHIER, G.; McCALLA, G.I. eds., *Intelligent Tutoring Systems.* Second International Conference, ITS'92, Monstrela: Springer-Verlag, June, 1992, p. 593-600.

[NIC88] NICOLSON, R. *SCALD - Towards an Intelligent Authoring System.* In: SELF, J. ed. Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction, London: Chapman & Hall, 1988, pp. 236-254.

[NIL82] NILSSON, N.J. *Principles of Artificial Intelligence.* Los Altos: Springer-Verlag, 1982, 476p.

[NUN93] NUNES, M.G.V.; TURINE, M.A.S; MALTEMPI, M.V.; HASEGAWA, R. *Uso de Hipertexto/Hipermídia em Sistemas Tutores Inteligentes.* In: Notas Didaticas do ICMSC, no. 09, 1993, 32p.

[NWA90] NWANA, H.S. *Intelligent Tutoring Systems: An Overview.* Artificial Intelligence Review, 4:251-277, 1990.

[O'SH82] O'SHEA, T. *A Self-improving Quadratic Tutor.* In: SLEEMAN, D. & BROWN, S. eds. Intelligent Tutoring Systems, New York: Academic Press, 1982, pp. 304-336.

[PAR87] PARK, O.; PEREZ, R.S.; SEIDEL, R.J. *Intelligent CAI: Old Wine in New Bottles, or a New Vintage?* In: KEARSLEY, G. ed. Artificial Intelligent and Instruction Applications and Methods, New York: Academic Press, 1982, pp. 11-45.

[POL88] POLSON, M,C, & RICHARDSON, J.J. *Selected Intelligent Tutoring Systems.* In: POLSON, M,C, & RICHARDSON, J.J. eds., Foundations of ITS, Lawrence Erbaum, 1988, pp. 253-260.

[RIC83] RICH, E. *Artificial Intelligence.* Mc Graw-Hill Book Company, New York, 1983, 436p.

[RIC89] RICKEL, J.W. *Intelligent Computer-Aided Instructions: A Survey Organized Around System Components.* IEEE Transactions on Systems, Man and Cybernetics, 19:40-57, 1989.

[RUS88a] RUSSELL, D.M.; MORAN, T.P.; JORDAN, D.S. *The Instructional-Design Environmment.* In: PSOTKA, J.; MASSEY, L.D. & MUTTER, S.A. eds. Intelligent Tutoring Systems - Lesson Learned, New Jersey: Lawrence Erlbaum, 1988, pp. 203-228.

[RUS88b] RUSSELL, D.M. *IDE: The Interpreter.* In: PSOTKA, J.; MASSEY, L.D. & MUTTER, S.A. eds. Intelligent Tutoring Systems - Lesson Learned, New Jersey: Lawrence Erbaum, 1988, pp. 323-349.

[RUS91] RUSSELL, D.M.; BURTON, R.R.; JORDAN, D.S.; JENSEN, A.M.; ROGERS, R.A.; COHEN, J. *Creating Instruction with IDE: Tools for Instructional Designer.* In: YAZDAN, M. & LAWLER, R.W. eds. Artificial Intelligence and Education. vol. 2, Ablex Publishing, 1991, pp. 219-243.

[SLE87] SLEEMAN, D. *PIXIE: A Shell for Developing Intelligent Tutoring Systems.* In: LAWLER, R.W. & YAZDAN, M. eds. Artificial Intelligence and Education. vol. 1, Ablex Publishing, 1987, pp. 239-263.

[STE82] STEVENS, A.; COLLINS, A.; GOLDIN, S.E. *Misconceptions in Students' Understanding.* In: SLEEMAN, D. & BROWN, S. eds. Intelligent Tutoring Systems, New York: Academic Press, 1982, pp. 13-24.

[TAK93] TAKEHARA, R.S. *Um Modelo em Redes para um Sistema Tutor Inteligente.* Mini-dissertação apresentada ao ICMSC-USP, São Carlos, 1993, 81p.

[TAK94] TAKEHARA, R.S. ARQTEMA: *Um Modelo Genérico para Sistemas Tutores Inteligentes em Matemática.* Dissertação apresentada ao ICMSC-USP, São Carlos, 1994, 117p.

[VAS90] VASSILEVA, J. *A Classification and Synthesis of Student Modelling Techniques in Intelligent Computer-Assisted Instruction.* In: NORRIE, D.H. & SIX, H.W. eds. Lecture Notes in Computer Science, Computer Assisted Learning - 3rd International Conference, ICCAL'90, Hagen, FRG, Springer-Verlag, June, 1990, pp. 202-213.

[VIC92] VICCARI, R.M. & OLIVEIRA, F.M. *Sistemas Tutores Inteligentes.* Instituto de Informática da UFRGS, Pós-graduação em Ciência da Computação, Setembro, 1992, 64p.

[WAC88] WACHSMUTH, I. *Modeling the Knowledge Base of Mathematics Learners: Situation-Specific and Situation-Nonspecific Knowledge.* In: MANDL, H. & LESGOLD, A. eds. Learning Issues for Intelligent Tutoring Systems. New York: Springer-Verlag, 1988, pp. 63-79.

[WEN87] WENGER, E. *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge.* Los Altos: Morgan Kaufmann Publishers, January, 1987, 486p.

[WIL88] WILKINS, D.C.; CLANCEY, W.J.; BUCHANAN, B.G. *Using and Evaluating Diferential Modeling in Intelligent Tutoring and Apprentice Learning Systems.* In: PSOTKA, J.; MASSEY, L.D. & MUTTER, S.A. eds. Intelligent Tutoring Systems - Lessons Learned, New Jersey: Lawrence Erlbaum, 1988, pp. 257-277.

[WOO84] WOOLF, B. & McDONALD, D.D. *Building a Computer Tutor: Design Issues.* IEEE Transactions on Pattern Analysis and Machine Intelligence, September, 1984, p.61-72.

APÊNDICE I

Exemplos de Shells para STIs

Este apêndice fornece uma rápida descrição de algumas *shells* para STIs.

EMYCIN, HERACLES e GUIDON-WATCH [CLA83], [POL88]

GUIDON, um STI para diagnóstico médico, é um exemplo de um sistema tutor que foi construído para interagir com um sistema especialista. Ele é baseado no MYCIN, um sistema especialista baseado em regras para diagnosticar certas doenças infecciosas, tais como a meningite.

MYCIN consiste de duas partes separadas: a *shell* independente de domínio - EMYCIN - e a base de conhecimento. GUIDON foi construído para interagir com o EMYCIN e interativamente apresentar as regras da base de conhecimento para um aluno.

Pelo fato das combinações das regras de diagnósticos e os fatores médicos serem um processo de raciocínio difícil de entender, relembrar e incorporar, o MYCIN foi reconfigurado em um novo sistema chamado NEOMYCIN, no qual estratégias de diagnósticos foram separadas dos fatores médicos. HERACLES é a *shell* independente de domínio para o NEOMYCIN. Estes dois sistemas tornaram-se a base do tutor GUIDON-2. GUIDON-WATCH é a interface gráfica para todos estes sistemas, ou seja, esta interface contém mecanismos que interagem com os sistemas para ensinar, rodar consultas e editar a base de conhecimento.

SIIP [MAC88]

SIIP é um sistema que cria dinamicamente planos instrucionais, executa estes planos, replaneja e melhora o seu comportamento de planejar ao basear-se nas respostas dadas pelos alunos durante a interação. Sua estrutura é composta por uma arquitetura genérica que permite ao usuário adicionar conhecimento específico do domínio.

PIXIE [SLE87]

PIXIE é um sistema de diagnóstico *data-driven* que pode se adaptar a diferentes domínios. Ele é capaz de criar sistemas que fornecem ao aluno uma seqüência de tarefas, interpretam suas respostas, determinam a estratégia necessária para a resolução do problema e quais os *bugs* encontrados. PIXIE é capaz apenas de criar sistemas que detectam *bugs* pré-determinados pelo desenvolvedor.

IDE e IDE-interpreter [RUS88a], [RUS88b]

IDE é um sistema que torna explícito o processo de projetar e desenvolver instruções. Seu objetivo é auxiliar a criação de projetos de cursos, a estruturação dos conteúdos dos cursos e a criação das seqüências instrucionais, ou seja, ele é um ambiente interativo que lida com diversas decisões que devem ser realizadas, registradas, estruturadas e acessadas durante o projeto e desenvolvimento de instruções.

IDE-interpreter é um sistema que utiliza a arquitetura do IDE. Ao invés de utilizar uma única e rígida arquitetura para tutores inteligentes, o IDE-interpreter é um sistema mais flexível para expressar teorias instrucionais de modo explícito e empírico. Para que as representações, as regras e os termos sejam precisos e consistentes a fim de serem computados, é necessário que o autor do curso torne explícito os conceitos importantes que constituem a essência da instrução.

SCALD-1 [NIC88]

SCALD-1 é um sistema de autoria inteligente para desenvolver sistemas CAL. Sua estrutura é formada por três subsistemas interdependentes: subsistema para especificação baseado em regras, que obtém os requisitos do usuário; subsistema de projeto, que constrói um projeto adequado para CAL através das especificações do usuário; e subsistema de

implementação, que implementa automaticamente o projeto em BBC Basic. Ele utiliza o formalismo baseado em *scripts* para representar o conhecimento do domínio. Esse formalismo é formado por três tipos de dados - *scripts*, pacotes e dados primitivos. Um dado primitivo corresponde a um simples conceito no sistema CAL. O conjunto destes conceitos forma uma biblioteca de dados primitivos independente da aplicação que a esteja usando.

Um pacote é formado por dois descritores: um identificador e um tipo genérico que indica seu possível papel dentro do *script*. A diferença entre um *script* e um pacote está no fato do conteúdo do pacote ser totalmente fixo e ter a mesma implementação para diversas aplicações, o que não ocorre no *script*, pelo fato de sua instanciação depender de como seus *slots* são preenchidos.

KAFITS [MUR91]

KAFITS é uma *shell* formada por uma estrutura conceitual, que representa objetos, eventos e conexões; e uma interface para aquisição de conhecimento. O seu objetivo é facilitar a criação, alteração e teste de estratégias pedagógicas e conteúdo instrucional.

A representação do conhecimento utilizada pelo KAFITS é orientada a objetos. As principais classes de objetos são os tópicos e as apresentações. Os tópicos representam unidades de conhecimento, ou seja, informações pedagógicas tais como sumários, motivações, exemplos, entre outros. A composição de um ou mais tópicos formam uma lição.

As apresentações são interações expositórias e inquisitórias com o aluno. Elas são formadas por sugestões, congratulações, elaborações de respostas e exercícios de múltiplas escolhas durante a solução de algum problema.

A estrutura da base de conhecimento é organizada em quatros níveis de decisão: nível lição, que escolhe a meta do curso; nível tópico, que escolhe o próximo tópico a ensinar; nível apresentação, que faz a seleção e o seqüenciamento das apresentações através de analogias, exemplos, gráficos e simulações; e nível resposta, que elabora as respostas dadas pelo sistema, de acordo com as ações dadas pelo aluno. A cada iteração o mecanismo de controle percorre estes níveis para decidir a próxima meta a ser tomada pelo sistema.

O modelo do estudante utilizado pelo KAFITS é baseado no Modelo Overlay. Este modelo utiliza a estrutura em multi-níveis para obter as informações associadas a cada tópico ou habilidade ensinada.

Meno-tutor [WOO84]

Meno-tutor é uma *shell* para STI independente do domínio, ou seja, ela fornece uma estrutura genérica, na qual todas as decisões, tópicos, estratégias e regras tutoriais podem ser definidas e testadas.

O sistema é composto por um componente tutor e um gerador de linguagem. O componente tutor contém a base de conhecimento e o mecanismo de raciocínio, que planeja e organiza o texto a ser apresentado; e o gerador de linguagem, que produz expressões sintaticamente corretas a partir dos conceitos especificados pelo componente tutor.

A base de conhecimento é um conjunto de unidades de decisão divididas em três níveis: o nível pedagógico, que investiga e diagnostica o estado de um aluno; o nível estratégico, que contém as estratégias de ensino; e o nível tático, que tem por finalidade executar a estratégia definida no nível estratégico. Assim, estes três níveis formam uma rede de conhecimento, que estrutura as decisões e define as transições entre eles.

Bite-Sized Tutor [BON85]

Este sistema de autoria para STI, desenvolvido na Universidade de Pittsburgh, é baseado em uma arquitetura hierárquica de três camadas. A camada no nível mais baixo ou camada de conhecimento, contém conhecimentos declarativos e procedimentais contidos em uma rede de conceitos e conectados por predicados.

Acima da camada de conhecimento existe a camada de curriculum, que contém o conhecimento sobre o seqüenciamento de curriculum em termos de conhecimentos pré-requisitos.

Finalmente, a camada de aptidão, ou camada meta-cognitiva, é responsável pela individualização da instrução para diferentes alunos.

A arquitetura do Bite-Sized Tutor é baseada em uma linguagem de programação orientada a objetos.

CMU-Tutor [HAN88]

O CMU-Tutor é um ambiente desenvolvido no **Andrew** para criar lições. O ambiente do CMU-Tutor inclui um Editor de Lições, um Executor de Lições e uma Base de Lições.

Cada lição é dividida em unidades, que descrevem um único passo na seqüência de imagens que o aluno irá ver. As unidades são sub-rotinas, que limpam a tela para o seu início, mostram alguma imagem ou animação, ou perguntam por uma resposta do aluno.

Para demonstrar o funcionamento do CMU-Tutor, foram desenvolvidas lições que ensinam a escrever equações de movimentos periódicos, denominada de Waves. Após cada animação, Waves pergunta para o usuário descrever qual o tipo de movimento descrito pela animação. A resposta do usuário é comparada pelo ambiente, com diferentes possibilidades previstas pelo autor.

APÊNDICE II

Classes de Objetos do TOOTEMA

Este apêndice fornece uma rápida descrição de como é a especificação das classes no **Andrew**, um diagrama ilustrando a hierarquia das principais classes utilizadas pelo TOOTEMA e as respectivas listagens das especificações.

Em qualquer definição de classes no **Andrew**, existem duas partes, a especificação e a implementação, que devem ser fornecidas em arquivos separados.

Na especificação é possível definir uma nova classe de objeto ao dar seu nome, sua superclasse (se possuir) e um conjunto de atributos da classe, em um formato que se assemelha bastante com a definição de estrutura na linguagem C.

Estes atributos definem a forma como a nova classe difere de sua superclasse. Tal diferença consiste de:

a) **novos métodos** não encontrados na superclasse;

b) **sobreposições** (*overrides*), que especificam que certos métodos da superclasse vão ser substituídos por novas implementações;

c) **procedimentos**, que não operam sobre nenhum objeto específico, e assim em geral não precisam que tais objetos sejam passados como parâmetros;

d) **macrométodos**, que são semelhantes aos métodos exceto que ao invés de olhar na tabela de métodos e chamar alguma função, eles são definidos como as macros em C, que são expandidas em tempo de compilação e assim nunca envolvem a sobrecarga da chamada de funções; e

e) **dados**, que são usados para diferenciar as instâncias de uma classe.

Para cada especificação existe uma correspondente implementação, que deve conter

os procedimentos definidos para cada método ou sobreposição especificado.

No **Andrew**, quando possível, uma classe de objetos é dividida em duas classes distintas: *dataobject* e *view*. A classe *dataobject* contém todas informações a respeito do estado em que se encontra o objeto, enquanto a classe *view* contém informações a respeito da maneira que o objeto é mostrado na tela. Por convenção, quando a classe *view* vier acompanhada da classe *dataobject*, a classe *view* terá o mesmo nome da classe *dataobject*, mas acrescentado de um "view" ou "v", como por exemplo, tootema e tootemav.

Na Figura II.1 é apresentada a hierarquia das principais classes de objetos utilizadas pelo TOOTEMA.

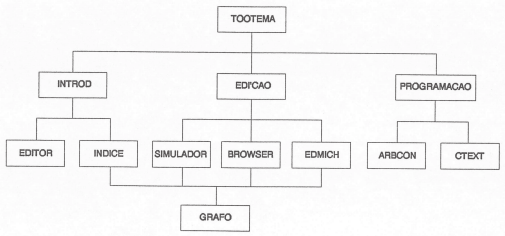


Figura II.1: Diagrama hierárquico das principais classes utilizadas pelo TOOTEMA.

A seguir são mostrados apenas os códigos fontes das especificações das principais classes utilizada pelo TOOTEMA.

As classes tootema e tootemav são responsáveis pelo controle das telas apresentadas pelo TOOTEMA.

```
class tootema : apt {
  classprocedures:
      InitializeObject( struct tootema *self ) returns boolean;
      FinalizeObject( struct tootema *self ) returns void;
```



```
      Create(FILE *fp) returns struct tootema *;
  methods:
      InicializaDominio(FILE *fp) returns long;
      SetArquivo(char *name);
  overrides:
      Read(FILE *file,long id) returns long;
      Write(FILE *file,long id,long level) returns long;
      ViewName() returns char *;
  macromethods:
      Grafo() (self->grafo)
      Dual() (self->dual)
      Introd() (self->introd)
      Browser() (self->browser)
      Autor() (self->autor)
      Titulo() (self->titulo)
      Resumo() (self->resumo)
      ApresentarIndice()
          (htext_ApresentarIndice(introd_Indice(tootema_Introd(self))))
      GetArquivo() (self->filename)
      GetArquivoIndefinido() (self->arquivoindefinido)
  data:
      struct grafo  *grafo;  /* Principal objeto manipulado pelo
tootema. */
      struct dual   *dual;   /* Os proximos 3 objetos sao utilizados
pelas*/
      struct introd *introd; /* suas respectivas telas. */
      struct browser     *browser;
      char               filename[1000];
      int           arquivoindefinido;
};

class tootemav : aptv {
  classprocedures:
      InitializeClass() returns boolean;
      InitializeObject( struct tootemav *self ) returns boolean;
      FinalizeObject( struct tootemav *self ) returns void;
  overrides:
      FullUpdate( enum view_UpdateType Type, long left, long top, long
          width, long height ) returns void;
      Hit( enum view_MouseAction action, long x, long y, long numClicks
          ) returns struct view *;
      PostMenus( struct menulist *menulist ) returns void;
      PostKeyState( struct keystate *kstate ) returns void;
      SetDataObject( struct tootema *tootema ) returns void;
      GetApplicationLayer() returns struct view *;
  methods:
      NovaJanela(struct im *im, struct view *janela) returns boolean;
      SetFrame(struct frame *frame);
      SetParent(struct cntelv *parent);
      SetTituloPainel(char *titulo);
  macromethods:
      GetFrame() (self->frame)
      GetParent() (self->parent)
  data:
      struct tootema     *tootema;
      struct grafov      *grafov;
      struct introdv     *introdv;
      struct dualv       *dualv;
      struct browserv    *browserv;
      struct suite       *controle;
      struct scroll      *scroll;
      struct lpair       *lp;
      struct im          *ajuda;
      int                objeto;
```

```
        struct keystate     *keyState;
        struct keymap       *kmap;
        struct cursor       *cursor;
        struct menulist     *menu;
        boolean             ultimo_nivel;
        struct frame        *frame;
        struct cntelv       *parent;
};
```

As classes introd e introdv são responsáveis pela interação inicial com o professor. Através desta classe, o professor pode fornecer alguns dados iniciais à ferramenta, tais como: seu nome, título do domínio, resumo sobre o domínio, referências bibliográficas, opcionalmente os tópicos que formam o domínio, entre outros.

```
class introd : dataobject [dataobj] {
  classprocedures:
        InitializeObject(struct introd *self) returns boolean;
        FinalizeObject(struct introd *self) returns void;
        Create(struct grafo *grafo) returns struct introd *;
  overrides:
        Read(FILE *file,long id) returns long;
        Write(FILE *file,long id,long level) returns long;
        ViewName() returns char *;
  macromethods:
        Grafo() (self->grafo)
        Autor() (self->autor)
        Titulo() (self->titulo)
        Resumo() (self->resumo)
        Indice() (self->indice)
  data:
        struct grafo  *grafo;
        char          autor[150]; /* Contem o nome da pessoa que esta
utilizando a ferramenta. */
        char          titulo[80]; /* Contem o titulo dado ao dominio. */
        struct text   *resumo;    /* Resumo sobre o dominio. */
        struct indice *indice;    /* Contem o indice do dominio. */
};

class introdv : view {
  classprocedures:
        InitializeObject(struct introdv *self) returns boolean;
        FinalizeObject(struct introdv *self) returns void;
  overrides:
        FullUpdate(enum view_UpdateType tipo, long esq, long top, long
            comp, long larg);
        Hit(enum view_MouseAction acao, long x, long y, long clicks)
            returns struct view *;
        SetDataObject(struct introd *introd);
  macromethods:
        GetRespNome() (self->nome)
        GetRespTitulo() (self->titulo)
        GetText() (self->txt)
        GetTextview() (self->txtv)
        GetTextEd() (self->txted)
        GetTextvEd() (self->txtved)
        GetTextResumo() (self->resumo)
        GetIndiceview() (self->indicev)
  data:
        struct introd *introd;
        struct respv  *nome;
        struct respv  *titulo;
        struct editorv     *edit;
        struct indicev     *indicev;
```



```
        struct lpair  *lp;
        struct lpair  *lp1;
        struct lpair  *lp2;
        struct lpair  *lp3;
        struct lpair  *lp4;
        struct lpair  *lp5;
        struct view   *view1;
        struct view   *view2;
        struct text   *resumo;
        struct logotipo    *logotipo;
        struct scroll *scroll;
};
```

As classes indice e indicev podem ser utilizadas para criar uma estrutura inicial do domínio, ou seja, é uma opção para montar o nível tópicos da rede de conhecimento.

```
class indice: text {
  overrides:
        Clear();
        Read(FILE *file, long id) returns long;
        Write(FILE *file, long writeID, int level) returns long;
  methods:
        ApresentarIndice() returns boolean;
        Indent(struct mark *range) returns long;
        ReindentLine(long pos);
        RecommentBackward(long pos);
        ReverseBalance(long pos) returns long;
        RedoStyles();
        TabAndOptimizeWS(long pos, int inc) returns long;
  macromethods:
        GetGrafo() (self->grafo)
  data:
        struct style *italic_style, *topic_style, *michener_style;
        struct grafo *grafo;
};

class indicev: textview[textv] {
  overrides:
        SetDataObject(struct indice *indice);
        PostKeyState(struct keystate *keystate);
        PostMenus(struct menulist *menulist);
  classprocedures:
        InitializeClass() returns boolean;
        InitializeObject(struct indicev *self) returns boolean;
        FinalizeObject(struct htextv *self);
        SetBounceTime(long time) returns long;
  data:
        struct keystate    *state;
        struct menulist    *menus;
        struct indice      *indice;
};
```

A classe edicaov é responsável pelo controle do Editor de Documentos (classe edmich), Editor Gráfico (classe grafov) e o Browser (classe browser).

```
class edicaov : view {
  classprocedures:
        InitializeObject(struct edicaov *self) returns boolean;
        InitializeClass(struct edicaov *self) returns boolean;
        FinalizeObject(struct edicaov *self) returns void;
  overrides:
        PostMenus(struct menulist *menuList);
```



```
        FullUpdate(enum view_UpdateType tipo, long esq, long top, long
            comp, long larg) returns void;
        Hit(enum view_MouseAction acao, long x, long y, long numClicks)
            returns struct view *;
        SetDataObject(struct edicao *edicao) returns void;
  methods:
        GetGrafoFile(char *filename) returns int;
  macromethods:
        Browser() (self->browser)
        ArvoreV() (self->treev)
        GrafoV() (self->grafov)
        Editor() (self->edview)
        UltimoItemApresentado() (self->ultimoitem)
  data:
        struct browser     *browser;
        struct nos         *no_atual;
        struct nos         *no_inicial;
        struct arvorev     *treev;
        struct grafov      *grafov;
        struct scroll      *scroll;
        struct scroll      *scroll1;
        struct suite       *lista_conceito;
        struct suite       *lista_resultado;
        struct suite       *lista_exemplo;
        struct suite       *lista_topicos;
        struct suite       *lista_exercicio;
        struct suite       *lista_catalogo;
        struct lpair       *lp;
        struct lpair       *lp0;
        struct lpair       *lp1;
        struct lpair       *lp2;
        struct lpair       *lp3;
        struct lpair       *lp4;
        struct lpair       *lp5;
        struct lpair       *lp6;
        struct text        *text;
        struct editmich    *edview;
        struct keystate    *state;
        struct menulist    *menus;
        struct simular     *simula;
        long               flags;
        char               filename[100];
        struct rede        *ultimoitem;
};
```

A classe browser contém todos os mecanismos de buscas utilizados pelo *Browser*.

```
class browser : dataobject [dataobj] {
  classprocedures:
        InitializeObject( struct browser *self ) returns boolean;
        FinalizeObject( struct browser *self ) returns void;
        Create(struct grafo *grafo) returns struct browser *;
  methods:
        InitTree(struct rede *domain) returns void;
        ProximoItem(struct nos *no, struct rede *item) returns struct
nos*;
        DestroiFilho(struct nos no) returns void;
  overrides:
        Read(FILE *file,long id) returns long;
        Write(FILE *file,long id,long level) returns long;
        ViewName() returns char *;
  macromethods:
        Arvore()       (self->tree)
        ArvoreRaiz()   (arvore_Raiz(self->tree))
        NoItem(tn)     (tn->item)
```



```
        NoNome(tn)     (tn->item->ident)
        Pai(tn)   (arvore_No_Pai(self->tree,tn))
        ArvoreNumeroNos()  (arvore_Numero_Nos(self->tree))
        Grafo()        (self->grafo)
        GrafoRaiz()          (grafo_Grafo(self->grafo))
        CriaItem(n,ca,cl,t)      (grafo_CriaItem(self->grafo,n,ca,cl, t))
        ListaItens()         (grafo_ListaItens(self->grafo))
        ListaItemNumero(n)       (grafo_ListaItemNumero(self->grafo,n))
        NumeroItens()            (grafo_NumeroItens(self->grafo))
        ItemIdentificador(tn)    (grafo_ItemIdentificador(self->grafo,tn))
        ItemNome(tn)         (grafo_ItemNome(self->grafo,tn))
        ItemTipo(tn)         (grafo_ItemTipo(self->grafo,tn))
        ItemClasse(tn)           (grafo_ItemClasse(self->grafo,tn))
        ItemClassficacao(tn)
(grafo_ItemClassificacao(self->grafo,tn))
        ItemTexto(tn)            (grafo_ItemTexto(self->grafo,tn))
        ItemComprimentoTexto(tn)
(grafo_ItemComprimentoTexto(self->grafo,tn))
        ItemPosicaoTexto(tn)       (grafo_ItemPosicaoTexto(self->grafo,tn))
        ItemPredecessor(tn,n)    (grafo_ItemPredecessor(self->grafo,tn,n))
        itemSucessor(tn,n)       (grafo_ItemSucessor(self->grafo,tn,n))
        ItemNumeroPre(tn)
(grafo_ItemNumeroPredecessores(self->grafo,tn))
        ItemNumeroSuc(tn)  (grafo_ItemNumeroSucessores(self->grafo,tn))
        ItemDualPre(tn,n)  (grafo_ItemDualPre(self->grafo,tn,n))
        ItemDualSuc(tn,n)  (grafo_ItemDualSuc(self->grafo,tn,n))
        ItemNumeroDualPre(tn)    (grafo_ItemNumeroDualPre(self->grafo,tn))
        ItemNumeroDualSuc(tn)    (grafo_ItemNumeroDualSuc(self->grafo,tn))
        ItemPosicaoLista(tn)     (grafo_ItemPosicaLista(self->grafo,tn))
        ListaTopicos()           (grafo_ListaTopicos(self->grafo))
        NumeroTopicos()          (grafo_NumeroTopicos(self->grafo))
        TopicoNumero(n)          (grafo_TopicoNumero(self->grafo,n))
        TopicoItem(tn)           (grafo_TopicoItem(self->grafo,tn))
        TopicoIdentificador(tn)
(grafo_TopicoIdentificador(self->grafo,tn))
        TopicoNome(tn)           (grafo_TopicoNome(self->grafo,tn))
        TopicoTipo(tn)           (grafo_TopicoTipo(self->grafo,tn))
        Simular()          (self->simula)
  data:
        struct arvore *tree;
        struct grafo  *grafo;
        struct simular  *simula;
};
```

A classe grafo contém toda a estrutura utilizada para representar o domínio e os procedimentos utilizados para manipulá-la. Essa estrutura contém o nível tópico e as cincos categorias do conhecimento (conceitos, resultados, exemplos, exercícios e catálogo de *bugs*).

```
class grafo: dataobject [dataobj] {
  classprocedures:
        InitializeObject(struct grafo *self) returns boolean;
        FinalizeObject(struct grafo *self);
        RealocaItem(struct rede **p, int n) returns struct rede **;
        RealocaLigacao(struct ligacao **p, int n) returns struct ligacao
**;
        Imprime(struct grafo *self);
        Atualiza(struct rede *item, int topico);
        Ligacao(struct rede *item1, struct rede *item2, long tipo) returns
            boolean;
        LigacaoDual(struct rede *item1, struct rede *item2, long tipo)
            returns boolean;
        DestroiLigacao(struct rede *item1, struct rede *item2) returns
```

```
        boolean;
        DestroiLigacaoDual(struct rede *item1, struct rede *item2) returns
            boolean;
        ExisteLigacao(struct rede *item1, struct rede *item2) returns
        boolean;
    overrides:
        Read(FILE *file, long id) returns long;
        Write(FILE *file, long writeID, int level) returns long;
    methods:
        Clear();
        AtualizaGrafo(struct rede **topicos);
        CriaItem(char *nome, long tipo, long classe, int classif) returns
            struct rede*;
        PosicaoItemNome(char *nome) returns int;
        RecuperaItemTipo(long tipo, int ident) returns struct rede*;
        DestroiItemNome(char *nome) returns boolean;
        DestroiItem(struct rede *item) returns boolean;
        LigacaoNome(char *nome1, char *nome2, long tipo) returns boolean;
        DestroiLigacaoNome(char *nome1, char *nome2) returns boolean;
        PercorreLargura();
        EItemFilhoNome(char *nome1, char *nome2) returns boolean;
        EItemFilho(struct rede *t1, struct rede *t2) returns boolean;
        ExisteLigacaoNome(char *nome1, char *nome2) returns boolean;
    macromethods:
        Grafo() (self->grafo)
        ListaItens() (self->lista)
        ListaItemNumero(n) (self->lista[n])
        NumeroItens() (self->numero_itens)
        ItemIdentificador(tn) (tn->Identificador)
        ItemNome(tn) (tn->nome)
        ItemTipo(tn) (tn->tipo)
        ItemClasse(tn) (tn->classe)
        ItemClassficacao(tn) (tn->classif)
        ItemTexto(tn) (tn->text)
        ItemComprimentoTexto(tn) (tn->comprimentxt)
        ItemPosicaoTexto(tn) (tn->posicaotxt)
        ItemPredecessor(tn,n) (tn->pre[n])
        ItemSucessor(tn,n) (tn->suc[n])
        ItemNumeroPredecessores(tn) (tn->num_pre)
        ItemNumeroSucessores(tn) (tn->num_suc)
        ItemDuaisPre(tn) (tn->dual_pre)
        ItemDuaisSuc(tn) (tn->dual_suc)
        ItemDualPre(tn,n) (tn->dual_pre[n])
        ItemDualSuc(tn,n) (tn->dual_suc[n])
        ItemNumeroDualPre(tn) (tn->num_dual_pre)
        ItemNumeroDualSuc(tn) (tn->num_dual_suc)
        ItemPosicaoLista(tn) (tn->pos)
        ItemCoordenadaX(tn) (tn->x)
        ItemCoordenadaY(tn) (tn->y)
        ItemLigacoes(tn,n) (tn->link[n])
        ItemNumeroLigacoes(tn) (tn->num_suc + tn->num_dual_suc)
        ItemFolha(tn) ((tn->num_suc + tn->num_dual_suc <= 0) ? 1 : 0)
        ListaTopicos() (self->topicos)
        NumeroTopicos() (self->numero_topicos)
        TopicoNumero(n) (self->topicos[n])
        TopicoIdentificador(tn) (tn->identificador)
        TopicoNome(tn) (tn->nome)
        TopicoTipo(tn) (tn->tipo)
        TipoLigacao(tn) (self->tipoligacao[tn])
    data:
        struct rede   *grafo;   /* Aponta para o primeiro item no grafo.*/
        struct rede   **lista;  /* Lista que contem todos os itens do
            grafo.*/
        int           numero_itens; /* Contem o total de itens no
grafo.*/
```



```
        struct rede   **topicos;    /* Lista que contem todos os topicos
            do grafo.*/
        int       numero_topicos;   /* Contem o total de topicos no
        grafo. */
        int identc, identr, idente; /* Contem o numero de itens de cada
        tipo */
        char tipoligacao[LIGACOES][150]; /* Contem os nomes dados aos
        tipos */
                /* de ligacoes. */
};
```

A classe grafov representa o Editor Gráfico e contém os procedimentos responsáveis pela interação visual da estrutura do domínio com o usuário.

```
class grafov: estrutv {
  classprocedures:
        InitializeClass() returns boolean;
        InitializeObject(struct grafov *self) returns boolean;
        FinalizeObject(struct grafov *self);
  overrides:
        CoordenadaX(long desl, long width, long esp);
        CoordenadaY(long desl, long height, long esp);
        SetDataObject( struct grafo *grafo );
        WantUpdate(struct view *req);
        Update();
        FullUpdate(enum view_UpdateType type, long x, long y, long w, long
h);
        Hit(enum view_MouseAction action, long x, long y, long numClicks)
            returns struct view *;
  methods:
        DeterminaNivel();
        DeterminaEscopo(int no) returns int;
        DesenhaArco(int arco);
        DestacaArco(int arco);
        SetHitFunction(procedure HitFunction, long rock);
  macromethods:
        GetNumberArcos() (self->total_arcos)
        GetHitFunction() self->HitFunction)
        GetRock() (self->rock1)
        GetArcoPai(i) (self->list_arco[i]->pai)
        GetArcoFilho(i) (self->list_arco[i]->filho)
        GetArcoTopico(i) (self->list_arco[i]->topico)
        GetArcoTipo(i) (self->list_arco[i]->tipo)
  data:
        struct grafo *grafo;
        int total_arcos;
        struct menulist *menu;
        int no_atual, arco_atual;
        int objeto;
        int (*HitFunction) ();
        long rock1;
        struct CoordArco *list_arco[MAX_ARCO];
        int nivel_no[MAX_NO];
        int escopo[MAX_NO];
        float vetor_y[MAX_NO];
        long nodex;
        long nodey;
        long minimo;
        long maximo;
};
```

A classe estruv é a superclasse de grafov e arvorev, e contém os procedimentos responsáveis pela criação do desenho do diagrama.


```
class estrutv: view {
  classprocedures:
        InitializeClass() returns boolean;
        InitializeObject(struct estrutv *self) returns boolean;
        FinalizeObject(struct estrutv *self);
  overrides:
        GetInterface(char *interface) returns char*;
        FullUpdate(enum view_UpdateType type, long x, long y, long w, long
h);
  methods:
        CoordenadaX(long desl, long width, long esp);
        CoordenadaY(long desl, long height, long esp);
        DesenhaNo(int x, int y, char *nome, long tipo);
        DestacaNo(int x, int y, char *nome, long tipo);
        SetDadoX(long min, long max, long desl);
        SetDadoY(long min, long max, long desl);
  macromethods:
        GetAlturaTotal() (self->maxY-self->minY)
        GetComprimentoTotal() (self->maxX-self->minX)
        GetMinX() (self->minX)
        GetMinY() (self->minY)
        GetMaxX() (self->maxX)
        GetMaxY() (self->maxY)
        GetComp() (self->comp)
        GetAlt() (self->alt)
        GetEspX() (self->espX)
        GetEspY() (self->espY)
        GetDesX() (self->desX)
        GetDesY() (self->desY)
  data:
        struct menulist *menu;
        long minY, maxY;
        long minX, maxX;
        long espX, espY;
        long desX, desY;
        long alt;
        long comp;
};
```

A classe editmich representa o Editor de Documentos e contém os procedimentos utilizados para editar e alterar o conteúdo dos itens que formam a rede de conhecimento, bem como, os utilizados para estruturar a rede de conhecimento.

```
class editmich : editorv {
  overrides:
        PostMenus(struct menulist *menuList);
        PostKeyState(struct keystate *keyState);
        SetDataObject(struct grafo *grafo) returns void;
        WantUpdate(struct view *req);
  methods:
        MenuMichener(boolean tn);
        SetRocks(long *r1, long *r2, long *r3, long *r4, long *r5);
        SetFunction(procedure listfunction);
        MontaNoCaminho(long nivel, struct rede *vertice);
        CriaNo(long type) returns struct rede*;
  classprocedures:
        Create(struct grafo *grafo) returns struct editmich*;
        InitializeObject(struct editmich *self) returns boolean;
        InitializeClass(struct editmich *self) returns boolean;
        FinalizeObject(struct editmich *self);
  macromethods:
        CopiaCaminho(caminho) (strcpy(self->caminho,caminho))
        NivelNo() (self->nivel)
        NoAtual() (self->no_atual)
```



```
  data:
        struct keystate *state;
        struct menulist *menus;
        long flags;             /* flags do menu */
        struct grafo *grafo;    /* data no formato grafo */
        char caminho[350];      /* caminho no grafo dos nos percorrido */
        int (*listfunction) ();
        long rock1, rock2, rock3, rock4, rock5;
        struct rede *listno[50];
        struct rede *no_atual;
        long nivel;
};
```

A classe editorv é um editor mutimídia utilizado pela classe introdv e como superclasse de edimich.

```
class editorv : view {
  overrides:
        PostMenus(struct menulist *menuList);
        PostKeyState(struct keystate *keyState);
        FullUpdate(enum view_UpdateType type, long l, long t, long w, long
r);
        WantUpdate(struct view *req);
        Update();
        LinkTree(struct view *parent);
        Hit (enum view_MouseAction action, long x, long y, long Clicks)
            returns struct view *;
  methods:
        GetEditOn(char *aname, long isnew) returns int;
        AtualizaComprimentoTexto();
        TextoOriginal();
  classprocedures:
        InitializeObject(struct editorv *self) returns boolean;
        InitializeClass(struct editorv *self) returns boolean;
        FinalizeObject(struct editorv *self);
  macromethods:
        ApagaCaracteres(p, l) \
            (text_AlwaysDeleteCharacters((struct text*)self->data, p, l))
        CopiaTexto(p, pr, pl, l) \
            (text_AlwaysCopyText((struct text*)self->data, p, pr, pl, l))
        NotificaObservadores(tn) \
            (text_NotifyObservers((struct text*)self->data, tn))
        ComprimentoTexto() \
            (text_GetLength((struct text*)self->data))
        PegaPosicaoBloco() \
            (textview_GetDotPosition((struct textview*)self->view))
        PegaComprimentoBloco() \
            (textview_GetDotLength((struct textview*)self->view))
        CopiaNomeArquivo(nome) \
            (strcpy(self->name, nome))
        PosicaoBloco(tn) \
            (textview_SetDotPosition((struct textview*)self->view, tn))
        ComprimentoBloco(tn) \
            (textview_SetDotLength((struct textview*)self->view, tn))
        TrocaTexto(tn) \
            (textview_SetDataObject((struct textview*)self->view,tn))
        PegaComprimentoTexto() (self->comprimento)
        EditarTexto() (self->editar)
        Texto() ((struct text*)self->data)
        TextoV() ((struct textview*)self->view)
  data:
        struct keystate *state;
        struct menulist *menus;
```

```
        char name[100];             /* nome do arquivo*/
        long flags;                 /* flags do menu */
        struct view *view;          /* view do arquivo */
        struct scroll *scroll;      /* scrollbar do view */
        struct dataobject *data;    /* data (text) do arquivo */
        struct dataobject *aux;
        int (*listfunction) ();
        long rock1, rock2, rock3, rock4, rock5;
        struct scroll *app;
        long comprimento;
        boolean editar;
        struct cursor *waitCursor;
};
```

A classe simular representa o Simulador e contém os procedimentos utilizados para simular uma possível aula do domínio.

```
class simular : dataobject [dataobj] {
  classprocedures:
        Create(struct grafo *grafo) returns struct simular*;
        InitializeObject( struct simular *self ) returns boolean;
        FinalizeObject( struct simular *self ) returns void;
  methods:
        InsereCSV(struct rede *item) returns boolean;
        InsereCVA(int topico, struct rede *item) returns boolean;
        InsereCV(int topico, struct rede *item) returns boolean;
        RetirarUltimoItemCSV() returns struct rede*;
        ME_PERTENCE_CSV(struct rede *item) returns boolean;
        InSereNivelEstudante(int nivel);
        ME_PERTENCE_CVA(struct rede *item) returns boolean;
        ME_PERTENCE_CV(struct rede *item) returns boolean;
        INTERFACE_MOSTRA(struct rede *item);
        TransfereCVparaCVA();
        LimpaMEstudante();
        LimpaCVA();
        LimpaCV();
        LimpaCSV();
        PreparaInterface(procedure Funcao, int paramentro);
        RegrasTutoriais() returns long;
  macromethods:
        Aula() (self->aula)
        Grafo() (self->grafo)
        ModeloEstudante() (self->estudante)
        ME_NIVEL_ESTUDANTE() (self->estudante->nivel_estudante)
        CSV() (self->estudante->CSV)
        CSVNumero(n) (self->estudante->CSV[n])
        TotaldeCSV() (self->estudante->total_CSV)
        CVA() (self->estudante->CVA)
        CVisto() (self->estudante->CV)
        TopicoAtual() (self->estudante->CV->topico)
        Contexto() (praula_Contexto(self->aula))
        Topico() (praula_Topico(self->aula))
        Apresentar() (self->ApresentaConteudo)
        MostraConteudo(item) (self->ApresentaConteudo(self->parametro1,
item))
  data:
        struct grafo  *grafo;
        struct praula   *aula;
        struct m_estudante *estudante;
        int (*ApresentaConteudo) ();
        long parametro1;
        boolean loop;
};
```


A classe praula representa o Módulo Tutorial do ARQTEMA, contém os procedimentos de preparação de aula e as regras tutoriais.

```
class praula: dataobject [dataobj] {
  classprocedures:
        InitializeObject( struct praula *self ) returns boolean;
        FinalizeObject( struct praula *self ) returns void;
  methods:
        PreparaAula(int topico, struct rede *elemento, struct pilha
*pilha);
        RegrasTutoriais();
        DeterminaElemento(int topico) returns struct rede*;
        ColocaNaListaCDE(struct ligacao **ldual);
        ColocaNaListaCDD(struct ligacao **ldual);
        PrimeiroDaListaCDE() returns struct rede*;
        PrimeiroDaListaCDD() returns struct rede*;
        Simular(int true);
  macromethods:
        Aula() (self->aula)
        AulaConceito() (self->aula_conceito)
        Topico() (self->topico)
        Contexto() (self->contexto)
        Grafo() (self->grafo)
        Simulador() (self->simula)
  data:
        struct pilha *aula;
        struct pilha *aula_conceito;
        int topico;
        struct rede *elemento, *elem_rel, *elem_dual;
        long contexto;
        struct grafo *grafo;
        struct ConceitoDual *CDE, *CDD;
        boolean simular;
        struct simular *simula;
};
```

As classes arvore e arvorev são utilizadas para fazer a interação entre o usuário e o Simulador e apresentar a ordenação topológica, criada pelo Simulador, da rede de conhecimento.

```
class arvore: dataobject [dataobj] {
  classprocedures:
        InitializeObject(struct arvore *self) returns boolean;
        FinalizeObject(struct arvore *self);
  methods:
        Limpa_Objeto();
        CriaNo(struct nos *pai, struct rede *item) returns struct nos *;
        DestroiNo(struct nos *no) returns boolean;
  macromethods:
        Raiz()        (self->raiz)
        Numero_Nos()  (self->numero_nos)
        No_Item(tn)   (tn->item)
        No_Pai(tn)    (tn->pai)
        No_Posicao(tn)      (tn->posicaofilho)
        No_Numero_Filhos(tn) (tn->numerofilho)
        No_Filho(tn, i) (tn->filhos[i])
  data:
        struct nos *raiz;       /* Aponta para o noh raiz da arvore.*/
        int numero_nos;         /* Contem o total de nos da arvore.*/
};

class arvorev: view {
```



```
  classprocedures:
        InitializeClass() returns boolean;
        InitializeObject(struct arvorev *self) returns boolean;
        FinalizeObject(struct arvorev *self);
  overrides:
        GetInterface(char *interface) returns char*;
        SetDataObject( struct arvore *arvore );
        Update();
        FullUpdate(enum view_UpdateType type, long x, long y, long w, long
h);
        Hit(enum view_MouseAction action, long x, long y, long numClicks)
              returns struct view *;
  methods:
        CoordenadaX(long desl, long width, long esp);
        CoordenadaY(long desl, long height, long esp);
        DesenhaNo(int no);
        DestacaNo(int no);
        SetHitFunction(procedure HitFunction, long rock);
  macromethods:
        GetAlturaTotal() (self->maxY - self->minY)
        GetComprimentoTotal() (self->maxX - self->minX)
  data:
        struct arvore *arvore;
        struct menulist *menu;
        long minY, maxY;
        long minX, maxX;
        int (*HitFunction) ();
        long rock1;
        long nodex;
        long nodey;
        long minimo;
        long maximo;
        struct nos *no_atual;
};
```

As classes programacao e programacaov são responsáveis pelo controle e interação do Arbcon, editor e o usuário.

```
class programacao : dataobject [dataobj] {
  classprocedures:
        InitializeObject(struct programacao *self) returns boolean;
        FinalizeObject(struct programacao *self) returns void;
  overrides:
        Read(FILE *file,long id) returns long;
        Write(FILE *file,long id,long level) returns long;
        ViewName() returns char *;
  macromethods:
        Arbcon() (self->arbcon)
        CText() (self->ctext)
  data:
        struct progtoo     *arbcon;
        struct ctext  *ctext;
};


class programacaov : view {
  classprocedures:
        InitializeClass() returns boolean;
        InitializeObject(struct programacaov *self) returns boolean;
        FinalizeObject(struct programacaov *self) returns void;
  overrides:
        PostMenus(struct menulist *menuList);
        FullUpdate(enum view_UpdateType tipo, long esq, long top, long
             comp, long larg);
```



```
        Hit(enum view_MouseAction acao, long x, long y, long clicks)
             returns struct view *;
        SetDataObject(struct programacao *programacao);
  macromethods:
        GetArbconV() (self->arbconv)
        GetProgramacao() (self->programacao)
        GetCtextV() (self->ctextv)
        GetFrame() (self->frame)
  data:
        struct programacao      *programacao;
        struct progtoov    *arbconv;
        struct ctextview   *ctextv;
        struct frame       *frame;
        struct lpair       *lp1;
        struct lpair       *lp2;
        struct scroll      *scroll;
        struct keystate    *state;
        truct menulist     *menus;
};
```