UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UM TUTORIAL INTELIGENTE PARA O AMBIENTE DELPHI

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

DANTON CAVALCANTI FRANCO JUNIOR

BLUMENAU, JUNHO/1999

1999/1-11

PROTÓTIPO DE UM TUTORIAL INTELIGENTE PARA O AMBIENTE DELPHI

DANTON CAVALCANTI FRANCO JUNIOR

ESTE TRABALHO DE CONCLUSÃO DE CURSO FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Maurício Capobianco Lopes — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Maurício Capobianco Lopes

_____

Prof. Marcel Hugo

_____

Prof. Jomi Fred Hübner

“A imaginação é mais importante que o conhecimento”.

(Albert Einstein)

Este trabalho é dedicado à minha família,
em especial a meu Pai e minha Mãe,
pelo apoio recebido ao longo de
toda a minha vida.

Agradecimentos

Gostaria de agradecer ao meu orientador, Prof. Maurício Capobianco Lopes, que me deu todo o apoio e incentivo na confecção deste trabalho, e em grande parte de minha vida acadêmica.

A meus pais, Danton Cavalcanti Franco e Dilcenira Cavalcanti, por todo a força e incentivo que me deram ao longo de minha vida.

A minha namorada Graciele Barbieri, que muitas vezes suportou minha ausência.

Aos professores, colegas e amigos do Curso de Ciências da Computação, principalmente ao meu amigo Rangel Gustavo Raulino, com certeza as discussões e conversas que tivemos foram de suma importância para nossa formação acadêmica.

A todos aqueles que direta ou indiretamente contribuíram para minha realização pessoal.

A Deus, pois sem Ele o hoje não seria uma realidade.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

CAI – *Computer Aided Instruction*

DLL – *Dynamic Link Libraries*

FURB – Fundação Universidade Regional de Blumenau

ICAI – *Intelligent Computer Aided Instruction*

MER – *Modelo Entidade Relacionamento*

POO – Programação Orientada a Objetos

SDI – *Single Document Interface*

SMA – Sistema Multiagente

STI – Sistema Tutor Inteligente

TI – Tutor Inteligente

UML – *Unified Modeling Language*

Resumo

Este trabalho tem como principal objetivo demonstrar a viabilidade da utilização de Sistemas Multiagentes na confecção de Sistemas Tutores Inteligentes. Os Sistemas Tutores Inteligentes têm demonstrado grande viabilidade quando bem utilizados e preparados. O ensino de um ambiente de programação não é nada fácil, desta forma um sistema automatizado pode ajudar em muito o desenvolvimento do aluno. Neste trabalho foi desenvolvido um protótipo de um Sistema Tutor Inteligente para o Ambiente Delphi, utilizando a técnica de Sistemas Multiagentes, possibilitando que o aprendizado seja facilitado. Na análise do protótipo foram utilizadas algumas etapas da Metodologia de Prototipação Fundamental e para a implementação utilizou-se a ferramenta de programação visual Delphi.

Abstract

This work aims to demonstrate the viability of Multiagent Systems use in the making of Intelligent Tutorial Systems. Intelligent Tutorial Systems has been demonstrating great viability when well used and prepared. The teaching of a programming language is not easy, this way an automated system that can help student's development. In this work a prototype of an Intelligent Tutorial System was developed for Delphi, using Multiagent Systems technique, this way that the learning is facilitated. In the analysis of the prototype were used some stages of Fundamental Prototypation Methodology and for the implementation was used Delphi.

1. Introdução

No passado o aprendizado era uma tarefa difícil, pois o conhecimento era adquirido através de um professor ou da leitura. Porém, isto muitas vezes gerava dúvidas por parte do aluno que, por receio ou esquecimento, deixava de lado muita coisa a ser aprendida.

Como forma de amenizar esse tipo de problema, surgiram, por volta da década de 50, os sistemas de ensino computadorizados, denominados *Computer Aided Instruction* (CAI). Os Tutores Inteligentes (TI), surgiram no final da década de 70, como aperfeiçoamento dos CAI [VIC98].

No entanto, os TI eram caracterizados como generativos, ou seja, eram geradas propostas de exercícios, e comparadas as respostas com as de seus usuários, o que os tornava bastante limitados. Para ajudar na solução deste problema surgiram os Sistemas Tutores Inteligentes (STI).

Os Sistemas Tutores Inteligentes (STI) baseiam-se em técnicas da Inteligência Artificial, utilizando-se de agentes capazes de interagir com o usuário e auxiliá-lo na solução dos problemas, ou seja, ensinam os alunos interagindo de forma totalmente automática e didática.

Os STI modernos evoluíram muito, principalmente com os avanços de suas ciências afins como a Inteligência Artificial (agentes), a Psicologia Cognitiva, Neurofisiologia, Neuropsicopedagogia e a própria Ciência da Computação que disponibilizou as tecnologias de Hipermídia, os Sistemas Distribuídos, as Redes de Comunicação, etc.

Uma das principais características dos STI é permitir a aprendizagem através de modelos cognitivos, que vem sendo cada vez mais utilizada, pois torna possível adaptar o sistema ao usuário, aprendendo com ele. Desta forma as simulações tornam-se mais realistas, tornando o processo de raciocínio do aluno mais aguçado.

Uma outra visão dos STI mais modernos é a possibilidade de se trabalhar com vários alunos ao mesmo tempo, na resolução de um mesmo problema, criando uma forma de se trabalhar em equipe, utilizando-se principalmente da tecnologia dos Sistemas Multiagentes.

Dada essa gama de recursos, a carência de ferramentas educacionais disponíveis no mercado e visando a melhora no ensino do ambiente Delphi, muito utilizado em nossa universidade, resolveu-se implementar um protótipo de um STI visando facilitar o aprendizado deste ambiente, auxiliando os alunos em seu desenvolvimento acadêmico.

1.1. Objetivos

O principal objetivo do trabalho é desenvolver o protótipo de um tutorial inteligente para o ambiente Delphi, visando demonstrar o funcionamento dos STI. Para tanto serão utilizadas técnicas de Inteligência Artificial (IA), mais especificamente, os recursos de Sistemas Multiagentes.

Os objetivos secundários do trabalho são:

a) estudar o funcionamento e uso dos STI;
b) avaliar uma metodologia adequada para o ensino do ambiente Delphi.

1.2. Estrutura

O trabalho foi dividido em seis capítulos, descritos a seguir.

O primeiro capítulo define os objetivos do trabalho, apresentando a justificativa para seu desenvolvimento e seus objetivos.

O segundo capítulo apresenta uma visão geral sobre os Sistemas Tutores Inteligentes, do qual o trabalho propõe-se a utilizar, mostrando conceitos, tipos, problemas e utilidades dos mesmos.

O terceiro capítulo enfatiza os Agentes e os Sistemas Multiagentes, conceituando, caracterizando, levantando limitações e demonstrando suas aplicações.

O quarto capítulo descreve o ambiente de programação Delphi 3.0, que é o ambiente sobre o qual será criado o tutorial e que será ensinado através dos agentes do protótipo.

O quinto capítulo apresenta a Metodologia de Prototipação Fundamental, proposta para o desenvolvimento do trabalho, bem como a análise dos dados, as características, o desenvolvimento e a utilização do modelo criado.

O sexto capítulo completa o trabalho, apresentando as conclusões, limitações e sugestões para serem implementadas e aprimoradas quando da utilização do modelo desenvolvido.

2. Sistemas Tutores Inteligentes

Neste capítulo serão verificados alguns conceitos sobre aprendizagem, *softwares* educacionais e Sistemas Tutores Inteligentes.

2.1. Método de Aprendizagem

Segundo Vasconcelos [VAS97],

> *"aprendizado é o resultado de todo um processo educacional que se dá pela interação entre o professor e o estudante, ocorrendo num ambiente institucional".*

Wilhelm [WIL97] define aprendizagem como:

> *"um processo estritamente individual no qual o aluno necessita vivenciar situações e praticar atitudes capazes de desenvolver de forma gradativa conhecimentos e habilidades inatas".*

Desta forma Wilhelm [WIL97] destaca que o paradigma do ensino clássico, onde o foco principal é o instrutor, deve sofrer mudanças que sigam em direção a uma adequação dos processos educacionais. Estas mudanças devem considerar as inteligências múltiplas humanas e as funções dos recursos tecnológicos.

Deste modo, um sistema estruturado que auxilie no aprendizado pode, de certa forma, fazer com que haja uma certa melhora na performance do aluno para a aquisição do conhecimento.

Visando o apoio a este processo (ensinar), têm sido propostos muitos sistemas computadorizados, que dão ênfase à relação professor-estudante, considerando os recursos disponíveis de informática [VAS97].

Giraffa e Oliveira [GIR95] destacam que o uso de computadores pessoais nas mais diversas áreas do conhecimento vem sendo cada vez mais difundido e permeia o dia-a-dia de todos os segmentos da sociedade. Ressaltam também que a escola tem apenas reagido ao crescimento tecnológico, sendo que o planejamento pedagógico tem sido orientado pela

tecnologia e esta parece ter ditado normas para a utilização do computador na sala de aula, quando na verdade deve-se refletir cuidadosamente sobre as necessidades e objetivos pedagógicos e depois selecionar métodos e recursos disponibilizados pela tecnologia para o suporte ao processo de ensino-aprendizagem. Deve-se ter, na tecnologia, uma ferramenta que proporciona novas possibilidades às tarefas discentes e docentes.

Desta forma a educação deixa de ser sinônimo de transferência de conhecimento para as gerações mais jovens, e passa a ser uma renovação constante das necessidades, ou seja, uma educação continuada [VAV98].

2.2. *Softwares* Educacionais

Segundo Giraffa e Oliveira [GIR95],

> *"o software educacional é um programa que visa atender necessidades e possui (ou deve possuir) objetivos pedagógicos".*

Giraffa em [GIR97], complementa que *software* educacional é um programa de computador desenvolvido para auxiliar as necessidades de alunos e professores no que concerne ao processo de ensino-aprendizagem. Desta forma é possível considerar que todos os programas podem ser usados para fins educacionais, porém, os mesmos necessitam utilizar uma metodologia adequada onde são ressalvados os aspectos educacionais, ou seja, devem ser contemplados os objetivos a nível de ensino-aprendizagem que o professor deseja atingir junto aos seus alunos.

Giraffa e Oliveira [GIR95] levantam as conseqüências pedagógicas do uso de um *software* educacional:

- individualização na aprendizagem;
- estímulo e motivação para o aluno;
- promoção da auto-estima no aluno;
- apresentação dos tópicos de modo atrativo, criativo e integrado.

Para Mesquita e Silva [MES98], a construção de *softwares* educacionais enfrenta hoje vários problemas como o tempo, o custo de desenvolvimento e a dificuldade de implementar mecanismos que facilitem a adaptação do conhecimento de ensino de acordo com as características de cada aprendiz. A solução destes problemas se dá pela utilização das

ferramentas de autoria para *software* educacionais. Elas permitem que a construção de um *software* educacional seja feita pelo próprio professor, sem a necessidade de um profundo conhecimento de programação, e nem de uma equipe de especialistas. As principais funções de controle e interação já estarão presentes na própria ferramenta de autoria. Basta o professor incluir o material de ensino (domínio de conhecimento) para que o *software* funcione.

Também os Sistemas Tutores Inteligentes (STI) são *softwares* de propósito educacional que utilizam técnicas de Inteligência Artificial para aquisição e representação do conhecimento [TOM98].

2.3. Fundamentos de Sistemas Tutores Inteligentes

Os Tutores Inteligentes começaram a ser utilizados no final da década de 70 e vieram como uma evolução dos *Computer Aided Instruction* (CAI) – Instrução Auxiliada por Computador. Nesta época os sistemas eram considerados generativos, ou seja, capazes de gerar automaticamente o material instrucional, no todo ou em parte. Juntamente com os TI, surgiram os STI. Porém, num primeiro momento, os STI tentavam superar algumas limitações dos sistemas generativos.

Num segundo estágio (segunda metade da década de 80), as pesquisas dos STI eram mais voltadas para as questões pedagógicas, ou seja, como aperfeiçoar o componente de ensino destes sistemas. Foi nesta época que os STI tiveram um grande avanço em sua divulgação.

No terceiro estágio (década de 90), os STI estão se tornando sistemas com alto grau de sofisticação, tornando-se cada vez mais flexíveis, caminhando para estratégias de ensino que se adaptam às necessidades do usuário em particular [VIC98].

O uso dos STI para o ensino pode representar um ganho muito grande em relação ao aprendizado do aluno. Zuanábar [ZUA96] faz um comparativo entre STI e o ensino tradicional por professores (tabela 2.1):

Sistemas Tutores Inteligentes	Professor
Ensino individual	Ensino clássico
Permanente	Volátil
Fácil transferência	Difícil transferência
Consistente	Inconsistente
Custo baixo de operação	Alto custo de operação

Tabela 2.1: Vantagens dos STI sobre os professores. Fonte: [ZUA96].

O objetivo do STI não é o de substituir o professor em uma sala de aula por um ou vários computadores, enfatizando somente o trabalho individual máquina-estudante. Uma das vantagens que os STI apresenta é permitir que o estudante dirija a instrução segundo suas necessidades.

O objetivo fundamental dos Sistemas Tutores Inteligentes é proporcionar uma instrução adaptada ao aluno. Os Tutores se comportariam de forma mais próxima a um professor humano ou um comportamento mais próximo possível disto. Porém, a realidade está muito distante de alcançar tais propósitos [VIC96].

2.3.1. Características dos Sistemas Tutores Inteligentes

Enquanto os Tutores Inteligentes induzem o aluno a uma resposta "correta" mediante uma série de estímulos cuidadosamente planejados, os Sistemas Tutores Inteligentes, objetivam simular algumas capacidades cognitivas do aluno e utilizam estes resultados como base das decisões pedagógicas a tomar.

Segundo Viccari [VIC90], os STI são programas que, interagindo com o aluno, modificam suas bases de conhecimento, percebem suas intervenções e possuem a capacidade de aprender e adaptar as estratégias de ensino de acordo com o desenrolar do diálogo com o aluno.

Estes sistemas caracterizam-se principalmente por construir um modelo cognitivo do aluno, através da interação, e, através da formulação e comprovação de hipótese sobre o estilo cognitivo do aluno, sobre o seu procedimento, o seu nível de conhecimento do assunto e suas estratégias de aprendizagem. Também possuem uma capacidade de formular uma estratégia de ensino-aprendizagem adequada ao aluno e à situação do momento.

A tabela 2.2 proposta por Viccari e Giraffa [VIC96], apresenta as diferenças entre um Sistema CAI (precursor dos Sistemas Tutores) e um Sistema Tutor Inteligente (ICAI[1]).

Aspecto	Sistema CAI	Sistema ICAI
Origem	Educação	Ciência da Computação
Bases Teóricas	Skinner (Behaviorista)	Psicologia Cognitivista
Estruturação e Funções	Uma única estrutura algoritmicamente pré-definida, onde o aluno não influi na seqüenciação	Estrutura subdividida em módulos cuja sequenciação se dá em função das resposta do aluno
Estruturação do Conhecimento	Algorítmica	Heurística
Modelagem do Aluno	Avaliam a última resposta	Tentam avaliar todas as respostas do aluno durante a interação
Modalidades	Tutorial, exercício e prática, simulação e jogos educativos	Socrático, ambiente interativo, diálogo bidirecional e guia

Tabela 2.2: Comprativo entre CAI e STI. Fonte: [VIC96].

2.3.2. Composição dos STI

Os STI são compostos por quatro módulos [RAM97]:

a) **Módulo Especialista (Domínio):** neste módulo está representado/ armazenado todo o conhecimento que o sistema tem sobre um determinado assunto. As principais decisões tomadas neste módulo são como adquirir o conhecimento e como o mesmo ficará armazenado (ou representado), sendo estes os atuais assuntos de pesquisa nesta área.

b) **Módulo do Aprendiz:** responsável por manter uma informação detalhada sobre a evolução do estudante no assunto. Vale ressaltar que ele guarda uma informação individualizada, ou seja, é personalizado para cada estudante que utiliza o sistema.

c) **Módulo Tutor:** neste módulo, a partir da informação contida no módulo do aprendiz e do conhecimento armazenado no módulo do domínio, ocorre a seleção e seqüenciamento do assunto a ser apresentado sendo enviado, posteriormente, ao módulo de comunicação.

[1] Para a comunidade científica, STI e ICAI também são utilizados como sinônimos.

d) **Módulo Comunicação:** é a interface entre o Sistema Tutor Inteligente e o aprendiz ou usuário no mundo real. A principal característica no projeto de uma interface é sua transparência, de modo a facilitar ao aprendiz as tarefas de:
   - ter que aprender sobre algo que não domina;
   - ter que usar a tecnologia para esta aprendizagem por si mesmo.

A figura 2.1 representa a interação entre os módulos de um STI.

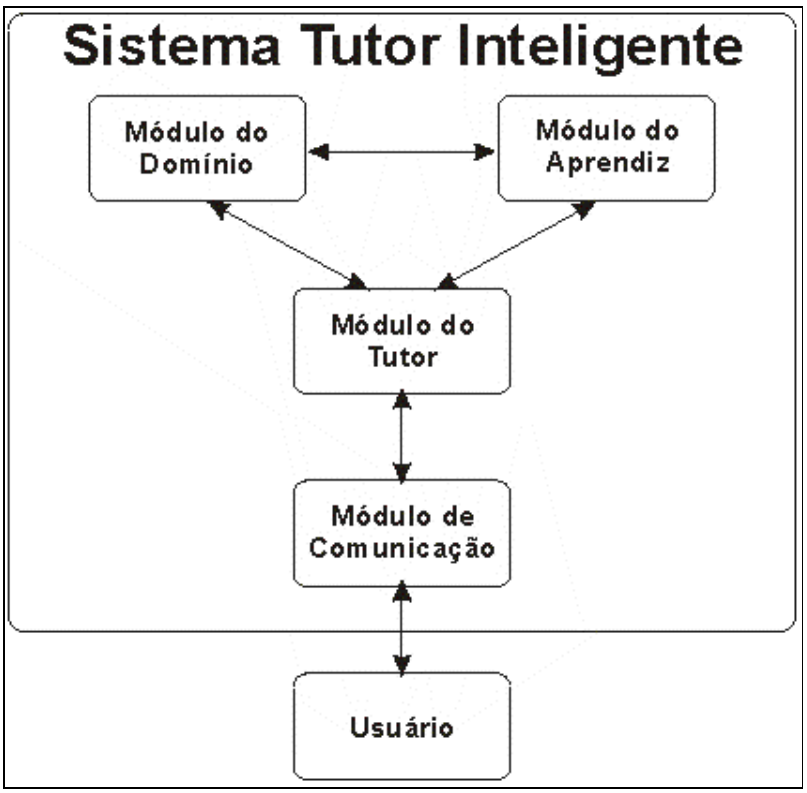

Figura 2.1: Esquema de interação dos módulos de um STI.

Ainda segundo Ramos [RAM97], os STI surgiram da aplicação da Inteligência Artificial na educação e treinamento, visando substituir a tradicional instrução auxiliada por computador. Os Sistemas Tutores Inteligentes (STI) devem apresentar três características que denotam inteligência:

a) o assunto deve ser conhecido bem o suficiente para que o sistema faça inferências sobre o domínio ou resolva problemas que estejam em seu escopo de atuação;
b) deve ser capaz de avaliar o aluno;

c) deve fornecer estratégias pedagógicas que minimizem a diferença entre o aprendiz e o especialista.

Para Giraffa e Oliveira [GIR95], um STI deve possuir uma interface atrativa, pois ela será de importância fundamental para definir a boa utilização do *software*. Assim como os produtores de filmes utilizam vários efeitos para chamar a atenção do público, um STI deve possuir um atrativo similar (cor, imagens, etc.) para tornar-se atrativo também.

2.4. Exemplos de STI

Alguns exemplos de STI desenvolvidos são:

a) Princípios de Geometria [RAM98]**:** este protótipo é utilizado para apoiar o ensino de Geometria Plana, assunto esse identificado como de maior interesse para o professor devido à grande disparidade de conhecimentos dos alunos oriundos de outras escolas.

Corresponde a uma ferramenta onde o próprio aluno pode avaliar o seu grau de compreensão das aulas assistidas, além de rever os conceitos aprendidos em sala.

Sua utilidade pode ser comprovada também pelo professor, que possui, à sua disposição, um instrumento didático eficiente que o auxilia nas revisões da matéria, além de ser de fácil manuseio. O professor também pode, com este instrumento, avaliar cada aluno ao final das seções e aluno por aluno através de tabelas acertivas e gráficos além de poder armazenar todas estas informações em um banco de dados.

b) Sistema para Treinamento de Operadores Utilizando STI [RAM97]**:** este trabalho apresenta uma aplicação de um STI para o treinamento de operadores de uma central de controle de processos. Sua finalidade é auxiliar um professor na tarefa de treinamento de operadores de uma Unidade de Recuperação de Solventes em manobras operacionais realizadas a partir do painel de controle durante instabilidades operacionais, paradas e partidas de equipamentos.

Desta forma é possível treinar o operador para agir o mais rápido possível em uma situação de emergência real, evitando danos severos em equipamentos ou perda de qualidade dos produtos.

c) **Sistema de Ensino Inteligente para o Diagnóstico de Epilepsia com o Paradigma de Sistema Multiagentes** [VAS97]**:** neste sistema em desenvolvimento por Vasconcelos, os agentes tutor, especialista e estudante entram num diálogo, com a finalidade de descobrir seus interesses, a partir deste é criado o "Tema Gerador", que dará lugar a estrutura curricular a ser seguida e as suas formas de trabalho.

   Durante toda a sessão estes diálogos persistirão de forma a propiciar meios de adaptação entre os agentes, redirecionando os temas, aprofundando outros e esclarecendo dúvidas.

d) **Leonardo: Máquina para Auxílio à Resolução de Problemas** [GIR95]**:** o *software* visa auxiliar o aluno na fase anterior à formalização do algoritmo, a qual é denominada de "análise do problema". Nesta fase o aluno faz o desmembramento do problema, identifica as partes que o compõem, monta a estratégia de resolução (com as ferramentas que dispõe) e pode testar a seqüenciação criada e o resultado sem precisar entrar no nível de uma linguagem mais formalizada.

   O sistema apresenta o problema (anteriormente especificado pelo professor) ao aluno, descrevendo sua temática. Mostra os dados de entrada e a saída de dados desejada. Através da correta utilização e ordenação das ferramentas disponíveis, o aluno deve solucionar este problema, podendo manipular e testar à vontade os recursos que tem.

3. Agentes

Neste capítulo serão enfatizados os conceitos de Agentes e de Sistemas Multiagentes, apresentando características, levantando limitações e demonstrando suas aplicações. No final deste capítulo, será apresentado uma formalização do uso de Sistemas Multiagentes para a implementação de Sistemas Tutores Inteligentes.

3.1. Definições

O termo agente não possui uma definição unanimemente aceita, pelo fato de ser usado para um amplo e heterogêneo campo de pesquisa e desenvolvimento e pela multiplicidade de enfoques sob os quais é estudado. Desta forma os pesquisadores não chegaram a um consenso sobre o termo agente [POR97].

Para Huhns e Singh [HUH97], agentes são (*software*) ativos, componentes persistentes que percebem, argumentam, agem e se comunicam.

Já Russell e Norvig, [RUS95] definem agentes como sendo sistemas capazes de perceber, através de sensores, e agir em um dado ambiente através de atuadores. Um Agente Ideal é um agente que executa a ação considerando a maximização de uma medida de performance que ele avalia a partir de suas percepções do mundo.

Para Alvares e Sichman [ALV97], agente é uma entidade real ou virtual, imersa num ambiente sobre o qual ele é capaz de agir, que dispõe de uma capacidade de percepção e de representação parcial deste ambiente, que pode se comunicar com outros agentes e que possui um comportamento autônomo, conseqüência de suas observações, de seu conhecimento e de suas interações com outros agentes. Além disso, pode-se associar uma identidade única a um agente. Um agente pode ser considerado como um meio que produz um certo número de ações, a partir de seu conhecimento e dos mecanismos internos que lhe são próprios.

Thompson [THO97 apud POR97] define agente da seguinte forma:

> *"É uma peça de software que executa uma determinada tarefa empregando informação extraída de seu ambiente para agir de forma adequada no*

sentido de completar sua tarefa de modo bem sucedido. O software deve ser capaz de adaptar-se a eventuais modificações ocorridas em seu ambiente de modo a que o resultado pretendido seja independentemente alcançado".

3.2. Especificação de Agentes

Hübner [HÜB95] especifica que um agente é formado por um conjunto de processos (seu funcionamento), mais as informações do agente (identificação, estrutura, etc.) que o identificam como único. Na figura 3.1, pode-se observar a estrutura de um agente segundo esta especificação.

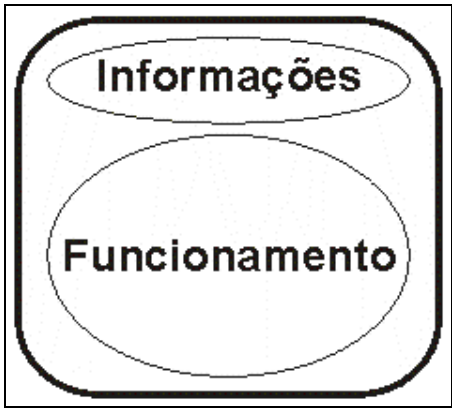

Figura 3.1: Estrutura do agente.

Desta forma fazendo-se uma analogia com a Programação Orientada a Objetos (POO), as Informações seriam as propriedades de uma classe, e o Funcionamento, os métodos a ela associados.

3.3. Características dos Agentes

Huhns e Singh [HUH97], Porto, Palazzo e Castilho [POR97] e Hübner [HÜB95], apresentam características dos agentes, que estão agrupadas a seguir:

a) **quanto à Autonomia/Design:** podem funcionar sem intervenção direta de operadores de qualquer tipo e possuir controle sobre suas ações e seu estado interno;

b) **quanto à Habilidade Social:** podem interagir com outros agentes e com seres humanos por meio de algum tipo de linguagem de comunicação;

c) **quanto à Reatividade:** podem perceber o seu ambiente e responder aos estímulos dele recebidos. Neste caso podem ser reativos ou cognitivos. Os agentes reativos possuem representação implícita, não têm história e o controle não é deliberativo, ou seja, está baseado em um modelo de organização biológica (como formigas), onde cada elemento em si não possui inteligência, mas a coletividade sim. Já os agentes cognitivos possuem uma representação explícita, têm história, o controle é deliberativo e existe organização social. Este tipo de agente, entre outras coisas, planeja ações futuras, criando planos que tornam-se mais um elemento dos estados mentais do agente;

d) **quanto à Iniciativa:** além da capacidade de reagir ao ambiente, podem exibir um comportamento orientado à satisfação de seus objetivos;

e) **quanto à Continuidade Temporal:** podem ser constituídos de processos de execução contínua, podendo tanto estar ativo, quanto adormecido[2];

f) **quanto à Complexidade:** podem ter capacidade de lidar com tarefas complexas de alto nível, tomando decisões de segmentá-la em sub-tarefas, quando necessário;

g) **quanto à Mobilidade:** podem ser estáticos ou móveis. Neste último caso devem ter a habilidade de movimentar-se em uma rede, ocupando diferentes nódos e recursos ao longo do tempo;

h) **quanto à Benevolência:** não podem apresentar objetivos conflitantes, sendo que cada agente irá sempre tentar fazer o que lhe for pedido;

i) **quanto à Racionalidade:** podem agir de forma a atingir seus objetivos e não contra eles, pelo menos dentro do alcance de suas crenças;

j) **quanto à Adaptabilidade:** podem possuir a capacidade de se adaptar aos hábitos, métodos de trabalho e preferências de seus usuários;

k) **quanto à Colaboração/Cooperação:** não podem executar instruções impensadamente, devendo ter a capacidade de interagir com outros agentes e com seres humanos, recusando ordens que possam vir a acarretar algum dano ao sistema;

---

[2] Alguns autores referem-se ao termo processo ativo como sendo processo em *foreground*, e processo adormecido como sendo processo em *background*.

l) **quanto ao Tempo de Vida:** podem ser passageiros ou ter muito tempo de "vida".

Os agentes não precisam ter necessariamente todas as características descritas acima. Destaca-se que em algumas delas os agentes são ou uma coisa ou outra, como por exemplo, quanto a mobilidade e quanto a reatividade.

Porto, Palazzo e Castilho [POR97] destacam que é fundamental que os agentes tenham pelo menos autonomia (os agentes podem operar por si próprios), aprendizado (capacidade de aprender, tomando iniciativas) ou cooperação (capacidade de interagir com outros agentes). No entanto é importante que eles possam combinar estas características, formando os chamados agentes híbridos. Esta visão parcial da tipologia de agentes está representada na figura 3.2.

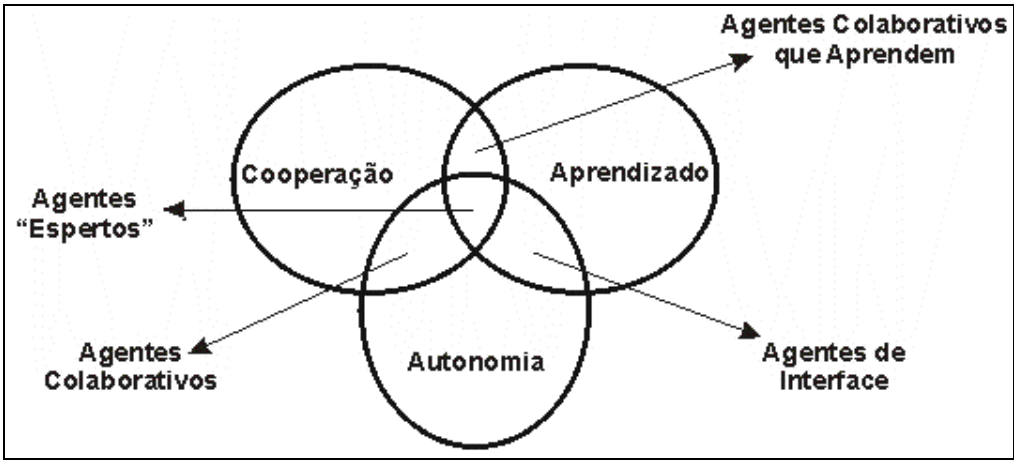

Figura 3.2: Visão parcial de uma tipologia de agentes.

3.4. Sistemas Multiagentes

Para Alvares e Sichman [ALV97], Sistemas Multiagentes (SMA) devem conceber meios de resolver um problema a partir da colaboração entre os agentes dentro de um sistema, sendo que:

- os agentes são concebidos independentemente de um problema em particular;
- a concepção das interações também é independente de uma aplicação alvo em particular;
- o mesmo vale para o projeto das organizações;

- não deve haver um controle global do sistema, este deve ser implementado de forma totalmente descentralizada nos agentes.

Os problemas do mundo real são melhor modelados usando-se Sistemas Multiagentes, ao invés de um único agente. Com SMA é possível opor-se às limitações dos agentes individuais, além de utilizar toda a capacidade de um sistema distribuído, como sua robustez, tolerância a falhas, paralelismo e escalabilidade [WEI97 apud HUH97].

Para Tan [TAN97 apud HUH97], em SMA o aprendizado dos agentes torna-se fácil, pois um agente individual não aprende tudo por sua própria descoberta, mas quando, em um ambiente, eles trocam informação e conhecimento entre si e aprendem com os outros agentes. Quando uma tarefa é muito grande para um único agente, eles podem cooperar entre si para realizá-la.

Enquanto um agente em particular pode não ter conhecimento a respeito de alguma coisa, podem existir outros agentes pertencentes a outros usuários que detém este conhecimento. Desta forma em vez de ter que aprender a respeito de algo, o agente pode pedir ajuda a outro agente para que o mesmo execute o procedimento desconhecido [LAS97 apud HUH97].

Nos SMA cada um dos agentes no sistema pode estar trabalhando em metas diferentes, ou até mesmo em metas contraditórias. Deve-se lidar com um sistema onde os agentes múltiplos estão competindo ou cooperando entre si [ROS97 apud HUH97].

Liu e Sycara [LIU97 apud HUH97] destacam que na maioria dos Sistemas Multiagentes atuais a interação entre os agentes ocorre somente quando um agente possui dados, fatos, visões, e soluções que são de interesse a outros agentes, ou quando os agentes precisam resolver estes conflitos. Em outras palavras a atividade de coordenação, embora essencial, não constitui uma parte significativa de um esforço de agentes para alcançar sua meta.

Segundo Hübner [HÜB95], em uma sociedade de agentes é necessário que cada agente conheça o seu papel e seja capaz de identificar os papéis de outros agentes em seu meio. Desta forma torna-se mais ágil o ambiente social dos agentes, impedindo por exemplo que um agente solicite uma informação a outro agente que a desconhece.

Em sociedades não abertas (número de agentes fixo), cada agente pode saber previamente quais os papéis dos outros agentes, sendo que para sociedades abertas, a recíproca não é verdadeira, pois os agentes devem identificar o papel do novo agente e vice-versa.

3.5. STI Utilizando SMA

As propostas de utilização de arquiteturas SMA em STI trazem uma grande vantagem em relação às arquiteturas tradicionais de STI: apresentam uma flexibilidade maior no tratamento dos elementos que compõem o sistema. Além disso, o fato de usar agentes para modelar os componentes de um STI possibilita o agrupamento da arquitetura tradicional (um módulo = um agente) ou na explosão de cada módulo. Neste último caso, o refinamento pode chegar até os estados mentais de um agente [GIR97].

Para Tambe, Johnson e Shen [TAM97 apud HUH97], nos SMA os agentes devem atuar dinamicamente e em tempo-real (*real-time)*, para permitir, desta forma, que a interação seja altamente flexível e reativa. Por exemplo, os STI precisam interagir com estudantes enquanto eles estão resolvendo problemas, o que torna o treinamento mais dinâmico.

Os itens **c** e **d** do capítulo 2, subtítulo 2.4 (Exemplos de STI), são exemplos de STI que utilizam SMA.

4. Delphi

Neste capítulo será descrito o ambiente de programação Delphi 3.0, que é o ambiente sobre o qual será criado o tutorial e que será ensinado através dos agentes do protótipo.

4.1. Características do Ambiente Delphi

O Delphi é um ambiente de programação voltado para facilitar o desenvolvimento de sistemas. Possui uma linguagem visual dotada de recursos gráficos do padrão Windows, incorpora recursos dedicados a banco de dados, aproveita as bibliotecas de outros programas, gera código executável e oferece soluções para tratamento de questões de ergonomia, segurança, consistência e integridade de dados [WIL97].

Davis [DAV95] apresenta alguns recursos e vantagens de se utilizar Delphi:

- a capacidade de criar verdadeiros arquivos executáveis independentes e bibliotecas de vínculos dinâmicos (DLL);
- uma área de trabalho (*desktop*) de desenvolvimento visual elegante e eficiente;
- uma linguagem de programação subjacente ao mesmo tempo simples e eficiente;
- um ambiente que encoraja a reutilização por meio do uso de gabaritos e componentes;
- a capacidade de ampliar o Delphi criando novos componentes a partir dele mesmo;
- a capacidade de responder a todas as mensagens do Windows;
- uma implementação ponderada e completa de conceitos de herança de classe e encapsulamento, permitindo a utilização da programação orientada a objeto.

Uma outra característica importante do Delphi é a possibilidade de uso da técnica de programação orientada a objetos (POO) e da programação orientada a eventos [ANS], [DAV] e [WAR].

4.2. O Ambiente Delphi

O Delphi é composto por uma janela principal que controla outras janelas, criando uma interface Windows, tipo SDI. A figura 4.1 mostra as principais janelas do Delphi (os números representam cada elemento descrito) [ANS97], [CAN98], [DAV95], [DUN96] e [WAR96]:

1) **Janela *Main*:** contém o menu com os comandos e funções do Delphi, a paleta de componentes (*Component Palette*) – que disponibiliza os elementos que serão utilizados pela aplicação – e a barra de botões de acesso rápido aos comandos mais utilizados (*SpeedBar*) – que fornece um atalho aos comandos e funções mais utilizados do Delphi;
2) **Janela *Form Designer*:** é o ponto central de desenvolvimento do Delphi. Através do *Form Designer*, são desenhadas as janelas do um projeto em desenvolvimento. Qualquer alteração efetuada aqui, será automaticamente refletida na interface, demonstrando como será a aplicação final;
3) **Janela *Code Editor*:** é o editor de códigos do ambiente, providenciando total acesso ao código gerado pelo projeto, além de incluir recursos de edição (copiar, colar, pesquisar, etc.). Uma outra característica importante do *Code Editor*, é a possibilidade de se formatar a visualização do código conforme o desejo do usuário;
4) **Janela *Object Inspector*:** providencia a conexão entre a interface visual e o código. É composto por duas páginas – *Properties* (propriedades) e *Events* (eventos) – que mostram as propriedades e os eventos do objeto selecionado. Disponibiliza um fácil caminho para a personalização dos objetos.

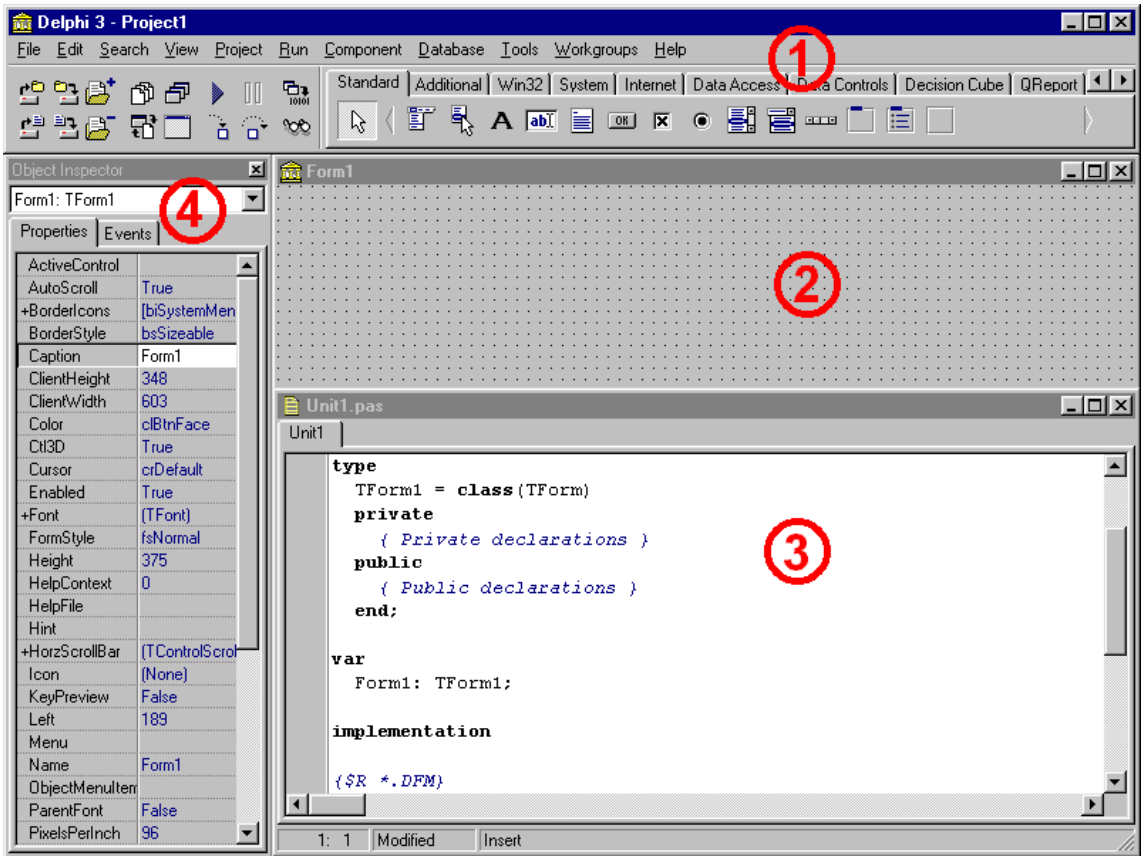

Figura 4.1: O Delphi e suas principais janelas.

4.3. Componentes

Por ser uma linguagem voltada a orientação a objetos, o Delphi utiliza o conceito de classes. Segundo Cantu [CAN98],

> *"Classe refere-se a um tipo de dado, e objeto a uma instância de tipo de dado, uma variável".*

As classes possuem propriedades e métodos, sendo fundamental ao programador compreender suas definições. Além disto, elas agregam o conceito de eventos, ou seja, tem a capacidade de responder a uma determinada ação feita sobre a classe.

O Delphi possui dois tipos de classes: as classes não visuais e os componentes. As classes não visuais não apresentam interface direta com o usuário (desenvolvedor), podendo ser utilizadas apenas através do código do programa. Já os componentes são classes que oferecem interação direta com o usuário, podendo ser definidos e configurados através da interface, durante a fase de desenvolvimento do projeto.

A partir do momento que são inseridos em um projeto, os componentes tornam-se objetos[3] da classe selecionada.

Existem dois tipos de componentes:

- **Visuais:** podem ser visualizados/alterados diretamente pelo usuário, inclusive podendo-se definir sua posição, tamanho, forma, etc., durante o desenvolvimento do projeto (figura 4.2).

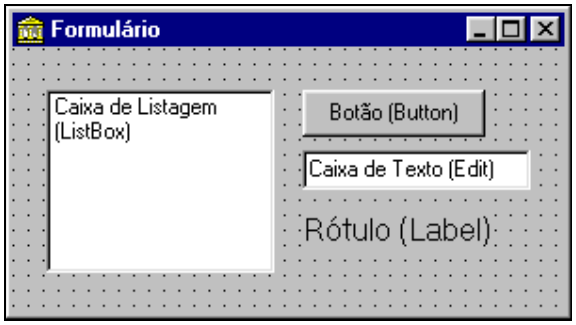

Figura 4.2: Exemplos de componentes visuais.

- **Não-Visuais:** não apresenta uma interface que possa ter interação com o usuário, podendo apenas ter propriedades e eventos definidos durante o projeto. (figura 4.3).

Figura 4.3: Exemplos de componentes não-visuais.

4.4. Propriedades

As propriedades são os atributos que definem como os componentes são exibidos e como funcionam no aplicativo em execução [DUN96].

Existem três tipos de propriedades:

[3] Quando um componente é inserido em um projeto, ele passa a ser denominado objeto (um objeto do projeto).

- **Propriedades em Tempo de Projeto:** estas propriedades, podem ser alteradas durante o desenvolvimento do projeto através do *Object Inspector*, e suas alterações refletem imediatamente no componente (Ex.: *Width* – Largura, *Height* – Altura, *Caption* – Legenda).
- **Propriedades em Tempo de Execução:** este tipo de propriedade somente pode ser alterada no momento em que a aplicação está executando (*ComponentIndex* – Índice do componente, *TableLevel* – Nível da tabela), este tipo de propriedade não está disponível em tempo de projeto.
- **Propriedade Read-only (somente leitura):** este tipo de propriedade não pode ser alterada, pois seu valor apenas pode ser lido, e está disponível somente em tempo de execução (Ex.: *Components* – Componentes, *ComponentState* – Estado do componente).

Algumas propriedades, possuem sub-propriedades (propriedades que apresentam um sinal de mais "+" ao lado se seu nome, apresentam esta característica), ou seja é possível configurar a propriedade em questão detalhadamente (Ex.: a propriedade *Font* – Fonte, possui sub-propriedades como *Name* – Nome, *Size* – Tamanho, *Style* – Estilo).

4.5. Eventos

Por ser totalmente baseado em Windows, o Delphi funciona através de eventos.

Uma boa definição de eventos é apresentada por Warner e Goldsman [WAR96]:

> *"Em termos de informática, um evento é algo que ocorre – realmente, qualquer coisa. Pressionar uma tecla é um evento. Mover o mouse é um evento. O computador possui um relógio interno que faz um "tique" diversas vezes por segundo; um "tique" deste relógio é um evento. Se sua impressora ficar sem papel, isto é um evento".*

Desta forma, para cada evento deve-se escrever o código (o que será processado) no momento em que este evento ocorrer. Isso não quer dizer que a lógica seqüencial possa ser esquecida; ela existe porém de forma oculta ao usuário [DAV95].

Para visualizar/alterar os eventos dos objetos, utiliza-se o *Object Inspector*, que relaciona todos os eventos associados ao objeto em questão.

São exemplos de eventos:

- **OnMouseMove:** disparado quando o *mouse* move-se sobre o objeto;
- **OnClick:** disparado quando um objeto recebe o clique do *mouse*;
- **OnKeyDown:** ocorre quando o usuário pressiona uma tecla sobre o selecionado.

4.6. Métodos

Para Cantu [CAN98], um método é:

> *"Um tipo especial de função ou procedimento que está relacionado ao tipo de dados, uma classe".*

Métodos são procedimentos ou funções específicas, associadas a um objeto [DAV95] e [WAR96]. Desta forma, a ação efetuada sobre um objeto está relacionada a um método.

São exemplos de métodos:

- **Create:** cria um componente ou objeto;
- **Refresh:** atualiza a exibição de um componente ou objeto;
- **Focused:** verifica se determinado objeto ou componente possui o foco em determinado instante.

5. Desenvolvimento do Protótipo

Neste capítulo será apresentada a Metodologia de Prototipação Fundamental, proposta para o desenvolvimento do trabalho, bem como a análise dos dados, as características, o desenvolvimento e a utilização do modelo criado.

5.1. Metodologia Adotada

Para o desenvolvimento do protótipo utilizou-se a Metodologia de Prototipação de Sistemas de Informação, mais especificamente a Metodologia de Prototipação Fundamental ou Básica, que utiliza o protótipo como o próprio Sistema. Essa metodologia de prototipação é proposta por Rubem Melendez Filho [MEL90], que se baseia em 8 etapas. Porém apenas as cinco primeiras serão utilizadas neste protótipo, sendo que a quinta etapa foi executada parcialmente.

As etapas da Metodologia de Prototipação Fundamental, são:

1) **Necessidades do Usuário:** esta etapa é destinada à identificação dos objetivos do sistema, baseado nas necessidades dos usuários, os problemas que os mesmos têm e o que eles esperam do sistema.
2) **Levantamento de Requisitos:** nesta etapa deve-se fazer um levantamento dos recursos de *hardware* e *software* necessários, bem como o exame dos fatos geradores do sistema, além das políticas e diretrizes básicas para a implantação do ambiente de prototipação. A análise dos dados, elaboração do modelo lógico, análise das funções, etc., também são fatos a serem observados nesta fase.
3) **Desenvolvimento do Protótipo:** construção do protótipo, baseado nos dados das etapas anteriores.
4) **Demonstração do Protótipo:** nesta etapa o protótipo desenvolvido é apresentado ao usuário para que sejam feitos os testes de uso e identificadas as falhas do protótipo.

5) **Revisão do Protótipo:** identificados os problemas na fase anterior, deve-se analisar o protótipo, melhorando as falhas e, até mesmo, descartando funções que foram consideradas redundantes.
6) **Elaboração da Documentação do Usuário:** elaboração de uma documentação mínima para o usuário, incluindo aspectos sobre operação básica, esquemas de segurança, roteiro de utilização, instruções gerais sobre eventos executados, etc.
7) **Incorporação de Esquemas de Segurança:** como o protótipo é o próprio sistema final, deve-se garantir todos os requisitos relacionados com a área de produção e de administração de banco de dados.
8) **Implantação:** colocação do sistema em funcionamento, treinando o usuário, alocando recursos necessários, etc.

A figura 5.1, mostra graficamente as etapas da Metodologia de Prototipação Fundamental.

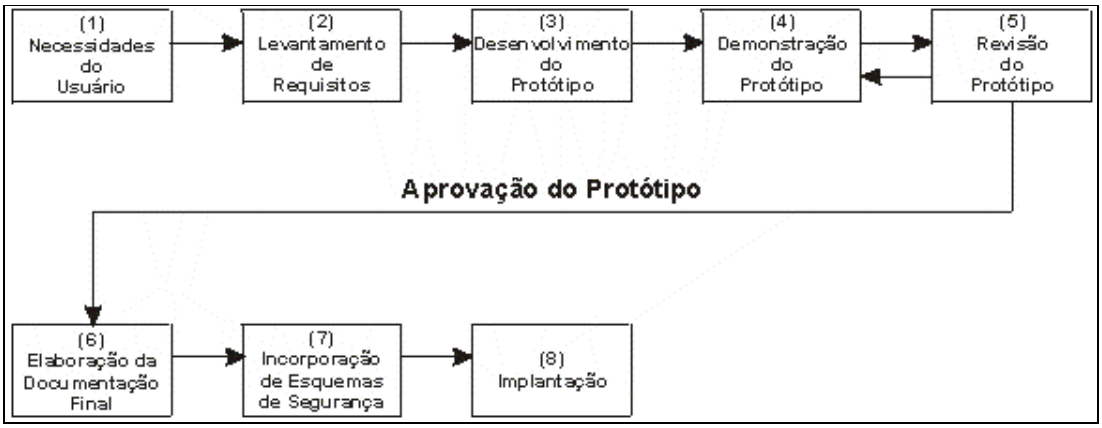

Figura 5.1: Etapas da Metodologia de Prototipação Fundamental.

5.1.1. Necessidades do Usuário

A arte de ensinar e de aprender nunca foi uma tarefa fácil. O professor enfrenta dificuldades para repassar o conhecimento, isso porque cada aluno apresenta um rendimento próprio, por vezes adiantado, outras não.

Outro fator que agrava o aprendizado são as dúvidas, pois muitas vezes o aluno, com receio de ser ridicularizado pelos colegas, não expõe incertezas, que acabam por passar em branco.

O aprendizado de um ambiente de programação não é uma tarefa trivial, pois engloba muitos detalhes. O ambiente Delphi não é diferente; muitos alunos têm dificuldades para aprendê-lo.

Pensando nisso, resolveu-se implementar um Tutorial Inteligente para Delphi, para tentar minimizar os problemas encontrados pelos estudantes.

Porém, por se tratar de um protótipo, nem todas as funções do Delphi serão explicadas, pois o objetivo é demonstrar a viabilidade de se implementar tal ferramenta para auxílio no aprendizado.

O protótipo apresentará as seguintes limitações:

- na quantidade de componentes utilizados, procurando-se utilizar os mais comuns;
- quanto à quantidade de eventos possíveis de serem selecionados, foi dada atenção aos mais utilizados pelos componentes em questão;
- as propriedades dos componentes não são explicadas, pois podem ser personalizadas conforme o desejo do usuário.

5.1.2. Levantamento de Requisitos

O levantamento de requisitos será representado através da Modelagem de Objetos utilizando algumas representações da *Unified Modeling Language* (UML) [FUR98], tais como o Diagrama de Classes, o Diagrama de Casos de Uso e o Diagrama de Seqüência.

Para o desenvolvimento do protótipo utilizou-se um microcomputador Intel PENTIUM® 200MHz, com 64MB de memória RAM, sistema operacional Windows95© e Delphi 3.0©. O espaço necessário para o armazenamento do protótipo em disco (HD) é de aproximadamente 3MB.

5.1.2.1 Casos de Uso

São identificados 3 casos de uso (cenários) para o protótipo, relacionados a seguir.

5.1.2.1.1 Caso 1 – Cadastrar Exercício

Neste caso, o ator Professor informa o conhecimento a respeito de um exercício (figura 5.2).

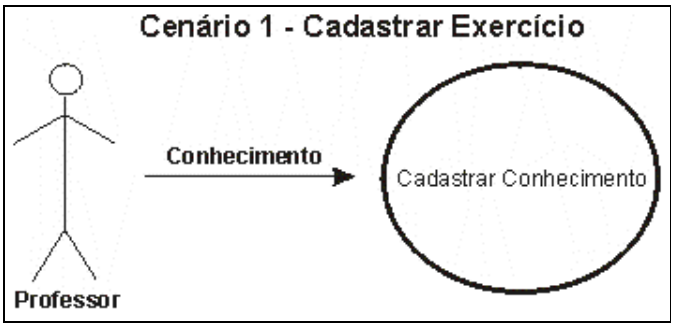

Figura 5.2: Caso de Uso 1 - Cadastrar Exercício.

5.1.2.1.2 Caso 2 – Aprender Exercício

Neste caso (figura 5.3), o ator aluno informa seus dados ao sistema, e recebe do mesmo o conhecimento do professor, dividido em passos.

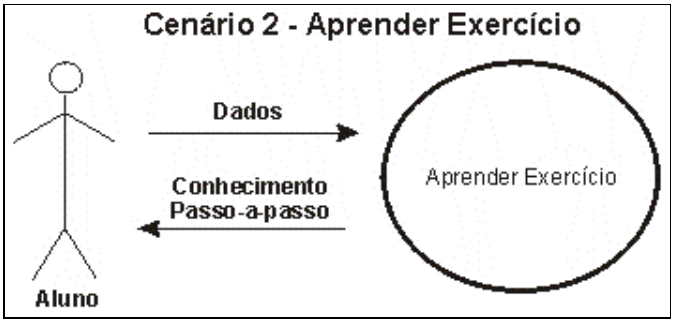

Figura 5.3: Caso de Uso 2 - Aprender Exercício.

5.1.2.1.3 Caso 3 – Corrigir Exercício

A figura 5.4 representa o caso de uso 3, no qual o ator Aluno identifica-se ao sistema e recebe do mesmo o resultado da correção do exercício. O ator Delphi informa o exercício realizado pelo aluno (o exercício em questão é o arquivo .PAS gerado na resolução do problema). O professor recebe do sistema a mesma relação de erros enviada ao aluno, que é utilizada para a avaliação mesmo.

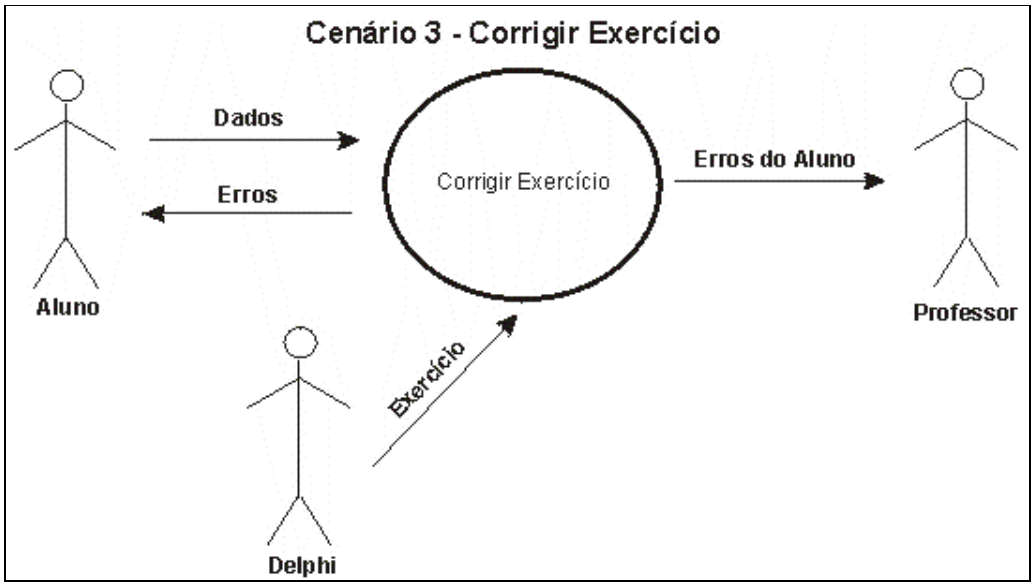

Figura 5.4: Caso de Uso 3 - Corrigir Exercício.

5.1.2.2 Diagrama de Classes

As Classes identificadas no sistema são:

- **Domínio**: esta Classe é responsável pela manutenção do conhecimento do professor, ou seja, cadastra e altera o conhecimento que o professor domina;
- **Aprendiz**: esta Classe controla a manutenção dos alunos cadastrados no sistema, ou seja, os dados de cada aluno e os acessos feitos pelo mesmo;
- **Tutor:** esta Classe é responsável pela associação entre as Classes Domínio e Aprendiz, controlando a troca de dados entre as mesmas. Através dela serão sistematizados os dados do Domínio e do Aprendiz para serem apresentados aos alunos;
- **Componente:** esta Classe é responsável por manter a quantidade de componentes inseridos em cada exercício;
- **Eventos:** esta Classe mantém os eventos associados a cada componente do exercício;
- **Itens:** esta Classe é responsável por manter os detalhes de cada componente do exercício;
- **AprenderExercício:** esta Classe mantém os erros e acertos que os alunos cometeram ao efetuar os exercício.

A figura 5.5 mostra o Diagrama de Classes.

Os detalhamentos dos atributos das Classes podem ser observados no Anexo I.

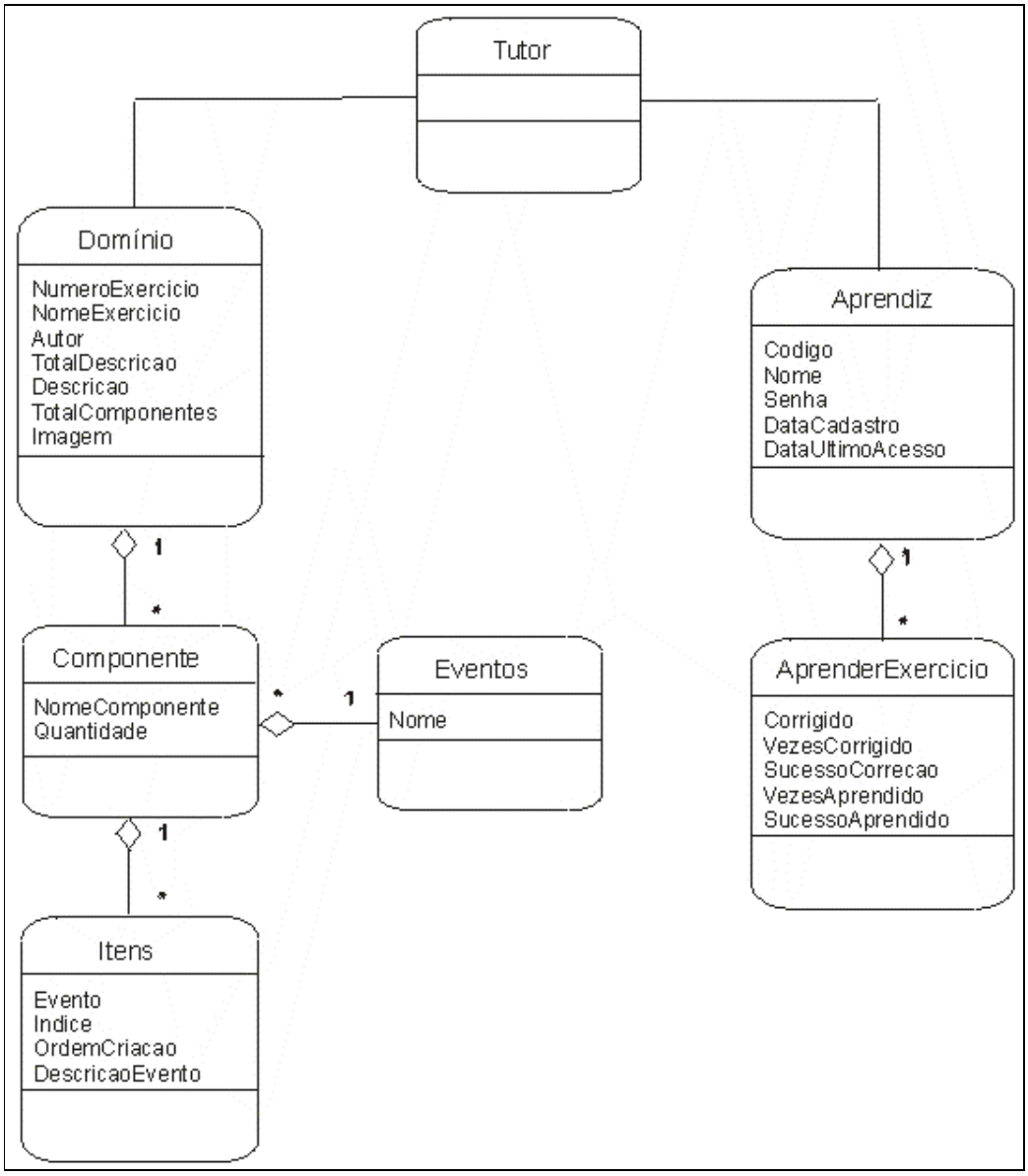

Figura 5.5: Diagrama de Classes.

Os Módulos do STI serão representados no protótipo por quatro agentes, cada qual composto pelo agrupamento das Classes definidas na figura 5.5.

- **Agente Domínio:** este agente agirá diretamente com o professor, sendo responsável pela criação da Base de Conhecimento, contendo a descrição de um problema e os passos a serem seguidos pelo aluno para tentar resolvê-lo (representado pelas Classes: Domínio, Componente, Eventos e Itens).

- **Agente Aprendiz:** depois que o aluno fizer o exercício utilizando os dados da Base de Conhecimento, este agente será responsável pela correção do mesmo, comparando o conhecimento do professor (Base de Conhecimento) com o conhecimento do aluno (Exercício). É importante ressaltar que o exercício criado pelo aluno é desenvolvido no Ambiente Delphi, e lido por este agente, que o compara com o conhecimento do professor (este Agente é composto pelas Classes: Aprendiz e AprenderExercício).
- **Agente Interface:** este agente será o responsável pelo gerenciamento da comunicação entre os demais Agentes, permitindo a interação com o usuário.
- **Agente Tutor:** uma vez cadastrado um conhecimento, este agente será responsável pela sistematização do mesmo para ser transmitido ao aluno (Classe Tutor).

Cada agente será responsável por um módulo do STI. A figura 5.6 apresenta os módulos do STI explicados no capítulos 2 e uma relação com os agentes.

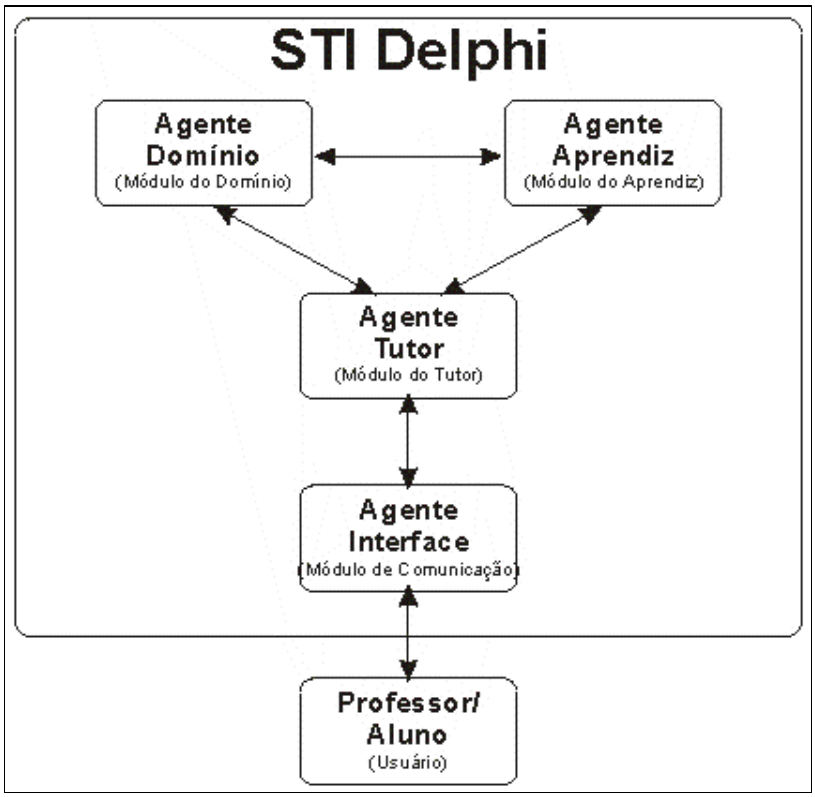

Figura 5.6: STI Delphi vs. módulos de um STI.

5.1.2.3 Diagramas de Seqüência

O Professor repassará o conhecimento para o Agente Domínio, que armazenará este conhecimento em uma Base de Conhecimento. O Agente Tutor irá interagir com o aluno, repassando a ele o conhecimento do professor, passo-a-passo.

Após isso, é a vez do Agente Aprendiz entrar em ação, comparando o conhecimento do professor (Base de Conhecimento) com o conhecimento do aluno (Exercício), reportando os erros encontrados e sugerindo ações para o aluno corrigir o exercício realizado.

Será utilizado o Agente Interface para facilitar a interação com o usuário, através da gerência dos outros agentes.

Os Diagramas de Seqüência, representam as trocas de mensagens entre as Classes. No Protótipo, são identificados 3 Diagramas de Seqüência (um para cada Caso de Uso), descritos a seguir.

5.1.2.3.1 Diagrama de Seqüência 1 – Cadastrar Exercício

O professor informa à Interface o conhecimento que ele detém, e a Interface repassa o conhecimento para o Agente Domínio que o armazena e finaliza a aplicação (figura 5.7).

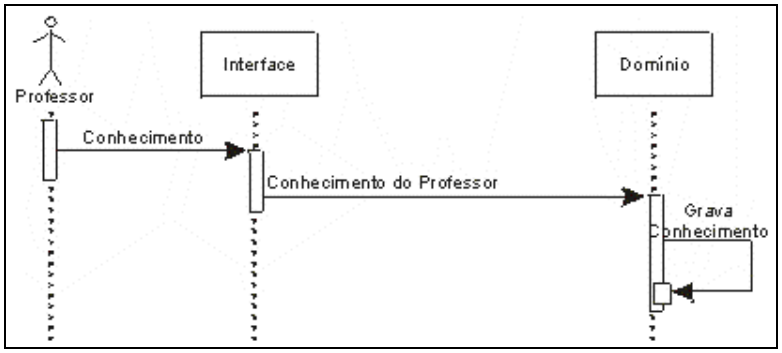

Figura 5.7: Diagrama de Seqüência – Cadastrar Exercício.

5.1.2.3.2 Diagrama de Seqüência 2 – Aprender Exercício

O aluno informa seus dados à Interface, que por sua vez repassa esses dados para o Agente Aprendiz. O Agente Domínio informa à Interface os conhecimentos disponíveis. O aluno seleciona o conhecimento que deseja aprender e o Agente Domínio repassa este conhecimento para o Agente Tutor, que o sistematiza em passos para serem repassados ao

aluno. Finalizado o aprendizado, o Agente Aprendiz armazena as informações sobre o aluno e o conhecimento por ele aprendido. A figura 5.8 mostra este Diagrama de Seqüência.

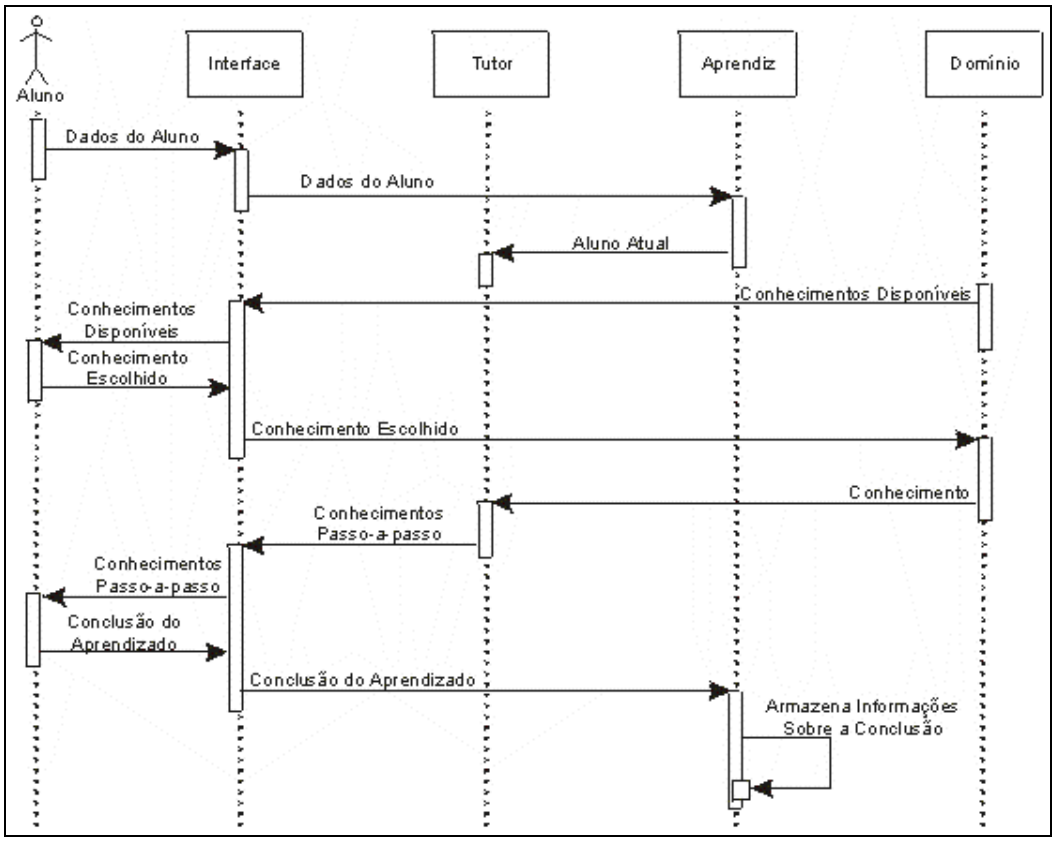

Figura 5.8: Diagrama de Seqüência – Aprender Exercício.

5.1.2.3.3 Diagrama de Seqüência 3 – Corrigir Exercício

O aluno informa seus dados à Interface que repassa ao Agente Aprendiz. O Agente Domínio informa os conhecimentos disponíveis para o aluno. Após a escolha do conhecimento, o ator Delphi informa ao Agente Domínio o exercício realizado pelo aluno. O Agente Domínio corrige o exercício e informa ao Agente Aprendiz os erros que o mesmo cometeu. O Agente Aprendiz armazena estes erros e fornece ao aluno e ao professor um relatório destes erros. A figura 5.9 mostra este Diagrama de Seqüência.

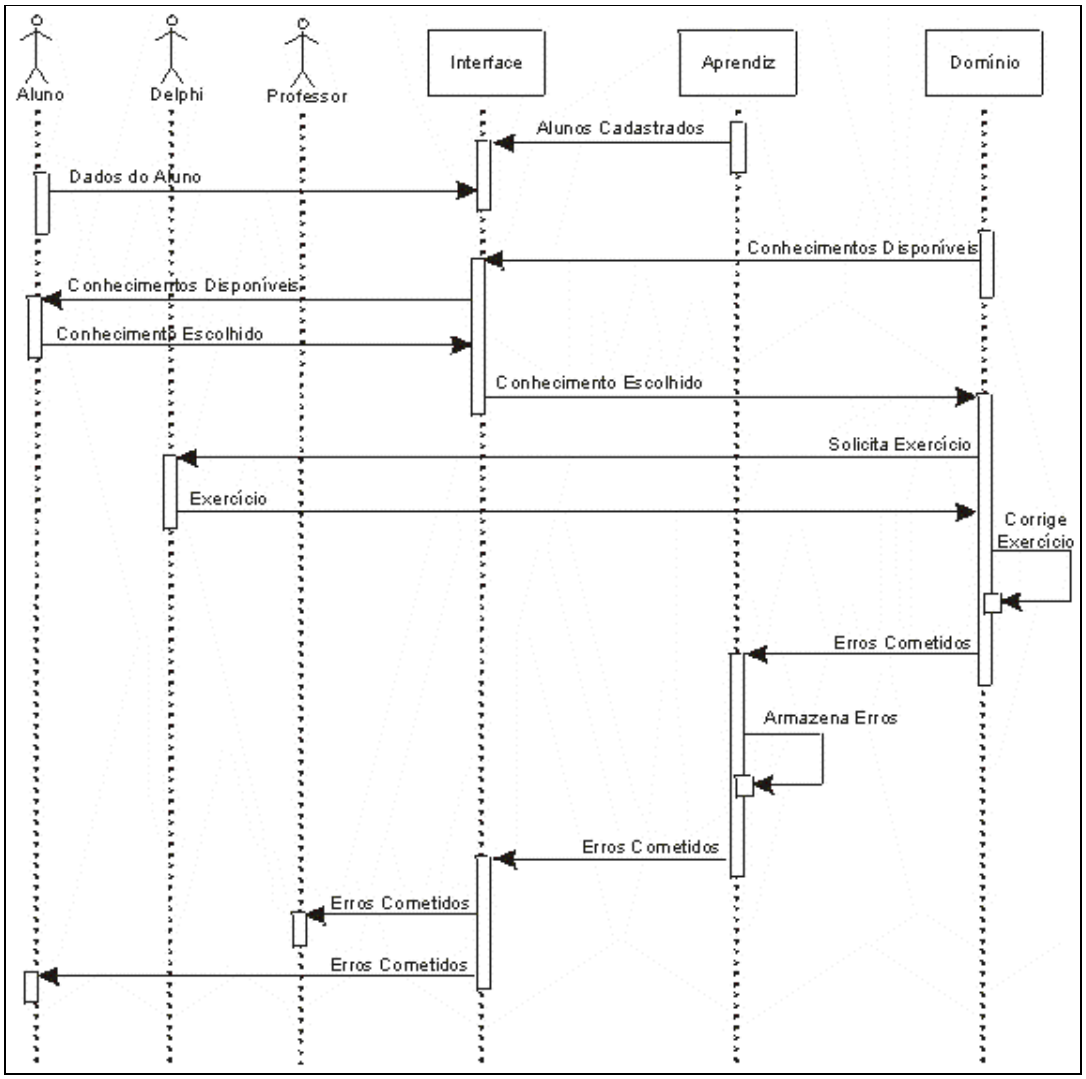

Figura 5.9: Diagrama de Seqüência – Corrigir Exercício.

5.1.3. Desenvolvimento do Protótipo

Neste tópico será apresentado o funcionamento de cada agente individualmente com suas características.

5.1.3.1 Agente Interface

Este agente é responsável pelo gerenciamento dos demais agentes do protótipo. Ele controla o uso dos agentes impedindo que mais de um execute ao mesmo tempo. Este agente, quando em funcionamento, sempre está visível na Barra de Tarefas.

A figura 5.10 mostra o Agente Interface em execução. Para executar qualquer outro Agente, basta clicar no botão correspondente.

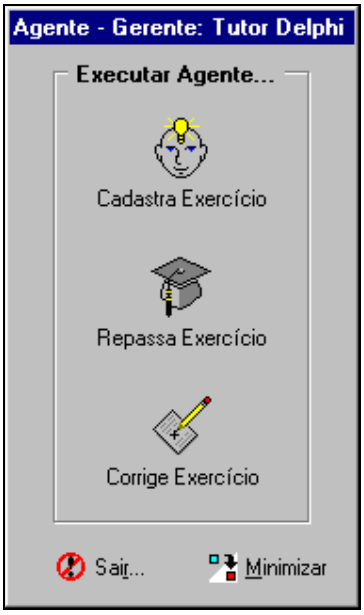

Figura 5.10: Agente Interface.

Este agente apresenta as seguintes características: Habilidade Social, Reatividade, Atividade Contínua, Estático, Benevolência, Racionalidade, Colaboração e Tempo de Vida passageiro.

5.1.3.2 Agente Domínio

Este agente cria uma Base de Conhecimento contendo o conhecimento do professor para um determinado problema.

Os dados informados nos passos do Agente Cadastra, formam várias listas que ao final são gravadas na Base de Conhecimento.

Os sete passos necessários para a criação da Base de Conhecimento são descritos a seguir:

1) **Informações do Exercício:**

   Nesta etapa (figura 5.11) são informados o nome do exercício, o autor do mesmo e uma descrição, indicando o que o exercício irá fazer e quais os principais componentes, por exemplo. O botão "Abrir" permite que um conhecimento previamente cadastrado, seja aberto para uma atualização, por exemplo.

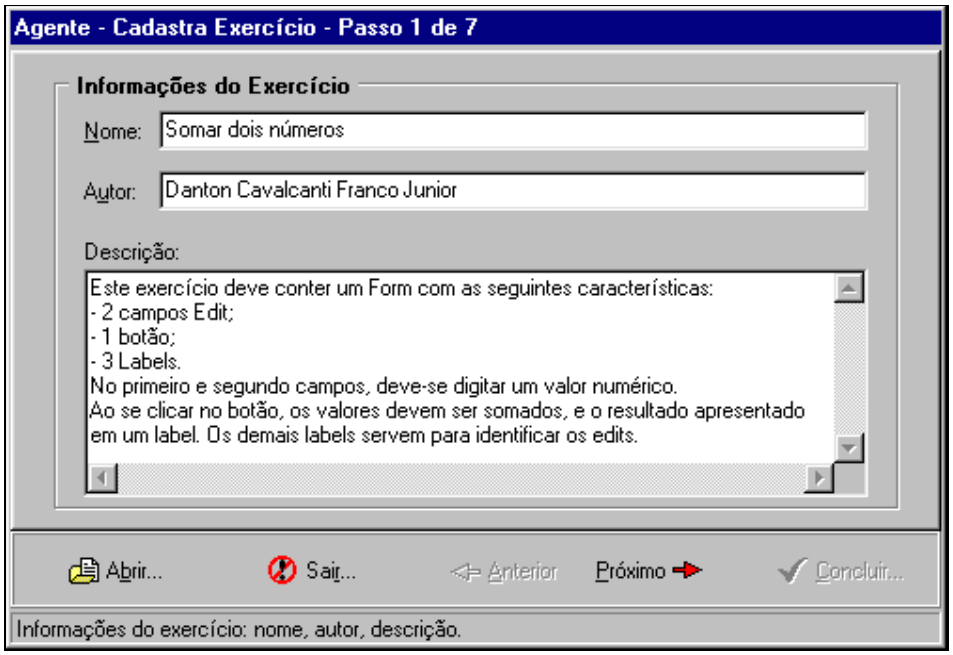

Figura 5.11: Passo 1 – Informações do Exercício.

2) Componentes do Exercício:

Neste passo (figura 5.12) devem ser informados quais componentes do Delphi o exercício irá ter, acrescentando-os à lista "Selecionados". Para avançar ao próximo passo, é necessário selecionar pelo menos um componente.

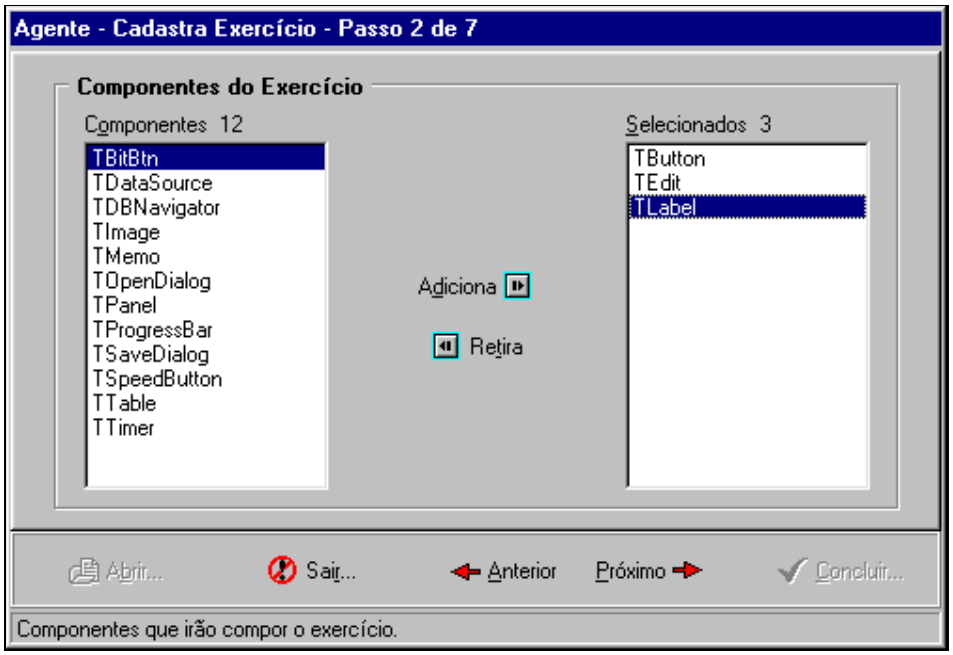

Figura 5.12: Passo 2 – Componentes do Exercício.

3) Quantidade de Cada Componente:

Aqui (figura 5.13), devem ser informadas as quantidades de cada componente do exercício. Deve-se selecionar o componente a partir da lista "Componentes Selecionados", e informar sua quantidade através da caixa "Quantidade".

Figura 5.13: Passo 3 – Quantidade de Cada Componente.

4) Ordem de Criação e Eventos dos Componentes:

Neste passo (figura 5.14), deve-se definir os eventos dos componentes, marcando a caixa referente ao evento que deve ser implementado. Definir também a ordem em que cada componente será criado, no momento de se explicar para o aluno. Observe que o nome dos componentes são automaticamente gerados, utilizando a mesma nomenclatura do Delphi.

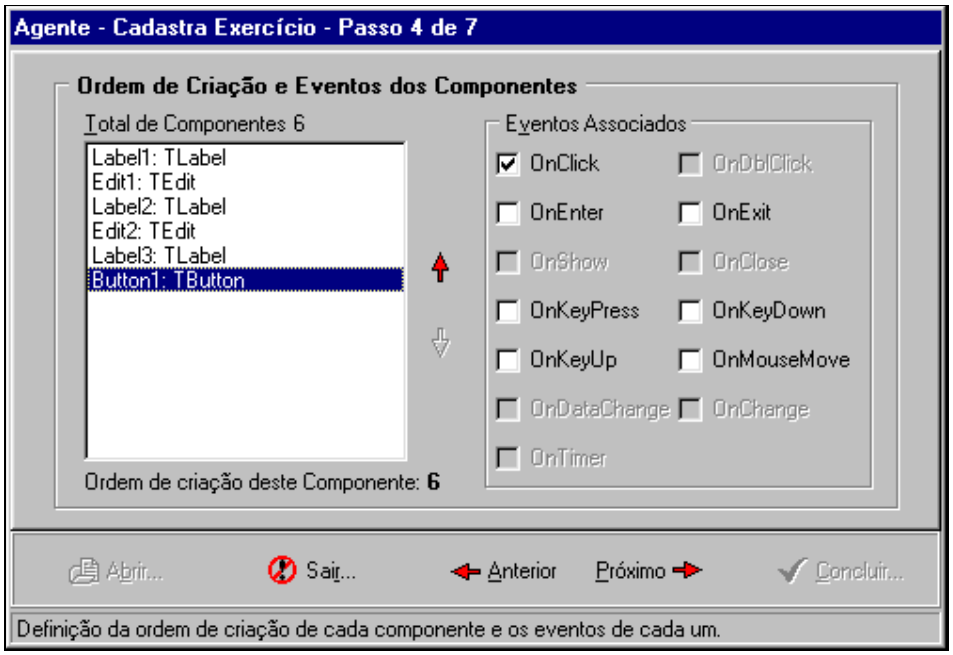

Figura 5.14: Passo 4 – Ordem de Criação e Eventos dos Componentes.

5) Descrição dos Eventos dos Componentes:

Neste item (figura 5.15), deve-se fazer um comentário e/ou descrição de um algoritmo, especificando o que o evento irá realizar quando o mesmo ocorrer.

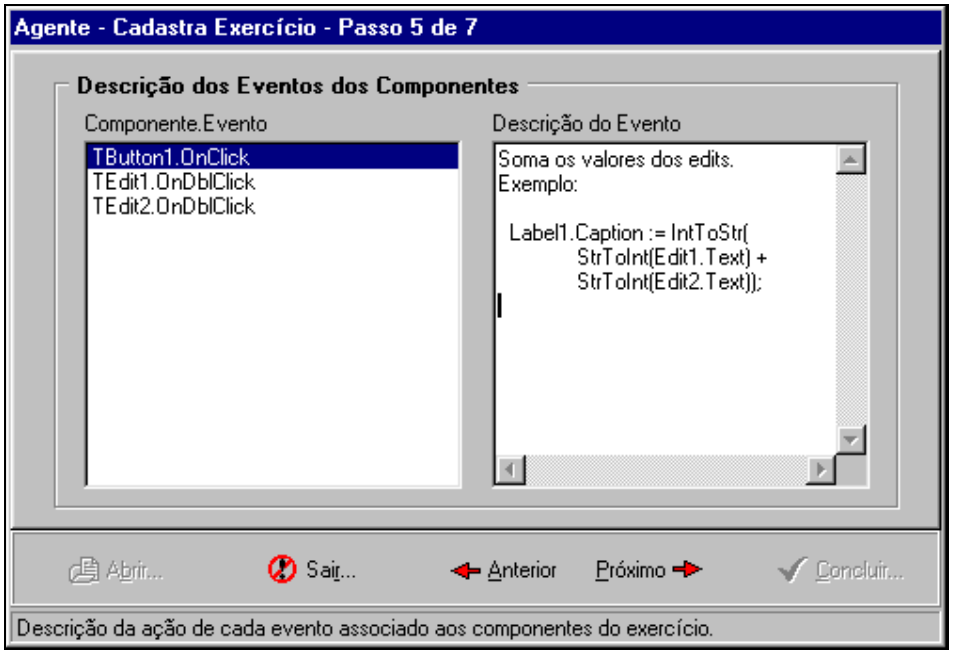

Figura 5.15: Passo 5 – Descrição dos Eventos dos Componentes.

6) Tela de Exemplo (figura):

Se existir uma aplicação exemplo, pode-se colocar neste passo (figura 5.16) uma figura (captura) da tela desta aplicação.

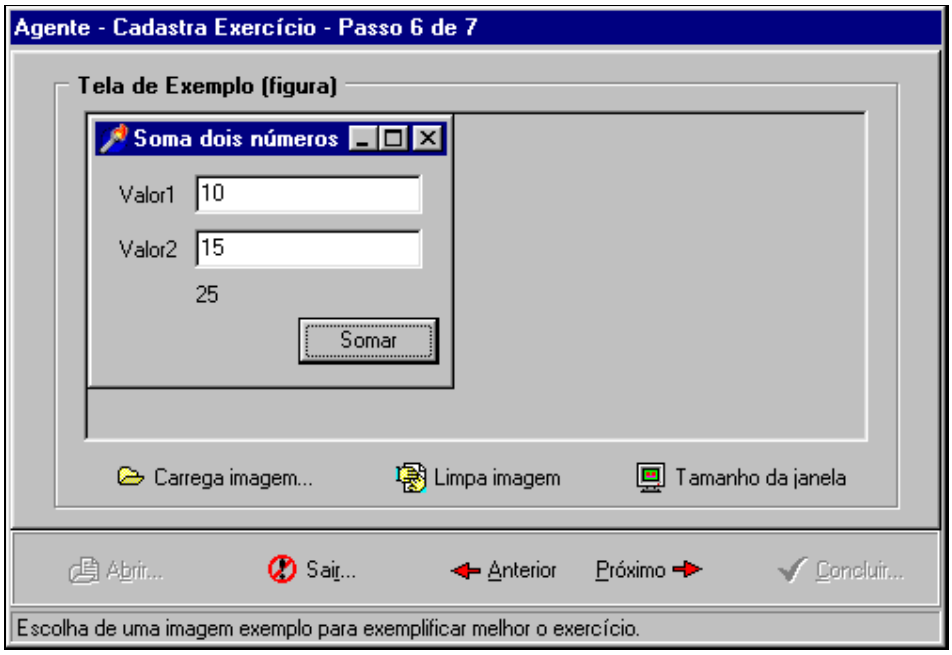

Figura 5.16: Passo 6 – Tela de Exemplo (figura).

7) Resumo do Exercício:

Para finalizar, um resumo de todo o exercício criado será mostrado neste passo (figura 5.17). Se tudo estiver correto, deve-se clicar no botão "Concluir", para salvar o conhecimento e terminar a execução do Agente. Caso alguma coisa não esteja correta, pode-se voltar aos passos anteriores e modificar o exercício.

O Botão "Concluir" cria uma instância da Classe Domínio e repassa os parâmetros necessários para a gravação do exercício. É importante ressaltar que as informações são as listas geradas durante o cadastramento das informações, mais alguns componentes de controle.

No Anexo II pode ser observado o código fonte do método Cadastrar.

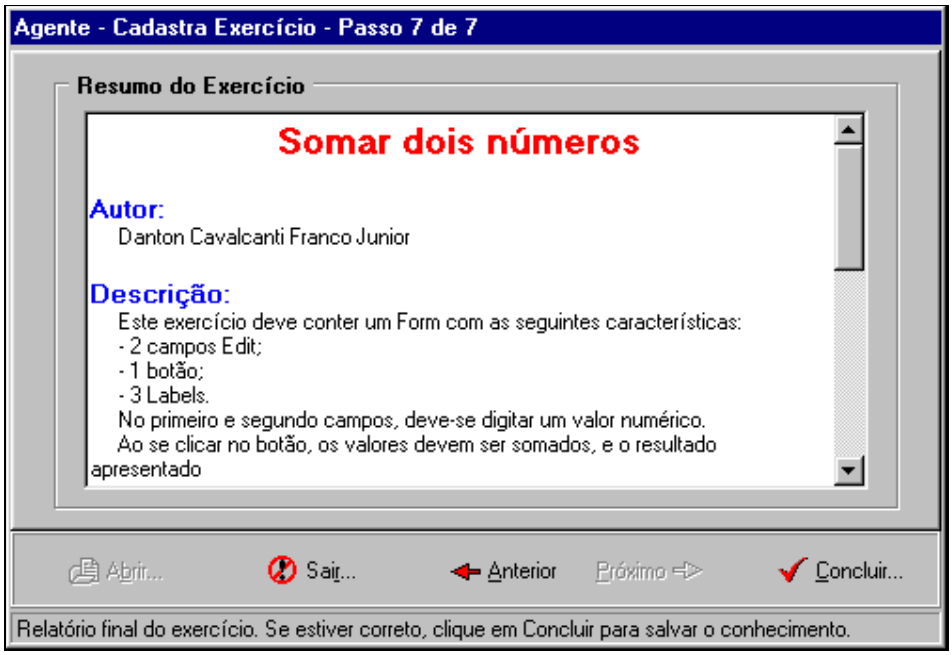

Figura 5.17: Passo 7 – Resumo do Exercício.

Este agente apresenta as seguintes características: Habilidade Social, Reatividade, Atividade Contínua, Estático, Benevolência, Racionalidade, Colaboração e Tempo de Vida passageiro.

5.1.3.3 Agente Tutor

Este Agente lê os dados da Base de Conhecimento e repassa ao aluno, passo-a-passo.

O funcionamento deste agente é análogo ao Agente Domínio, porém o número de passos realizados depende da complexidade do exercício.

Os passos básicos do agente são:

1) Aluno:

Nesta etapa (figura 5.18) deve-se selecionar o aluno que irá fazer o exercício. Se não existir alunos cadastrados, pode-se fazê-lo, clicando sobre o botão “Novo”. É importante ressaltar que o aluno só parte para o próximo passo se informar a senha correta, quando solicitada.

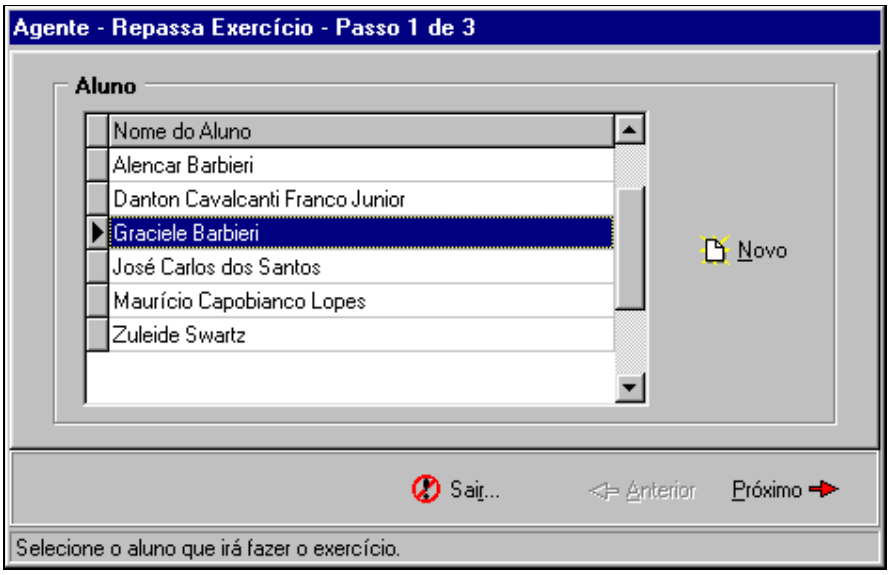

Figura 5.18: Passo 1 – Aluno.

2) Conhecimento a Ser Repassado:

Neste passo (figura 5.19) deve-se selecionar o exercício que o aluno deseja realizar.

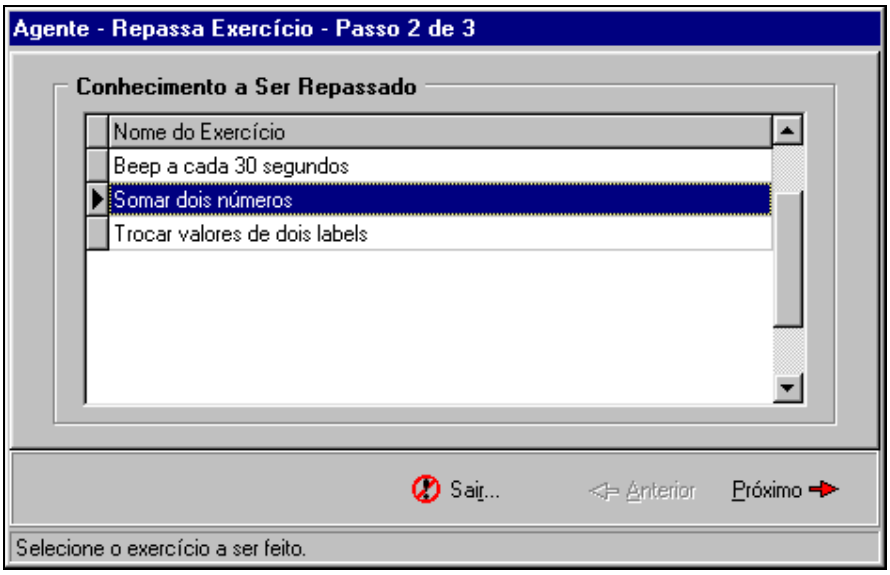

Figura 5.19: Passo 2 – Conhecimento a Ser Repassado.

A partir do passo 3 (figura 5.20), o agente opera em "modo exercício", ou seja, cria sub-passos, denominados Passos do Exercício, para exemplificar o conhecimento do professor. Esta quantidade de passos varia conforme a complexidade do exercício proposto.

Para criar os passos do exercício, o Agente quebra o conhecimento do professor em partes menores, sendo que cada componente que possui um evento contendo uma descrição

de exemplo, torna-se um passo novo. É importante ressaltar que seis passos são sempre mostrados, pois constituem a introdução do exercício.

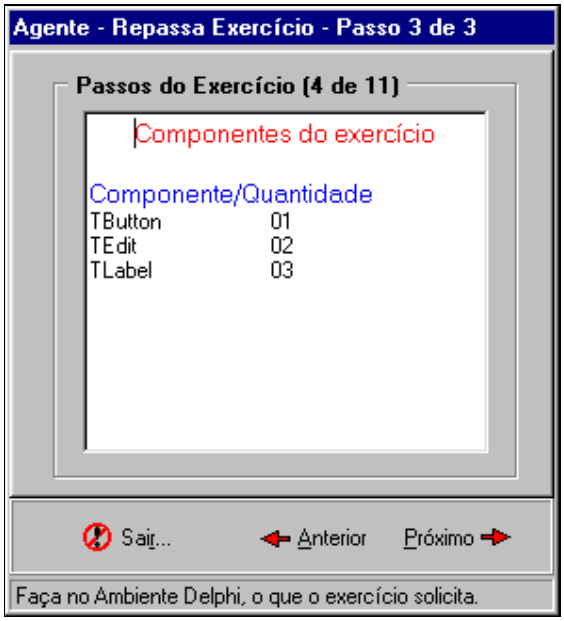

Figura 5.20: Passo 3 – Passos do Exercício.

Este agente apresenta as seguintes características: Habilidade Social, Reatividade, Atividade Contínua, Estático, Benevolência, Racionalidade, Colaboração e Tempo de Vida passageiro.

5.1.3.4 Agente Aprendiz

Da mesma forma que os demais agentes passo-a-passo, este agente possui passos que devem ser seguidos pelo aluno para que o mesmo verifique o exercício realizado.

Os Passos 1 e 2 deste agente são idênticos ao do Agente Tutor, constituindo-se da identificação do aluno. Porém ao invés de criar um novo aluno no passo 1, o aluno tem neste agente a possibilidade de verificar seu histórico, no qual tem a possibilidade de verificar os erros que já cometeu (figura 5.21) na realização de todos os exercícios.

Os erros são mantidos pelo Módulo Aprendiz, que atualiza-os sempre que o aluno voltar a repeti-los. Caso o erro seja novo, ele será inserido no cadastro de erros cometidos pelo aluno.

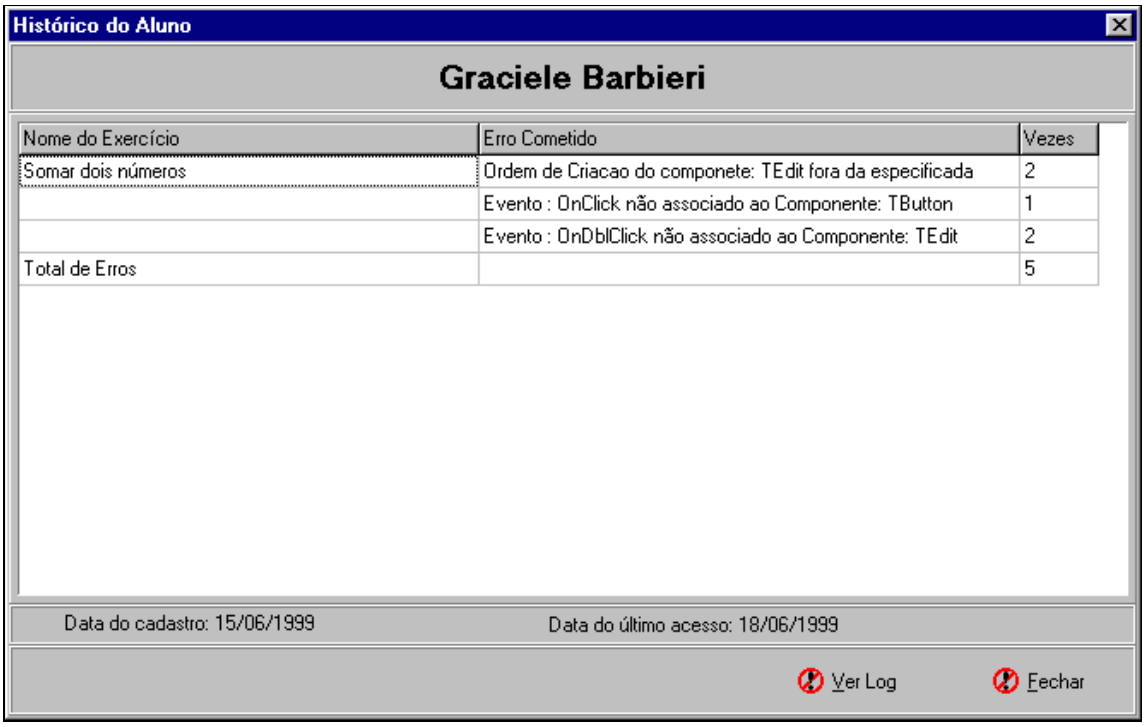

Figura 5.21: Histórico do Aluno.

Os demais passos são relacionados a seguir:

1) **Local do Exercício Realizado Pelo Aluno:**

Nesta etapa (figura 5.22), o aluno deve selecionar o local onde está salvo o exercício que o mesmo realizou, para que o sistema possa corrigi-lo. É importante ressaltar que somente arquivos com a extensão .PAS (arquivos que contém o código fonte dos projetos realizados em Delphi) serão mostrados.

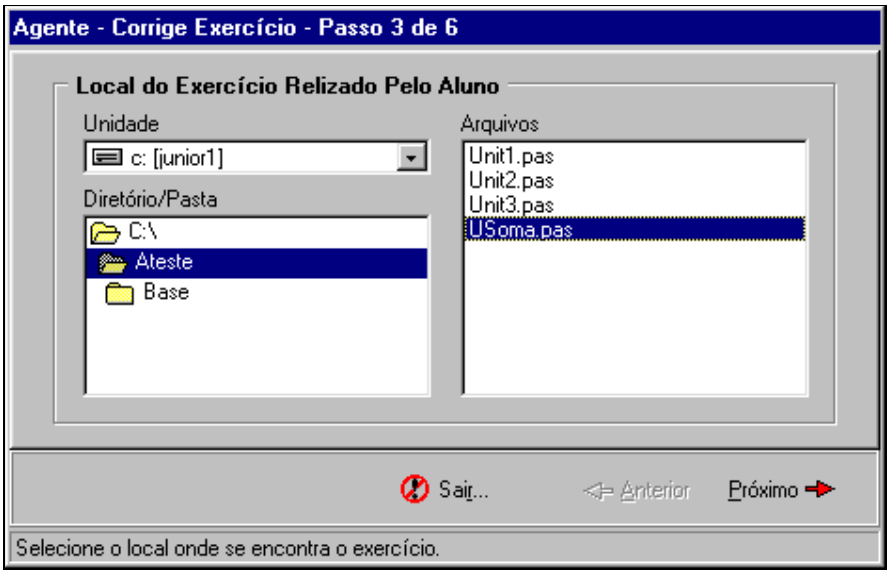

Figura 5.22: Passo 3 – Local do Exercício Realizado Pelo Aluno.

2) Visualização do Arquivo Selecionado:

Nesta etapa (figura 5.23), o aluno tem a possibilidade de verificar se o exercício escolhido é o que deseja corrigir, verificando seu conteúdo, que é mostrado em uma janela.

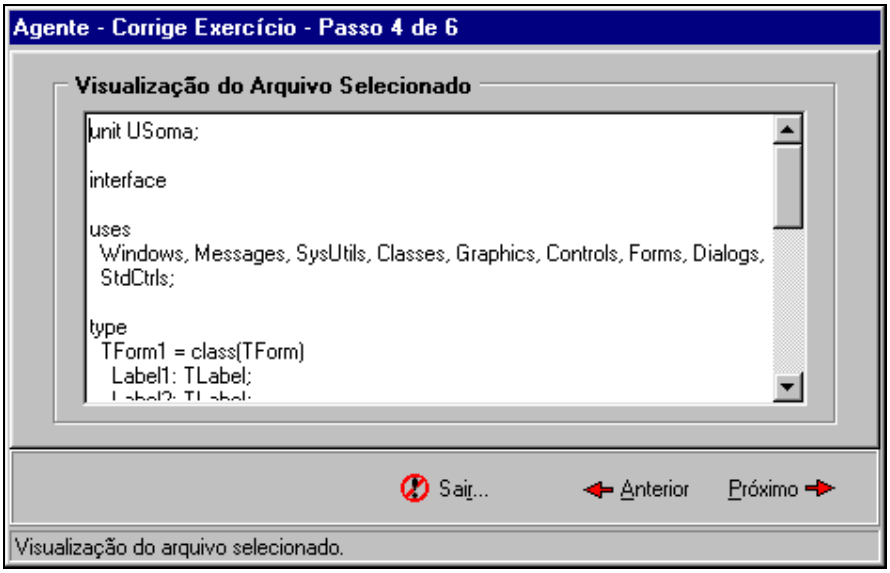

Figura 5.23: Passo 4 – Visualização do Arquivo Selecionado.

3) Correção do Exercício:

Após selecionado o exercício, o aluno tem a possibilidade de corrigi-lo. Ao clicar no botão "Corrigir" (figura 5.24), o agente inicia a correção do exercício, comparando o exercício realizado pelo aluno, com a Base de Conhecimento do exercício.

A correção é feita comparando-se as primeiras linhas do arquivo .PAS (da primeira linha até a palavra reservada *Private*). Para cada linha precedida pela palavra reservada *Procedure*, sabe-se que o complemento desta linha é um evento associado ao componente. Se a linha não apresentar esta palavra, sabe-se que é um componente. A ordem de leitura das linhas determina a ordem de criação dos componentes.

O código fonte deste procedimento pode ser observado no Anexo III.

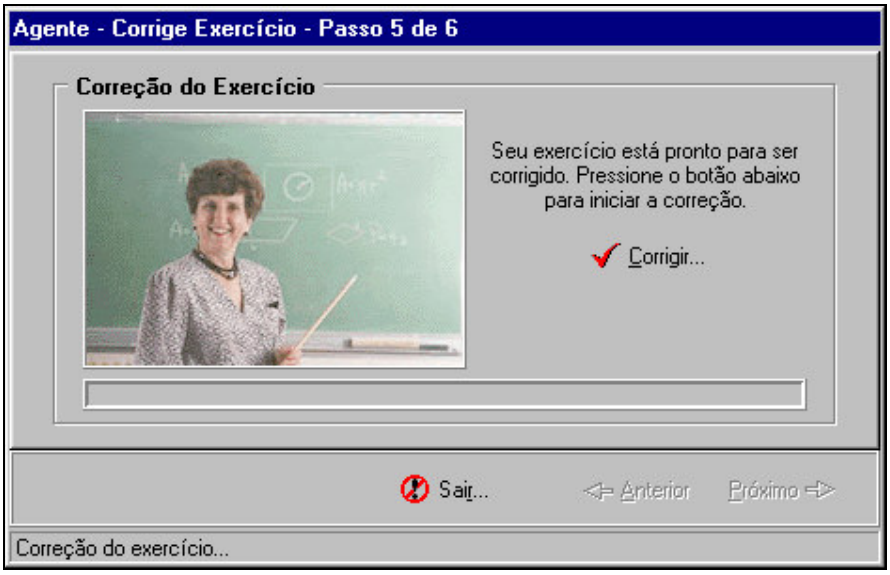

Figura 5.24: Passo 5 – Correção do Exercício.

4) Relatório de Erros Cometidos:

Por fim, o aluno tem um relatório com os erros cometidos na execução de seu exercício e as mensagens de atenção geradas durante a correção (figura 5.25). Estes erros são armazenados em uma base de dados para referência futura. Marcando-se a caixa “Gravar arquivo de Log em: ...”, pode-se gerar um arquivo texto contento o resumo de erros.

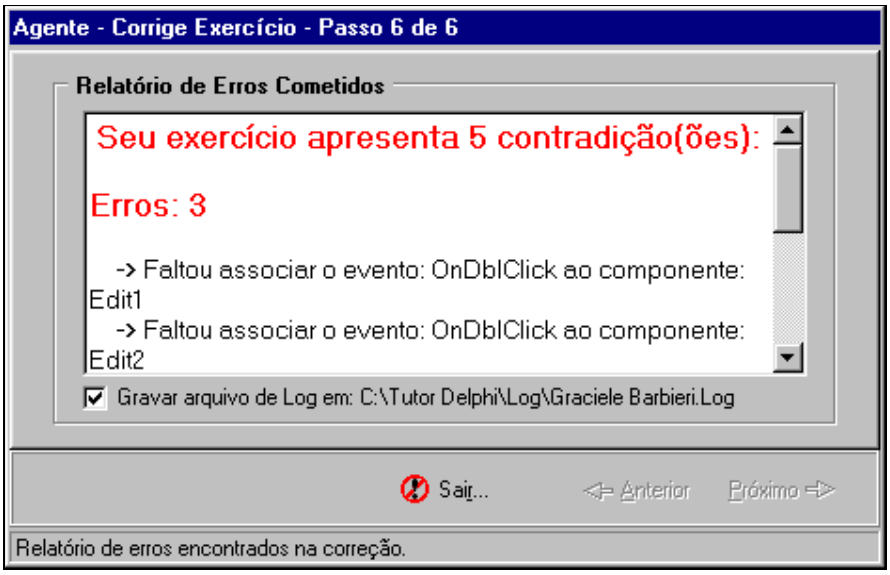

Figura 5.25: Passo 6 – Relatório de Erros Cometidos.

Este agente apresenta as seguintes características: Habilidade Social, Reatividade, Atividade Contínua, Estático, Benevolência, Racionalidade, Colaboração e Tempo de Vida passageiro.

5.1.4. Demonstração do Protótipo

O protótipo foi apresentado a alguns alunos do Curso de Ciências da Computação da FURB, tendo uma grande aceitação, inclusive com propostas de melhoras que foram implementadas dentro do possível.

5.1.5. Revisão do Protótipo

Esta etapa foi executada com base nas observações feitas pelos usuário es e pelo orientador durante a fase de demonstração do protótipo, sendo que todas as sugestões propostas para a melhora foram efetuadas, porém muita coisa ainda pode ser feita. As sugestões de melhoria serão apresentadas no capítulo final.

6. Conclusões e Sugestões

Este capítulo apresenta as conclusões, limitações e sugestões referentes ao trabalho desenvolvido.

6.1. Conclusões

O desenvolvimento de um STI demonstrou-se de grande viabilidade, pois possibilita ao professor dinamizar sua didática. Partindo deste princípio, o professor pode dedicar mais tempo ao aluno, deixando para o tutor a tarefa de repassar o conhecimento genérico.

Durante a construção do protótipo, foram utilizadas algumas etapas da Metodologia de Prototipação Fundamental, o que auxiliou muito o desenvolvimento do mesmo. A ferramenta de programação Delphi permitiu a realização do projeto de forma simples, pois possui grande variedade de recursos.

Os testes realizados durante o desenvolvimento do trabalho possibilitaram uma melhor avaliação de seu desempenho e da satisfação proporcionada aos usuários. As mudanças sugeridas pelos alunos foram implementadas dentro do possível, pois neste primeiro momento, o objetivo era demonstrar a viabilidade da implementação de um STI.

Os resultados obtidos com o protótipo podem ser considerados satisfatórios, pois atenderam aos objetivos propostos. Porém a possibilidade de se implementar melhoras no protótipo em futuras versões é válida, pois permite o aprimoramento de seu funcionamento.

6.2. Limitações

O protótipo desenvolvido apresenta as seguintes limitações:

a) número de componentes limitado;
b) utilização de eventos comuns, deixando de lado os menos utilizados;
c) Agente Tutor, não apresenta uma metodologia de ensino, que possa ser considerada atrativa para aluno;

d) o Agente Aprendiz não corrige a lógica do algoritmo feito pelo aluno, apenas se os componentes estão presentes e os eventos associados.

6.3. Sugestões

Sugere-se que o Agente – Repassa Conhecimento, pule os passos onde o aluno já demonstra conhecimento, tornando-se desta forma mais atrativo e inteligente.

Sugere-se também, a implementação de agentes móveis, ou seja, desenvolver agentes que possam trabalhar em rede, proporcionando desta forma uma interação melhor entre professor-aluno, viabilizando cada vez mais a comunicação entre ambos.

Em relação aos Sistemas Tutores Inteligentes, é interessante que outros trabalhos venham a ser desenvolvidos nesta área, pois a mesma é nova, possui grande potencial de crescimento, e dispõe de pouco material referencial.

ANEXO I

Este anexo representa os atributos das Classes definidas no capítulo 5.

Classe Domínio		
Campo	**Tipo**	**Descrição**
NumeroExercicio	Integer	Contém o índice (chave) que identifica o exercício
NomeExercicio	String[80]	Contém o nome do exercício
Autor	String	Contém o nome do autor do exercício
TotalDescrição	Integer	Contém a quantidade de linhas definida para a descrição do exercício
Descrição	String	Contém a descrição do exercício
TotalComponentes	Integer	Contém o número total de componentes
Imagem	String	Contém o nome do arquivo da imagem exemplo associada ao conhecimento

Tabela I.1: Estrutura da Classe Domínio.

Classe Aprendiz		
Campo	**Tipo**	**Descrição**
Codigo	Integer	Contém o índice (chave) que identifica o aluno
Nome	String[50]	Armazena o nome do aluno
Senha	String[15]	Armazena a senha do aluno criptografada
DataCadastro	Date	Armazena a data de cadastramento do aluno no sistema
DataUltimoAcesso	Date	Armazena a data em que o aluno acossou o sistema pela última vez

Tabela I.2: Estrutura da Classe Aprendiz.

Classe Componente		
Campo	**Tipo**	**Descrição**
NomeComponente	String	Contém o nome do componente
Quantidade	Integer	Contém a quantidade do componente

Tabela I.3: Estrutura da Classe Componente.

Classe AprenderExercício		
Campo	**Tipo**	**Descrição**
Corrigido	Boolean	Indica se o exercício já foi corrigido ou não
VezesCorrigido	Integer	Contém o número de vezes que o exercício já foi corrigido
SucessoCorrecao	Boolean	Indica se o exercício foi corrigido com sucesso ou não
VezesAprendido	Integer	Armazena a quantidade de vezes que o aluno realizou o exercício selecionado
SucessoAprendido	Boolean	Indica se o aluno realizou o aprendizado do exercício até o fim

Tabela I.4: Estrutura da Classe AprenderExercício.

Classe Itens		
Campo	**Tipo**	**Descrição**
Evento	String	Contém o nome do evento
Indice	Integer	Contém a quantidade de linhas identificando os eventos
OrdemCriacao	Integer	Armazena a ordem de criação dos componentes
DescricaoEvento	String	Armazena a descrição do evento

Tabela I.5: Estrutura da Classe Itens.

Classe Eventos		
Campo	**Tipo**	**Descrição**
Nome	String	Contém o nome do Evento

Tabela I.6: Estrutura da Classe Eventos.

ANEXO II

Este anexo representa o código fonte do método Cadastrar, da Classe Domínio.

```
function TDominio.Cadastrar(Nome : String; Passos : Integer; Anterior : TSpeedButton;
                            Mensagem : TPanel; Autor : String; Descricao : TMemo;
                            ComponentesSelecionados : TListBox;
                            LElementos : TStringList; LOrdemCriacao : TStringList;
                            LEventos : TStringList; LDescricao : TStringList;
                            LimpaImagem : TSpeedButton; FiguraExemplo : TImage;
                            Conhecimento : TTable) : Boolean;
// Gera o arquivo da Base de Conhecimento do exercício e finaliza o Agente
var
  Apoio               : String;
  Linha,
  Indice              : Integer;
  ArquivoConhecimento : TIniFile;

begin
  Result := False;
  MessageBeep(MB_ICONQUESTION);
  try
    if MessageDlg('Deseja gravar o conhecimento do exercício:' + #10 + Nome + '?',
                  mtConfirmation, [mbYES, mbNO], 0) = mrYes then
    begin
      ForceDirectories('C:\Tutor Delphi\Conhecimento');
      ForceDirectories('C:\Tutor Delphi\Conhecimento\Imagens');
      if FileExists('C:\Tutor Delphi\Conhecimento\' + Nome + '.CTD') then
      begin
        MessageBeep(MB_ICONQUESTION);
        if MessageDlg('Já existe um exercício com este nome na base de conhecimento.'
                      + #10 + 'Deseja sobrescrevê-lo?', mtConfirmation,
                      [mbYES, mbNO], 0) = mrNo then
        begin
          MessageBeep(MB_ICONEXCLAMATION);
          MessageDlg('Volte ao passo 1 e mude o nome do exercício.',
                     mtInformation, [mbOK], 0);
          MessageBeep(MB_ICONQUESTION);
          if MessageDlg('Deseja voltar ao passo 1 agora', mtConfirmation,
                        [mbYES, mbNO], 0) = mrYes then
            for Indice := Passos - 1 downto 1 do
              Anterior.Click;
          Result := False;
          Exit;
        end;
      end;
      Mensagem.Caption := 'Gravando Conhecimento...';
      Mensagem.Visible := True;
      Mensagem.Refresh;
      DeleteFile('C:\Tutor Delphi\Conhecimento\' + Nome + '.CTD');
      //Conhecimento Tutor Delphi
      ArquivoConhecimento := TIniFile.Create('C:\Tutor Delphi\Conhecimento\' +
                             Nome + '.CTD');
      ArquivoConhecimento.WriteString('Informações do Exercício', 'Nome', Nome);
      ArquivoConhecimento.WriteString('Informações do Exercício', 'Autor', Autor);
      ArquivoConhecimento.WriteInteger('Informações do Exercício', 'TotalDescrição',
Descricao.Lines.Count);
      for Linha := 0 to Descricao.Lines.Count - 1 do
        ArquivoConhecimento.WriteString('Informações do Exercício', 'Descrição' +
IntToStr(Linha), Descricao.Lines.Strings[Linha]);
      ArquivoConhecimento.WriteInteger('Componentes do Exercício', 'TotalComponentes',
ComponentesSelecionados.Items.Count);
      for Linha := 0 to ComponentesSelecionados.Items.Count - 1 do
      begin
        ArquivoConhecimento.WriteString('Componentes do Exercício', 'Componente' +
IntToStr(Linha), ComponentesSelecionados.Items.Strings[Linha]);
        ArquivoConhecimento.WriteString('Componentes do Exercício', 'Quantidade' +
IntToStr(Linha),
LElementos.Strings[LElementos.IndexOf(ComponentesSelecionados.Items.Strings[Linha]) + 1]);
      end;
      ArquivoConhecimento.WriteInteger('Ordem de Criação dos Componentes',
'TotalOrdemCriação', LOrdemCriacao.Count);
      for Linha := 0 to LOrdemCriacao.Count - 1 do
        ArquivoConhecimento.WriteString('Ordem de Criação dos Componentes', 'Ordem' +
IntToStr(Linha), LOrdemCriacao.Strings[Linha]);
```

```
      Linha := 0;
      repeat
        ArquivoConhecimento.WriteString('Eventos dos Componentes', LEventos.Strings[Linha],
LEventos.Strings[Linha + 1]);
        Inc(Linha, 2);
      until Linha >= LEventos.Count;
      Indice := 0;
      ArquivoConhecimento.WriteInteger('Descrição dos Eventos', 'TotalDescriçãoEventos',
Indice);
      Linha := 0;
      if LDescricao.Count <> 0 then
      begin
        repeat
          if LDescricao.Strings[Linha + 1] <> '' then
          begin
            ArquivoConhecimento.WriteString('Descrição dos Eventos', 'Evento' +
IntToStr(Indice), LDescricao.Strings[Linha]);
            Inc(Indice);
          end;
          Inc(Linha, 2);
        until Linha >= LDescricao.Count;
        ArquivoConhecimento.WriteInteger('Descrição dos Eventos', 'TotalDescriçãoEventos',
Indice);
        Linha := 0;
        repeat
          Apoio := LDescricao.Strings[Linha + 1];
          if Apoio <> '' then
          begin
            Indice := 0;
            ArquivoConhecimento.WriteInteger('Descrição dos Eventos',
LDescricao.Strings[Linha] + 'TotalDescrição', 0);
            repeat
              ArquivoConhecimento.WriteString('Descrição dos Eventos',
LDescricao.Strings[Linha] + 'L' + IntToStr(Indice), Copy(Apoio, 1, Pos(#13, Apoio) - 1));
              Delete(Apoio, 1, Pos(#13, Apoio));
              Inc(Indice);
            until Length(Apoio) <= 1;
            ArquivoConhecimento.WriteInteger('Descrição dos Eventos',
LDescricao.Strings[Linha] + 'TotalDescrição', Indice);
          end;
          Inc(Linha, 2);
        until Linha >= LDescricao.Count;
      end;
      if LimpaImagem.Enabled then
      begin
        try
          // Grava a figura exemplo para posterior referência
          FiguraExemplo.Picture.SaveToFile('C:\Tutor Delphi\Conhecimento\Imagens\' + Nome +
'.ITD'); //Imagem Tutor Delphi
          ArquivoConhecimento.WriteString('Imagem Exemplo', 'Imagem', 'C:\Tutor
Delphi\Conhecimento\Imagens\' + Nome + '.ITD');
        except
          ArquivoConhecimento.WriteString('Imagem Exemplo', 'Imagem', 'Erro!');
          MessageBeep(MB_ICONASTERISK);
          MessageDlg('Ocorreu um erro ao salvar a imagem exemplo' + #10 + #10 +
                     'Verifique e tente novamente.', mtError, [mbOK], 0);
        end;
      end
      else
        ArquivoConhecimento.WriteString('Imagem Exemplo', 'Imagem', 'Nenhuma');
      ArquivoConhecimento.Free;
      Conhecimento.IndexFieldNames := 'NumeroExercicio';
      if not Conhecimento.Locate('NomeExercicio', Nome, [loCaseInsensitive]) then
      begin
        Conhecimento.Last;
        Indice := Conhecimento.FieldByName('NumeroExercicio').AsInteger + 1;
        Conhecimento.Insert;
        Conhecimento.FieldByName('NumeroExercicio').AsInteger := Indice;
        Conhecimento.FieldByName('NomeExercicio').AsString := Nome;
        Conhecimento.Post;
      end;
      Mensagem.Visible := False;
      MessageBeep(MB_ICONEXCLAMATION);
      MessageDlg('Sucesso ao salvar o conhecimento do exercício!' + #10 + 'Finalizando...',
                 mtInformation, [mbOK], 0);
      Result := True;
    end;
  except
```

```
    MessageBeep(MB_ICONASTERISK);
    MessageDlg('Ocorreu erro um gravar a base de conhecimento.' + #10 +
               'Seu disco deve estar cheio, ou o arquivo de conhecimento tem mais de 64Kb.'
               + #10 + #10 + 'Verifique estas informações e tente novamente.', mtError,
[mbOK], 0);
    Result := False;
  end;
end;
```

ANEXO III

Este anexo representa o código fonte do procedimento de Correção do exercício.

```
procedure TfrPrincipal.sbCorrigirClick(Sender: TObject);
var
  LinhaE,
  LinhaI,
  Indice,
  Elementos,
  ContaComponentes : LongInt;
  Evento           : Byte;
  Associado,
  EventosAtual,
  ComponenteAtual  : String;
  Fim,
  Achou            : Boolean;
  Dominio          : TDominio;
  Aprendiz         : TAprendiz;

begin
  lbCorrigindo.Visible := True;
  sbCorrigir.Enabled := False;
  pbCorrecao.Position := 0;
  pbCorrecao.Max := 9;
  Fim := False;
  pbCorrecao.Position := pbCorrecao.Position + 1;
  for LinhaE := 0 to reArquivo.Lines.Count - 1 do
  begin
    if Pos('= class(TForm)', reArquivo.Lines.Strings[LinhaE]) > 0 then
    begin
      pbCorrecao.Position := pbCorrecao.Position + 1;
      for LinhaI := LinhaE + 1 to reArquivo.Lines.Count - 1 do
        if Pos('private', reArquivo.Lines.Strings[LinhaI]) > 0 then
        begin
          pbCorrecao.Position := pbCorrecao.Position + 1;
          Fim := True;
          Break;
        end
        else
          if Pos('procedure', reArquivo.Lines.Strings[LinhaI]) > 0 then

ListaEventosExe.Add(frPrincipal.RetiraEspacos(Copy(reArquivo.Lines.Strings[LinhaI], 1,
Pos('(', reArquivo.Lines.Strings[LinhaI]))))
          else

ListaComponentesExe.Add(frPrincipal.RetiraEspacos(reArquivo.Lines.Strings[LinhaI]));
    end;
    if Fim then
      Break;
  end;
  Dominio := TDominio.Create;
  if not Dominio.InformaConhecimentoReduzido(tbExercicios, ListaComponentes,
                                             pbCorrecao, ListaOrdemCriacao,
                                             ListaEventos) then
    Application.Terminate;
  Dominio.Free;
  // Verificação dos componentes e quantidades
  Aprendiz := TAprendiz.Create;
  Elementos := 0;
  repeat
    ContaComponentes := 0;
    for Indice := 0 to ListaComponentesExe.Count - 1 do
      if Pos(ListaComponentes.Strings[Elementos], ListaComponentesExe.Strings[Indice]) > 0
then
        Inc(ContaComponentes);
    if ContaComponentes < StrToInt(ListaComponentes.Strings[Elementos + 1]) then
    begin
      ListaErros.Add('Componente: ' + ListaComponentes.Strings[Elementos] + ' com quantidade
inferior a solicitada. Esperado(s): ' + ListaComponentes.Strings[Elementos + 1] + ',
encontrado(s): ' + IntToStr(ContaComponentes));
      if not tbErrosAluno.FindKey([tbAlunos.FieldByName('CodigoAluno').AsInteger,
                                   tbExercicios.FieldByName('NumeroExercicio').AsInteger,
                                   Erro_QuantidadeInvalida,
```

```
UFuncoes.PosicaoArray(Copy(ListaComponentes.Strings[Elementos], 2,
Length(ListaComponentes.Strings[Elementos]))),
                                0]) then
        Aprendiz.InsereErroNovo(tbErrosAluno, tbAlunos, tbExercicios,
                                Erro_QuantidadeInvalida,

UFuncoes.PosicaoArray(Copy(ListaComponentes.Strings[Elementos], 2,
Length(ListaComponentes.Strings[Elementos]))),
                                0)
      else
        Aprendiz.IncrementaErro(tbErrosAluno);
      tbErrosAluno.IndexName := 'CodAluExeErrCla';
    end;
    Inc(Elementos, 2);
  until Elementos > ListaComponentes.Count - 1;
  pbCorrecao.Position := pbCorrecao.Position + 1;
  // Verificação da ordem de criação
  for Indice := 0 to ListaOrdemCriacao.Count - 1 do
    for Elementos := 0 to ListaComponentesExe.Count - 1 do
      if ListaOrdemCriacao.Strings[Indice] = ListaComponentesExe.Strings[Elementos] then
        if Indice = Elementos then
          Break
        else
        begin
          ListaAtencao.Add('O componente: ' + ListaOrdemCriacao.Strings[Indice] + ', está fora
da ordem de criação especificada');
          if not tbErrosAluno.FindKey([tbAlunos.FieldByName('CodigoAluno').AsInteger,
                                       tbExercicios.FieldByName('NumeroExercicio').AsInteger,
                                       Erro_OrdemCriacaoInvalida,

UFuncoes.PosicaoArray(Copy(ListaOrdemCriacao.Strings[Indice], Pos(':',
ListaOrdemCriacao.Strings[Indice]) + 3, Length(ListaOrdemCriacao.Strings[Indice]))),
                                       0]) then
            Aprendiz.InsereErroNovo(tbErrosAluno, tbAlunos, tbExercicios,
                                    Erro_OrdemCriacaoInvalida,

UFuncoes.PosicaoArray(Copy(ListaOrdemCriacao.Strings[Indice], Pos(':',
ListaOrdemCriacao.Strings[Indice]) + 3, Length(ListaOrdemCriacao.Strings[Indice]))),
                                    0)
          else
            Aprendiz.IncrementaErro(tbErrosAluno);
          tbErrosAluno.IndexName := 'CodAluExeErrCla';
        end;
  pbCorrecao.Position := pbCorrecao.Position + 1;
  // Verificação dos eventos associados aos componentes
  Elementos := 0;
  repeat
    ComponenteAtual := Copy(ListaEventos.Strings[Elementos], 2,
Length(ListaEventos.Strings[Elementos]));
    EventosAtual := ListaEventos.Strings[Elementos + 1];
    for Evento := 1 to 13 do
      if EventosAtual[Evento] = '1' then
      begin
        Associado := ComponenteAtual + Copy(UFuncoes.DescricaoEvento(Evento), 3,
Length(UFuncoes.DescricaoEvento(Evento)));
        Achou := False;
        for Indice := 0 to ListaEventosExe.Count - 1 do
          if Pos(Associado, ListaEventosExe.Strings[Indice]) > 0 then
          begin
            Achou := True;
            Break;
          end;
        if not Achou then
        begin
          ListaErros.Add('Faltou associar o evento: ' + UFuncoes.DescricaoEvento(Evento) + '
ao componente: ' + ComponenteAtual);
          if not tbErrosAluno.FindKey([tbAlunos.FieldByName('CodigoAluno').AsInteger,
                                       tbExercicios.FieldByName('NumeroExercicio').AsInteger,
                                       Erro_EventoNaoAssociado,
                                       Evento,
                                       UFuncoes.PosicaoArray(Copy(ComponenteAtual, 1,
Length(ComponenteAtual) - 1))]) then
            Aprendiz.InsereErroNovo(tbErrosAluno, tbAlunos, tbExercicios,
                                    Erro_EventoNaoAssociado,
                                    Evento, UFuncoes.PosicaoArray(Copy(ComponenteAtual, 1,
Length(ComponenteAtual) - 1)))
          else
```

```
          Aprendiz.IncrementaErro(tbErrosAluno);
        tbErrosAluno.IndexName := 'CodAluExeErrCla';
      end;
    end;
  Inc(Elementos, 2);
 until Elementos > ListaEventos.Count - 1;
 Aprendiz.Free;
 pbCorrecao.Position := pbCorrecao.Position + 1;
 sbProximo.Enabled := True;
 lbCorrigindo.Visible := False;
 MessageBeep(MB_ICONEXCLAMATION);
 MessageDlg('Exercício corrigido!', mtInformation, [mbOK], 0);
end;
```

Referências Bibliográficas

[ALV97] ALVARES, Luis Otavio; SICHMAN, Jaime Simão. Introdução aos sistemas multiagentes. XVII Congresso da Sociedade Brasileira de Computação. **Anais...** Brasília, 1997. p. 1.

[ANS97] ANSELMO, Fernando. **Delphi 2.0: desvendando o caminho das pedras**. Goiânia : Editora Gráfica Terra Ltda., 1997.

[CAN98] CANTU, Marco. **Dominando o Delphi 3: a bíblia**. São Paulo : Makron Books, 1998.

[DAV95] DAVIS, Harold. **Delphi ferramentas poderosas**. São Paulo : Berkeley Brasil Editora, 1995.

[DUN96] DUNTEMANN, Jeff; MISCHAEL, Jim; TAYLOR, Don. **Delphi kit do explorador**. São Paulo : Berkeley do Brasil, 1996.

[FUR98] FURLAN, José Davi. **Modelagem de objetos através da UML: análise e desenho orientados a objeto**. São Paulo : Makron Books, 1998.

[GIR95] GIRAFFA, Lucia Maria Martins; OLIVEIRA, Marco Andrei Kichalowsky de. Leonardo – máquina de auxílio à resolução de problemas. VI SBIE - Simpósio Brasileiro de informática na Educação, 1995. **Anais...** Florianópolis: UFSC, 1995

[GIR97] _____. **Seleção e adoção de estratégias de ensino em sistemas tutores inteligentes**. Porto Alegre : Universidade Federal do Rio Grande do Sul, 1997. Tese de doutorado.

[HÜB95] HÜBNER, Jomi Fred. **Migração em sistemas multiagentes abertos**. Porto Alegre : Universidade Federal do Rio Grande do Sul, 1995. Dissertação de Mestrado.

[HUH97] HUHNS, Michael N.; SINGH, Munindar P. **Readings in agents**. San Francisco, California : Morgan Kaufmann Publishers, Inc., 1997.

[LAS97] LASHKARI, Yezdi; METRAL, Max; MAES, Pattie. Collaboratives interface agents. Readings in agents. **Anais...** San Francisco, California : Morgan Kaufmann Publishers, Inc., 1997. p. 111.

[LIU97] LIU, Jyi-Shane; SYCARA, Katia P. Multiagent coordination in tightly coupled task scheduling. Readings in agents. **Anais...** San Francisco, California : Morgan Kaufmann Publishers, Inc., 1997. p. 164.

[MEL90] MELENDEZ Filho, Rubem. **Prototipação de sistemas de informações: fundamentos, técnicas e metodologia**. Rio de Janeiro : Livros Técnicos e Científicos, 1990.

[MES98] MESQUITA, Lellis Marçal; SILVA, Wagner Teixeira da. Ferramenta de autoria para o domínio de conhecimento de STI baseado em Frames – FASTI. IX Simpósio Brasileiro de Informática na Educação – SBIE'98. **Anais...** Fortaleza, 1998.

[POR97] PORTO, Paulo R. Prestes; PALAZZO, Luiz A. Moro; CASTILHO, José M. Volkmer. Agentes de informação inteligentes. I Oficina de Inteligência Artificial. **Anais...** Universidade Católica de Pelotas, 1997. p. 81.

[RAM97] RAMOS, Alexandre C. B.; JESUS, Hamilton R. de; TEIXEIRA, Daniel B. Congresso de Informática e Telecomunicações - Petrobras e USP**. Anais...** INFTEL'97, 1997.

[RAM98] _____; RAMOS, Cláudia Aline B. Sistema baseado em regras para apoio ao ensino de geometria no segundo grau. Conferência Internacional de Educação à Distância. **Anais...** ICFEF'98, 1998.

[ROS97] ROSENSCHEIN, Jeffrey S.; ZLOTKIN, Gilad. Designing conventions for automated negotiation. Readings in agents. **Anais...** San Francisco, California : Morgan Kaufmann Publishers, Inc., 1997. p. 353.

[RUS95] RUSSELL, Stuart; NORVIG, Peter. **Artificial intelligence: a modern approach.** New Jersey : Prentice Hall Series in Artificial Intelligence, 1995.

[TAM97] TAMBE, Milind; JOHNSON, Lewis; SHEN, Wei-Min. Adaptative agent tracking in real-world multi-agent domains: a preliminary report. Readings in agents. **Anais...** San Francisco, California : Morgan Kaufmann Publishers, Inc., 1997. p. 504.

[TAN97] TAN, Ming. Multi-agent reinforcement learning: independent vs. cooperative agents. Readings in agents. **Anais...** San Francisco, California : Morgan Kaufmann Publishers, Inc., 1997. p. 487.

[THO97] THOMPSON, J. W. Leky. **Software agents mail list**. 1997. Endereço eletrônico: http://www.ee.mcgill.ca/~agent_faq.html.

[TOM98] TOMEI, Elaine. **Aquisição de conhecimento através de imagens.** 1998. Endereço eletrônico: http://www.comp.ita.cta.br/~elainet/paper.html.

[VAS97] VASCONCELOS, Maria Helena Schneid. Aplicação de inteligência artificial em sistemas de ensino inteligentes. I Oficina de Inteligência Artificial. **Anais...** Universidade Católica de Pelotas, 1997. p. 25.

[VAV98] VAVASSORI, Fabiane Barreto; GAUTHIER, Fernando Álvaro Ostuni. Proposta de ferramentas e agentes inteligentes para um ambiente de ensino/aprendizagem na Web. IX Simpósio Brasileiro de Informática na Educação – SBIE'98. **Anais...** Fortaleza, 1998.

[VIC90] VICCARI, Rosa Maria. **Um tutor inteligente para a programação em lógica – idealização, projeto e desenvolvimento**. Coimbra : Universidade de Coimbra, 1990. Tese de Doutorado.

[VIC96] _____; GIRAFFA, Lucia Maria Martins. Sistemas tutores inteligentes: abordagem tradicional x abordagem de agentes. Brazilian Symposium on Artificial Intelligence. **Anais...** Curitiba, 1996.

[VIC98] _____. Sistemas tutores inteligentes. VI Escola Regional de Informática. **Anais...** Regional Sul, 1998. p. 37.

[WAR96] WARNER, Scott L.; GOLDSMAN, Paul. **Delphi 2 em exemplos**. São Paulo : Makron Books, 1996.

[WEI97] WEISS, Gerhard. Learning to coordinate action in multiagent systems. Readings in agents. **Anais...** San Francisco, California : Morgan Kaufmann Publishers, Inc., 1997. p. 481.

[WIL97] WILHELM, Pedro Paulo Hugo. **Uma nova perspectiva de aproveitamento e uso dos jogos de empresas**. Florianópolis : Universidade Federal de Santa Catarina, 1997. Tese de doutorado.

[ZUA96] ZUANÁBAR, Delfa Mercedes Huatuco. **Integração das técnicas de aprendizagem acelerada nos Sistemas Tutores Inteligentes**. 1996. Endereço eletrônico: http://www.ita.cta.br/~delfa/paper.htm.