UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

SISTEMA TUTOR INTELIGENTE PARA O ENSINO DO GERENCIADOR DE ARMAZENAMENTO E ARQUIVOS

JEAN CARLOS SCHAPPO

BLUMENAU
2008

2008/1-18

JEAN CARLOS SCHAPPO

SISTEMA TUTOR INTELIGENTE PARA O ENSINO DO GERENCIADOR DE ARMAZENAMENTO E ARQUIVOS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Adilson Vahldick, Orientador

BLUMENAU
2008

2008/1-18

SISTEMA TUTOR INTELIGENTE PARA O ENSINO DO GERENCIADOR DE ARMAZENAMENTO E ARQUIVOS

Por

JEAN CARLOS SCHAPPO

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Adilson Vahldick, Especialista – Orientador, FURB
Membro:	Prof. Mauro Marcelo Mattos, Doutor – FURB
Membro:	Prof. André Luis Alice Raabe, Doutor – UNIVALI

Blumenau, dia 09 de julho de 2008

Dedico este trabalho a toda minha família e amigos, especialmente a meus pais, namorada e a meu orientador por acreditar no potencial deste trabalho.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

Aos meus pais, Brás Inácio Schappo e Erna Schappo que me ajudaram a custear os estudos e que sempre me incentivaram.

À minha namorada Samantha Luana Anhaya pela compreensão e apoio na elaboração deste trabalho.

Aos meus amigos, pelos empurrões e cobranças.

Ao meu orientador, Adilson Vahldick, por ter me incentivado e acreditado na conclusão deste trabalho.

Os bons livros fazem “sacar” para fora o que a pessoa tem de melhor dentro dela.

Lina Sotis Francesco Moratti

RESUMO

Este trabalho apresenta desenvolvimento de um Sistema Tutor Inteligente (STI) que utiliza lógica *fuzzy* em todos os seus módulos. Para sua experimentação foi desenvolvido um curso para o ensino dos componentes de gerenciamento de armazenamento e arquivos do Sistema Gerenciador de Bancos de Dados (SGBD). O STI apresenta o conteúdo no modelo SCORM, e nesse trabalho foi desenvolvido um Objeto de Aprendizagem (OA) que segue esse modelo. Nesse OA o aluno monta a estrutura de blocos e já acontece a correção. Os dados de correção são enviados ao STI seguindo estritamente o modelo SCORM. O OA foi desenvolvido como uma applet. Para o desenvolvimento do STI foram utilizadas a tecnologia Java Server Faces e banco de dados PostGreSQL. Para a execução dos pacotes SCORM foi utilizado o componente CELINE.

Palavras-chave: Lógica *fuzzy*. Sistema tutor inteligente. Gerenciador armazenamento e arquivos. SCORM.

ABSTRACT

This work presents the development of an Intelligent Tutor System (ITS) that uses fuzzy logic in all its modules. For their experiment was developed a course for learning the components of management of storage and archives of the Database Management System (DBMS). The ITS presents the content in SCORM model format, and that work was carried a Learning Object (LO). In this LO the student assembles the structure of the blocks and already is correct. The data are sent to the ITS following the structure of the SCORM model. The LO was developed like in applet. For the development of the ITS was used the Java Server Faces technology and the PostGreSQL DBMS. For the SCORM content execution was used the CELINE component.

Key-words: Fuzzy logic. Intelligent tutoring system. Storage and archiving manager. SCORM.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AJAX - *Asynchronous Javascript And XML*

API – *Application Programming Interface*

DOM – *Document Object Model*

E/S – Entrada ou Saída

HTML – *Hyper Text Markup Language*

JAR – Java *Archive*

JDK – Java *Development Kit*

JESS - *Java Expert System Shell*

LMS - *Learning Management System*

OA – Objeto de Aprendizagem

SCORM – *Sharable Content Object Reference Model*

SGBD – Sistema de Gerenciamento de Banco de Dados

SQL - *Structured Query Language*

STI – Sistema Tutor Inteligente

URL – *Uniform Resource Locator*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Os bancos de dados tornaram-se um componente essencial no cotidiano da sociedade moderna. E surge uma grande quantidade de dados que precisam ser armazenados fisicamente em mídias de maior capacidade e menor custo, tal que, sejam organizados de forma a ser recuperados posteriormente.

Conforme Elmasri e Navathe (2005, p. 4), banco de dados computadorizados podem ser criados e mantidos tanto por um grupo de aplicativos escritos especialmente para essa tarefa como por um Sistema Gerenciador de Banco de Dados (SGBD). E, portanto, o SGBD é um sistema de software de propósito geral que facilita os processos de definição, construção, manipulação e compartilhamento do banco de dados.

O módulo gerenciador de armazenamento e arquivos administra a alocação de espaço no disco magnético e as estruturas de dados usadas para representar estas informações. Elas são componentes de mais baixo nível de um SGBD.

A recuperação eficiente das informações gerenciadas por um SGBD está relacionada a forma pela qual foram projetadas as complexas estruturas da representação destes dados no banco de dados e estão divididas em três níveis: físico, que é o mais baixo nível de abstração e descreve como estes dados estão de fato armazenados; lógico, que descreve quais dados estão armazenados no banco de dados e quais os inter-relacionamento; e nível de visão, que é o mais alto nível descreve apenas parte do banco de dados (SILBERSCHATZ; KORTH; SUDARSHAN, 1999, p. 4).

O ensino sobre o funcionamento do gerenciador de armazenamento e arquivos requer motivação dos alunos, pois possui muitos conceitos que precisam ser praticados. Para Ausubel et al. (1980 apud FUJI; SILVEIRA, 2006, p. 3) o fator mais importante da aprendizagem é conhecer o que o aluno já sabe, evidenciando a importância da integração dos novos conteúdos à estrutura cognitiva existente do aprendiz.

Para isto um ambiente com capacidade de captar ao máximo as ações e necessidades particulares de cada aluno. O Sistema Tutor Inteligente (STI) tem como objetivo proporcionar um ensino individualizado, de acordo com as necessidades do estudante, tentando se aproximar a um professor e permitir ao estudante ter um ensino personalizado. Ao professor caberá tirar dúvidas não solucionadas pelo STI e apoiar o estudo (MOREIRA, 2007, p. 266).

Diante do exposto, neste trabalho será desenvolvido um STI que auxilie o aluno na execução de atividades sobre o módulo gerenciador de armazenamento e arquivo de um

SGBD. Essas atividades são selecionadas pelo ambiente conforme o desempenho do aluno. As interações do aluno com o STI são monitoradas para que possam ser disponibilizadas ao professor informações desse aprendizado. De posse deste conhecimento sobre o desempenho da turma a cada atividade ele poderá reavaliar suas práticas pedagógicas, assim como obter o desempenho individual de cada estudante. Para o monitoramento do aprendiz serão utilizadas regras de inferências *fuzzy*.

A lógica *fuzzy* (ou difusa) é a generalização da lógica booleana que admite valores lógicos intermediários entre a falsidade e a verdade e é baseada na teoria dos conjuntos *fuzzy*, onde um elemento pertença a um conjunto com certo grau de pertinência variando entre 0 e 1 e assim, as preposições não são mais somente verdadeiras ou falsas (MOREIRA, 2007, p. 273).

Com o desenvolvimento do sistema espera-se que seja possível automatizar a correção das atividades e provas, bem como acompanhar o desempenho individual do aluno, sabendo quem fez o que, quanto tempo levou para realizar a atividade, que atividades estão corretas e onde ocorreu o maior número de erros.

A utilização de STI proporciona um melhor desempenho e maior motivação ao estudante, pois ao captar o que o mesmo já aprendeu toma decisão de apresentar atividade com uma complexidade combatível ou em adicionar novos conceitos conforme a base de conceitos cadastrada.

E para definir qual é o desempenho e a melhor complexidade para atividade em um conceito de um determinado estudante, faz-se uso da lógica *fuzzy*. Pois a mesma permitir uma analise mais real dos dados ao captar diferentes graus de verdade, e modelar está informações em formas matemáticas. E assim, conforme a base de regras cadastradas é definido como o sistema irá agir.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um STI para o ensino do funcionamento dos componentes gerenciador de armazenamento e arquivos de um SGBD.

Os objetivos específicos do trabalho são:

a) disponibilizar um ambiente em que o aluno execute atividades de organização e modelagem de campos, registros, blocos;

b) desenvolver uma ferramenta para o professor montar os enunciados;
c) disponibilizar para o professor um ambiente para apresentar o desempenho da turma e dos alunos;
d) desenvolver o comportamento do ambiente através de sistema *fuzzy*.

1.2 ESTRUTURA DO TRABALHO

No próximo capitulo apresenta a fundamentação teórica necessária para o razoável entendimento das tecnologias e componentes empregadas no desenvolvimento deste trabalho.

O terceiro capítulo tem como foco o desenvolvimento do objeto de aprendizagem para exercício de gerenciamento de armazenamento e arquivos e do STI, descrevendo suas principais funcionalidades, bem como suas especificações e implementações.

O quarto capítulo apresenta as conclusões finais, mostrando os resultados obtidos, suas limitações e sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são explanados os fundamentos teóricos necessários para a compreensão do trabalho. Na última seção são descritos alguns trabalhos correlatos.

2.1 GERENCIADOR DE ARMAZENAMENTO E ARQUIVOS

Segundo Garcia-Molina, Ullman e Widom (2001, p. 22) o que distingue os SGBD de outros sistemas é a habilidade para lidar de forma eficiente com quantidade muito grande de dados. A melhor organização de dados oferece melhor suporte para manipulação eficiente dos mesmos.

“Os dados geralmente são armazenados na forma de registros. Cada registro consiste em uma coleção de valores ou itens relacionados, na qual cada valor é formado por um ou mais bytes e correspondem a um campo de dados do registro.” (ELMASRI; NAVATHE, 2005, p. 302).

Elmasri e Navathe (2005, p. 302) explicam que campos podem incluir tipos de dados numéricos (inteiro, inteiro longo ou ponto flutuante), cadeia de caracteres (de tamanho fixo ou variável), lógico (tendo apenas 0 e 1 ou verdadeiro e falso como valores) e, algumas vezes, tipos de dados especialmente codificados para data e hora. O número de bytes para cada tipo de dados é fixado para um determinado sistema. Números inteiros podem necessitar de 4 bytes, inteiros longos de 8 bytes, número real de 4 bytes, booleano de 1 byte, uma data 10 bytes (no formato AAAA-MM-DD) e uma cadeia de k caracteres de tamanho fixo de k bytes. A cadeia de caracteres de tamanho variável necessita de tantos bytes quantos forem os caracteres. E uma coleção de nomes de campos e seus respectivos tipos de dados constituem a definição do tipo do registro ou forma do registro.

“Os registros de um arquivo precisam ser alocados em blocos de disco porque um bloco é a unidade de transferência de dados entre o disco e a memória. (...)” (ELMASRI; NAVATHE, 2005, p. 304).

Conforme Garcia-Molina, Ullman e Widom (2001, p. 102) há uma questão que deve ser levada em conta quando se projeta o layout de um registro. Podem existir informações que devem ser mantidas no registro, mas que não correspondem ao valor de qualquer campo. Por

exemplo, pode ser necessário querer manter o comprimento do registro.

Outro exemplo acontece quando um ou mais campos de um registro têm comprimento variável, então é necessário que o registro contenha informações suficientes para permitir localizar qualquer campo. Uma estratégia simples mas efetiva é colocar todos os campos de comprimento fixo antes dos de comprimento variável. Em seguida, inserir no cabeçalho do registro, o comprimento do registro e ponteiro para o início de todos os campos de tamanho variado. Porém, se todos os campos de tamanho variado aparecem na mesma ordem, o primeiro não necessita de ponteiro, pois sabe-se que ele sempre vem imediatamente aos campos de registro fixo (GARCIA-MOLINA; ULLMAN; WIDOM, 2001, p. 118).

Os índices permitem localizar um registro sem necessitar examinar uma quantidade grande de dados. Uma estratégia comumente utilizada para construir índices é a Árvore B+, por esta permitir a pesquisa, inserção e a eliminação de registros com o uso de poucas operações de Entrada ou Saída (E/S) de disco por operações de arquivo (GARCIA-MOLINA; ULLMAN; WIDOM, 2001, p. 166).

Para melhoria da confiabilidade e desempenho utilizam-se vários discos. Essa técnica é conhecida como *Redundance Arrays of Inexpensive Disks* (RAID). O I de RAID atualmente significa *independent.* Para o problema de confiabilidade é introduzida a redundância, ou seja, para armazenar informações adicionais que não são normalmente utilizadas a técnica de espelhamento é a mais simples e consiste em duplicar cada disco. Outra técnica é a distribuição paralela de dados e consiste em distribuir os bits de cada byte pelos múltipos discos, chamada de distribuição paralela de bit. São propostas várias estratégias para prover redundância a baixo custo utilizando a idéia de espelhamento de disco combinada com “paridade” de bits. Estes esquemas possuem relação entre custo e desempenho e são classificados em seis níveis (SILBERSCHATZ; KORTH; SUDARSHAN, 1999, P. 300).

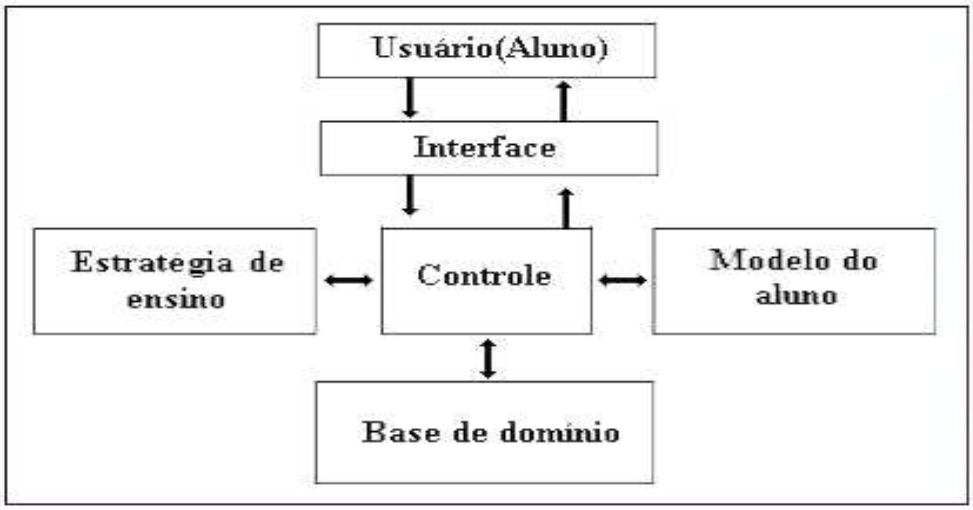
Garcia-Molina, Ullman e Widom (2001, p. 102) explica que RAID nível 0 (zero) refere-se a um conjunto de discos com distribuição paralela baseada em blocos, mas sem qualquer redundância (tal como redundância ou bits de paridade).

2.2 SISTEMA TUTOR INTELIGENTE (STI)

Segundo Hall e Wood (1990 apud LIMA; ROSATELLI, 2003, p. 05), os STI são uma composição de diversas disciplinas como psicologia, ciências cognitiva e inteligência

artificial.

A arquitetura clássica de um STI é composta basicamente por quatro módulos, como pode ser visto na Figura 1. Na base de domínio está o conhecimento armazenado do especialista na matéria a ser ensinada, portanto, o que deve ser transferido para o aluno. O módulo estratégico de ensino expõe de diferentes maneiras um assunto (domínio), tornando-o compreensível e interessante para o aluno. O módulo do aluno reúne informações a respeito do estudante, aspectos do conhecimento e do comportamento do estudante que tragam conseqüências para seu desempenho e aprendizagem. A interface é o canal de comunicação entre o STI e o estudante (MOREIRA, 2007, p. 265).



Fonte: Costa (2002, p. 1).

Figura 1 – Arquitetura clássica de um STI

Conforme Jonassen (1993 apud MOREIRA, 2007, p. 265) um STI deve passar em três testes antes de possuir um comportamento inteligente:

a) o conteúdo do tema ou especialista deve ser codificado de modo que o sistema possa acessar as informações, fazer inferências ou resolver problemas;
b) o sistema deve ser capaz de avaliar a aquisição deste conhecimento pelo estudante;
c) as estratégias tutoriais devem ser projetadas para reduzir a discrepância entre o conhecimento do especialista e o conhecimento do estudante.

Este comportamento é atingido por meio de uma arquitetura composta, basicamente por quatro módulos (domínio, pedagógico, estudante e interface com estudante), busca-se separar o domínio (matéria de ensino) das estratégias de ensino. E por meio desta arquitetura, vê-se que o estudante tem mais possibilidade de ter um ensino personalizado.

2.3 SISTEMA *FUZZY*

No mundo real são usadas palavras como: muito, pouco, grande, pequeno, freqüentemente, raramente, entre outros, para descrever situações. Essas situações não são nitidamente definidas e não podem ser precisamente descritas (CRUZ, 2004 apud COSTA et al, 2005, p. 3). “A lógica *fuzzy* tem por finalidade o estudo dos princípios formais do raciocínio aproximado.” (OLIVEIRA et al, 2007, p. 01).

A lógica *fuzzy* estende a lógica booleana, na qual um elemento pertence a um conjunto com certo grau de pertinência. Na teoria, esse grau é apresentado como uma função de pertinência. Essa função mapeia cada elemento do universo com um número entre 0 e 1. Deste modo, as proposições podem ser desde totalmente falsas até totalmente verdadeiras, passando por parcialmente falsas e parcialmente verdadeiras (MOREIRA, 2007, p. 273).

Segundo Transcheit (2007, p. 230), um conjunto *fuzzy* A em um universo X é definido por uma função de pertinência $\mu A(x)$: $X \rightarrow [0, 1]$, e representado por um conjunto de pares ordenados como apresentado no Quadro 1, onde $\mu A(x)$ indica quanto x é compatível com o conjunto A. Sendo que um determinado elemento pode pertencer a mais de um conjunto *fuzzy*.

- $A = \{(x, \mu A(x)) \mid x \epsilon U\}$;
- x é a variável do universo em estudo;
- μA é uma função cuja imagem pertence ao intervalo [0, 1];
- “1” representa o conceito de pertinência total;
- “0” representa a não pertinência.

Fonte: Moreira (2007, p. 274).

Quadro 1 – Regra de pertinência *fuzzy*

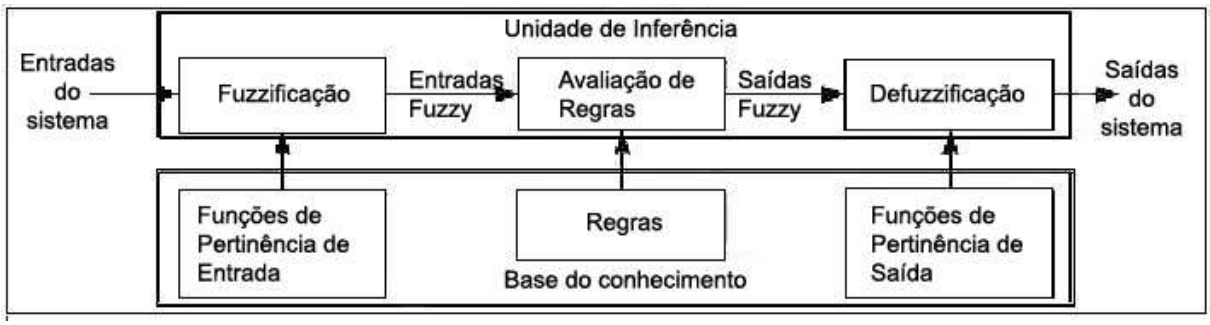
Os sistemas que utilizam a Lógica *Fuzzy* têm seu comportamento ditado por regras *fuzzy*. Que são caracterizadas por uma expressão condicional na forma SE <antecedente – expressão *fuzzy*> ENTÃO <conseqüente – expressão *fuzzy*> (MOREIRA, 2007, p. 275).

A arquitetura de um sistema *fuzzy* é formado por:

a) um módulo de fuzzificação onde são executados mapeamentos dos dados de entrada (em geral, números discretos) em números *fuzzy*;
b) um módulo de avaliação das regras, que é responsável por avaliar as variáveis de entrada aplicando as regras da base de conhecimento e atribuindo resposta ao processamento;

c) um módulo de defuzzificação, que recebe um conjunto *fuzzy* como resposta, porém, não é conveniente como resposta final para muitos sistemas, sendo necessário uma representação numérica sintética da resposta *fuzzy* (MOREIRA, 2007, p. 276).

A relação entre esse os módulos é ilustrada na Figura 2.



Fonte: Costa et al. (2005, p. 3).

Figura 2 – Arquitetura de um sistema *fuzzy*

Um modelo de inferência é o Mamdani, onde uma regra típica desse modelo é “se x é A e y é B então onde z é C”, onde A, B e C são conjuntos *fuzzy*. E a defuzzificação pode ser feita pelo método centróide, média dos máximos ou outros (OLIVEIRA et al, 2007, p. 47).

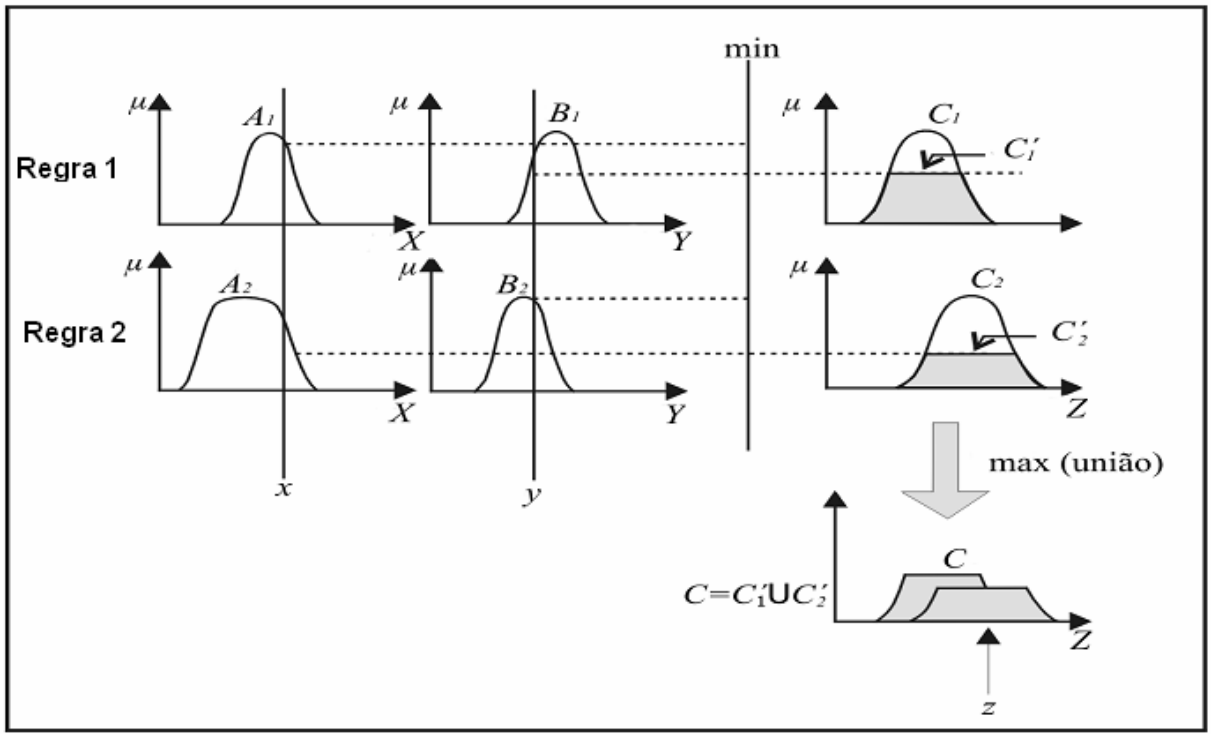
Segundo Lopes, Jafelice e Barros (2005, p. 81) no modelo Mamdani uma regra “se” (antecedente) “então” (conseqüente) é definida pelo produto cartesiano fuzzy dos conjuntos fuzzy que compõem os antecedentes e o conseqüente da regra. O método de Mamdani agrega as regras através do operador lógico “ou”, que é modelado pelo operador máximo, e em cada regra o operador lógico “e” é modelado pelo operador mínimo. Para exemplificar foram definidas as regras apresentadas no Quadro 2.

Regra 1: Se (x é A_1 e y é B_1) então (z é C_1)
Regra 2: Se (x é A_2 e y é B_2) então (z é C_2)

Fonte: Lopes, Jafelice e Barros (2005, p. 81).

Quadro 2 - Regras para exemplificar o modelo Mamdani

Figura 3 é ilustrada como uma saída real `z` de um sistema de inferência do tipo Mamdani é gerada a partir das entradas x e y reais e a regra de composição `max-min`, onde o valor discreto de z é obtido pelo defuzzificação do conjunto de saída C.



Fonte: Lopes, Jafelice e Barros (2005, p. 81).

Figura 3 - Modelo Mamdani

2.4 *NRC FUZZYJ TOOLKIT*

O NRC *FuzzyJ Toolkit* é um conjunto de classes Java que fornecem a facilidade para manipulação da lógica *fuzzy*. Estas classes estão divididas em dois pacotes: `nrc.fuzzy` que permite a construção de sistemas difusos em Java, e o pacote `nrc.fuzzy.jess` que prevê uma integração com o pacote `nrc.fuzzy` e com JESS[1]. A integração com Jess possibilita executar comandos *fuzzy* pelo *shell* (ORCHARD, 2006).

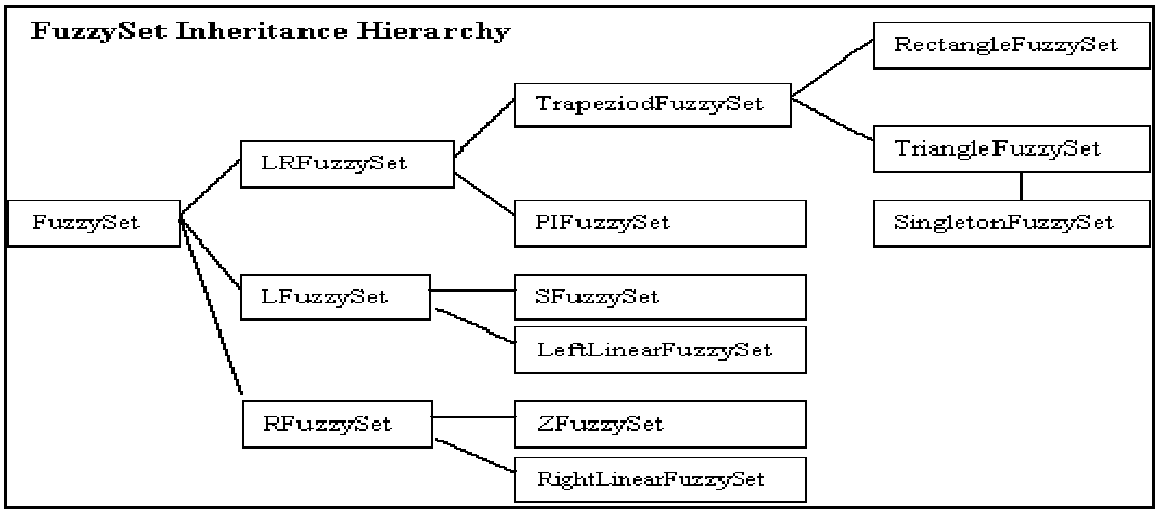
Os conceitos *fuzzy* são representados utilizando variáveis *fuzzy* (`FuzzyVariable`), conjuntos *fuzzy (*`FuzzySet`*)* e valores *fuzzy* (`FuzzyValue`). A `FuzzyVariable` é utilizada para descrever a forma geral de um conceito *fuzzy* e é constituída por um nome (por exemplo, "temperatura do ar"), a unidade (como graus C), o universo de discurso (exemplo de 0 à 100), e um conjunto de termos que são usados para descrever conceitos específicos para a variável. Os termos *fuzzy* são constituídos de um nome (por exemplo "frio", "quente"), e um `FuzzySet` que indica o grau de adesão a variável em questão (ORCHARD, 2006).

Estes termos juntamente com os *fuzzy* modificadores (tais como "muito" ou "*pouco")*,

os operadores "*and*", "*or*" (interseção e união, respectivamente) e os parênteses, fornecem a base para a gramática, que permite escrever expressões lingüísticas que descrevem os conceitos *fuzzy*. Estas expressões são codificadas em `FuzzyValue` que detém um conceito *fuzzy* específico (ORCHARD, 2006).

Conforme Orchard (2006) a diferença entre um `FuzzySet` e um `FuzzyValue` pode ser um pouco confusa, pois um `FuzzyValue` contém um `FuzzySet` e também esta associado ao seu contexto uma `FuzzyVariable`. A manipulação de `FuzzySet` pode ser realizada, por exemplo a interseção com outra `FuzzySet`. Por outro lado `FuzzyValue` também podem ser manipuladas, mas apenas com outras `FuzzyValue` que compartilhem a mesma `FuzzyVariable`.

Para simplificar a criação de `FuzzySet` são disponibilizadas as formas das funções de pertinência mais comuns como apresentado na Figura 4.



Fonte: Orchard (2006).

Figura 4 – Hierarquia do FuzzySet

No Quadro 3 é demonstrada a diferença na criação de função de pertinência triangular, fazendo uso da forma `TriangleFuzzySet` em relação a criação da mesma função utilizando `FuzzySet`.

Utilizando FuzzySet:

```
double xValues[] = {1, 2, 3};
double yValues [] = (0, 1, 0);
FuzzySet fSet = new FuzzySet (xValues, yValues, 3);
```

Utilizando TriangleFuzzySet:

```
FuzzySet fSet = new triangleFuzzySet (1, 2, 3);
```

Quadro 3 – Exemplo criação de funções de pertinência

Na lógica *fuzzy* são utilizadas regras para ditar o comportamento. A `FuzzyRule`, detém

[1] Jess é um motor de regras e um ambiente de *script* escrito interamente em Java (FRIEDMAN-HILL, 2008).

três conjuntos de `FuzzyValues` representado os antecedentes, a conclusão e entrada dos valores. Uma regra pode ser escrita como no Quadro 4, onde os antecedentes (*`antecedent`*) são as premissas da regra e devem ser verdadeiras, para que a regra execute as conclusões (*`conclusion`*) (ORCHARD, 2006).

```
if antecedent1 and antecedent2 and
          ...
   antecedentn
then
   conclusion1 and  conclusion2 and
          ...
   conclusionm
```

Fonte: Orchard (2006).

Quadro 4 – Regra *FuzzyJ*

Para iniciar o processo de inferência primeiro é necessário passar pela fuzzificação, onde os valores numéricos devem ser convertidos em valores *fuzzy*. Em seguida estes valores são adicionados aos respectivos antecedentes de cada regra definida, com adição de todos os antecedentes de uma regra a mesmo pode ser executada. Produzindo como saída valores *fuzzy*, o processo de conversão destes valores *fuzzy*, em representação numérica é conhecido como defuzzificação.

No Quadro 5 é apresentado um exemplo da utilização do *FuzzyJ*. Este exemplo demonstra a definição das variáveis temperatura e pressão. A construção de uma regra, onde são definidos os valores *fuzzy* que serão utilizados como antecedentes, conclusão e entrada da regra. E em seguida esta regra é executada e impresso seu resultado.

Na linha 18 é instanciada a regra, e na linha 19 é criado o antecedente desta regra onde é instanciado um `FuzzyValue` sendo passado a variável `temp` e a expressão lingüística `quente` (*hot*). Esta variável foi definida na linha 7 com o nome tem "`temperature`", o universo de discurso de 0 a 100 e com a unidade "C", na linha 8, 9 e 10 são definidas as expressões lingüísticas utilizadas e seu devidos valores. Na linha 20 é definida a conclusão da regra com a variável `pressure` e a expressão lingüística `baixo ou médio` (*low or medium*), a definição desta variável encontra-se na linha 13 e as expressão lingüísticas da mesma foi declarado nas linhas 14, 15 e 16 utilizando-se das funções de pertinência fornecidas pelo FuzzyJ. O conjunto *fuzzy* de entrada para a regra é instanciado na linha 21, e a execução da regra ocorre na linha 27 utilizando o modelo Mamdani max-min.

Como saída da execução da regra se tem um vetor de conjuntos fuzzy que correspondem ao número de conclusões adicionadas a regra, neste caso 1 (uma). A regra aqui demonstrada é escrita da seguinte forma: **`Se`** `a temperatura é quente` **`então`** `pressão é baixa ou média.`

```
1.  //some values used to describe the fuzzy terms in the temperature FuzzyVariable
2.  double xHot[] = {25, 35};
3.  double yHot[] = {0, 1};
4.  double xCold[] = {5, 15};
5.  double yCold[] = {1, 0};
6.  //define our temperature FuzzyVariable with terms hot,cold, very hot and medium
7.  FuzzyVariable temp = new FuzzyVariable("temperature", 0, 100, "C");
8.  temp.addTerm("hot", xHot, yHot, 2);
9.  temp.addTerm("cold", xCold, yCold, 2);
10. temp.addTerm("veryHot", "very hot");
11. temp.addTerm("medium", "(not hot and (not cold))");
12. // define our pressure FuzzyVariable with terms low, medium and high
13. FuzzyVariable pressure = new FuzzyVariable("pressure", 0, 10, "kilo-pascals");
14. pressure.addTerm("low", new ZFuzzySet(2.0, 5.0));
15. pressure.addTerm("medium", new PIFuzzySet(5.0, 2.5));
16. pressure.addTerm("high", new SFuzzySet(5.0, 8.0));
17. // let's build a rule ---
18. FuzzyRule rule1 = new FuzzyRule();
19. FuzzyValue antecedentFval = new FuzzyValue(temp, "hot");
20. FuzzyValue conclusionFval = new FuzzyValue(pressure, "low or medium");
21. FuzzyValue inputFval = new FuzzyValue(temp, "very medium");
22. rule1.addAntecedent(antecedentFval);
23. rule1.addConclusion(conclusionFval);
24. rule1.addInput(inputFval);
25. // execute this simple rule with a single antecedent and  a single consequent
26. using default //rule executor --  MamdaniMinMaxMinRuleExecutor
27. FuzzyValueVector fvv = rule1.execute();
28. // show the results using the plotting methods for FuzzyValues
29. FuzzyValue fvals[] = new FuzzyValue[2];
30. fvals[0] = antecedentFval;
31. fvals[1] = inputFval;
32. System.out.println(FuzzyValue.plotFuzzyValues("*+", 0, 50, fvals));
33. System.out.println(fval2.plotFuzzyValue("*", 0, 10));
34. System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("*", 0, 10));
35. //execute again with a different rule executor LarsenProductMaxMinRuleExecutor
36. fvv = rule1.execute(new LarsenProductMaxMinRuleExecutor());
37. // and show results
38. System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("*", 0, 10));
```

Fonte: Orchard (2006).

Quadro 5 – Exemplo em Java *FuzzyJ*

2.5 MATLAB – *FUZZY LOGICAL TOOLBOX*

No Fuzzy Logical Toolbox, estão disponíveis arquivos e funções de suporte ao uso da teoria *fuzzy*.

Para executar o Fuzzy Logical Toolbox na tela de comandos do Matlab deve-se digitar >> *fuzzy* em seguida precione Enter, a partir abrirá a tela inicial do *Toolbox*. No exemplo mostrado aqui será demonstrado o modelo Mamdani de inferência *fuzzy* Figura 5.

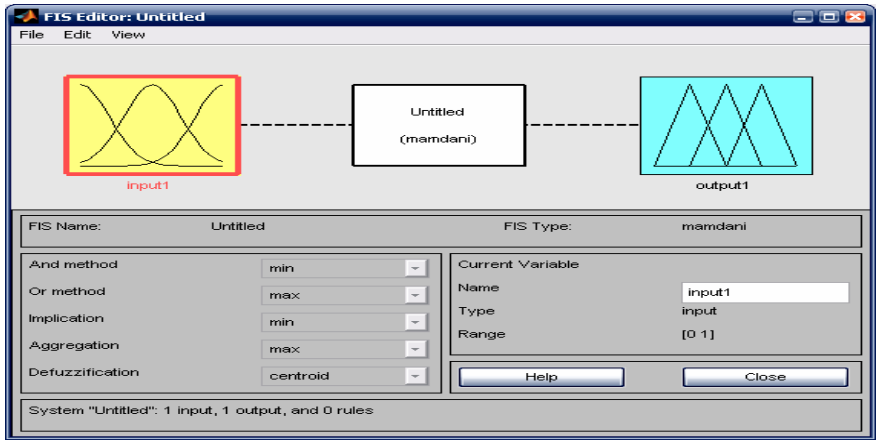


Figura 5 - Tela inicial do Fuzzy Logical Toolbox

É importante ressaltar que existem diversos sistemas de inferências *fuzzy*, e o *Fuzzy Logical Toolbox do* MatLab oferece duas opções: o modelo Mamdani e o modelo Sugeno.

Para adicionar variáveis, são criadas “caixas” para guardá-las, podendo haver várias variáveis de entrada e saída. Para cada variável deve se informar o domínio e suas funções de pertinências, para cada função de pertinência devem ser informado o nome, formato da função e seus respectivos pontos. Na Figura 6 é apresentada a tela onde são configuradas as variáveis.

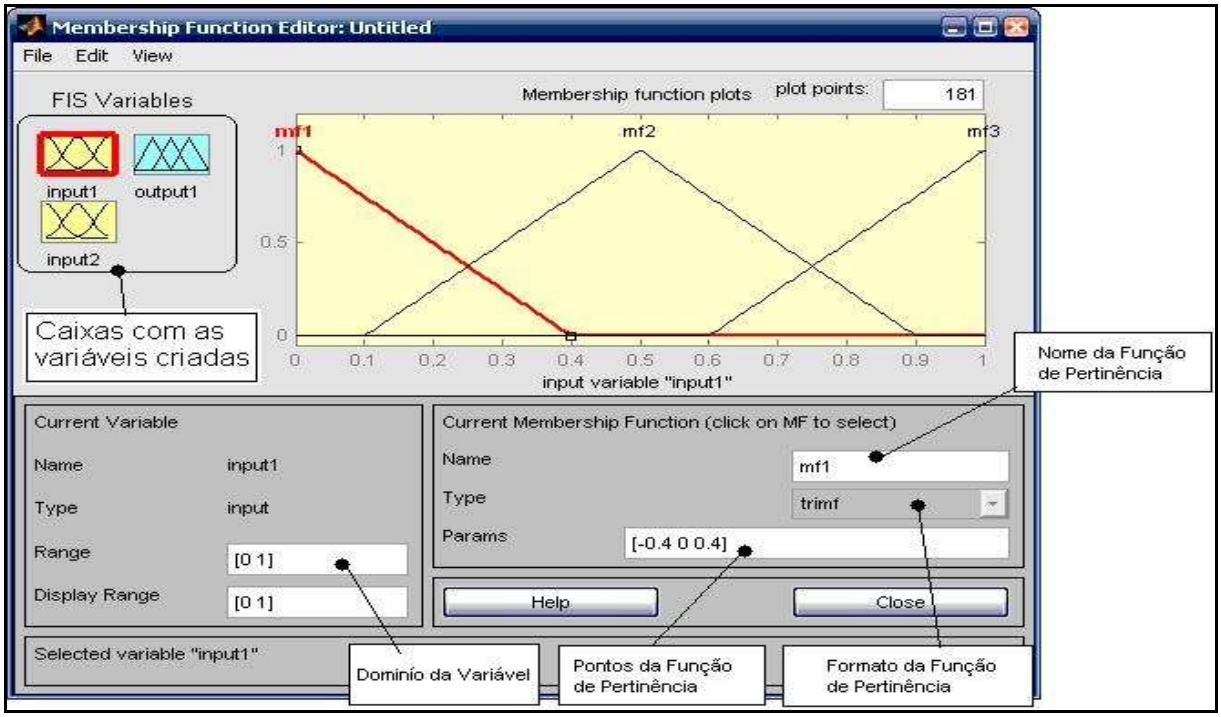


Figura 6 – Editando variáveis

Após a definição das variáveis, deve se criar a base de regras. Na construção da base de regras deve se informar a conexão entre as variáveis, através do operador lógico (Figura 7).

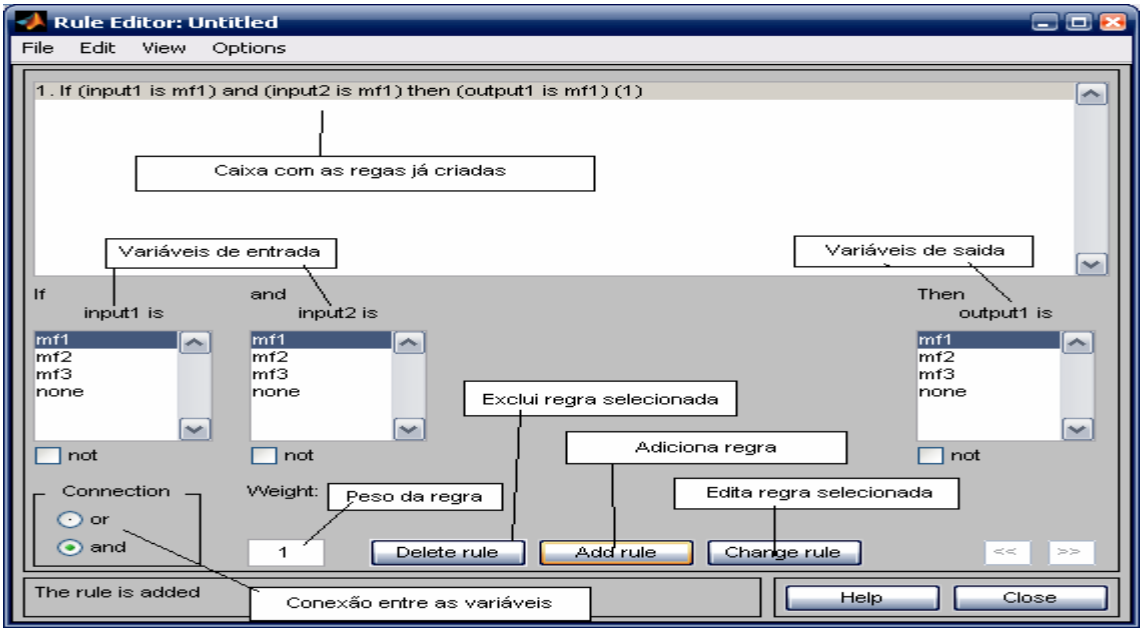


Figura 7 – Adicionando regras

2.6 CELINE

CELINE é um componente para facilitar a utilização de Objeto de Aprendizagem (AO) no padrão *Sharable Content Object Reference Model* (SCORM). O OA são entidades de ensino que podem ser reutilizadas. Os sistemas que tem como funcionalidade gerenciar OA e interações com usuários são chamados de *Learning Management System* (LMS). O SCORM é um conjunto de padrões e especificações, com propósito de definir um ambiente de aprendizagem que promova um ensino de qualidade e de custo baixo (VAHLDICK, RAABE, 2008).

Conforme Vahldick e Raabe (2008), CELINE provê recursos para atender os requisitos:

a) executar pacotes SCORM;
b) administrar pacotes SCORM;
c) administrar usuário;
d) permitir que usuário se registre em cursos;
e) permitir que o usuário interrompa a interação com o curso e continuem em outros instantes;
f) apresentar os estados das interações dos usuários com os curso.

O componente está dividido em duas partes. Uma que executa os pacotes SCORM (CELINESCORM) e a outra para toda a gestão do LMS (CELINELMS) (VAHLDICK, RAABE, 2008).

O CELINESCORM consiste em um conjunto de classes que segue a especificação SCORM, podendo ser executada em qualquer contexto, seja web, interface gráfica ou dispositivos móveis.

O CELINELMS possui um mecanismo de persistência que é responsável por manter os dados necessários para a tarefa de gestão. Para isso, são fornecidas duas interfaces DAO. A XMLDAO que grava e busca dados de um arquivo XML e RDBDAO que utiliza um banco de dados relacional para persistir os dados. Para o desenvolvimento de páginas são fornecidas *tags* customizadas, por exemplo, para listar os cursos cadastrados, encerrar e suspender a interação.

Para a utilização do componente é necessário adicionar os JAR do CELINE, além das bibliotecas de terceiros. Também será necessário definir um arquivo XML chamado `celine-config.xml` que conterá as informações de inicialização necessária ao componente.

Também são necessários adicionar ao arquivo `WEB-INF/web.xml` as linhas do Quadro 4 que configuram:

a) um objeto LMSContextListener executado quando inicia a aplicação e carrega o `celine-config.xml`;

b) *servlet* do CELINELMS (LMS);

c) *servlet* para tratar AJAX (DWRServlet);

d) mapeamento para cada um dos *servlets*.

```
<listener>
   <listener-class>br.univali.celine.lms.core.LMSContextListener</listener-class>
</listener>
<servlet>
   <servlet-name>lms</servlet-name>
   <servlet-class>br.univali.celine.lms.core.LMS</servlet-class>
</servlet>
<servlet>
   <servlet-name>dwr-invoker</servlet-name>
   <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>dwr-invoker</servlet-name>
   <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>lms</servlet-name>
   <url-pattern>/lms</url-pattern>
</servlet-mapping>
```

Fonte: Vahldick e Raabe (2008) .

Quadro 6 - Linhas a serem adicionadas no web.xml

2.7 TRABALHOS CORRELATOS

Forest (2004) produziu um sistema tutor para auxílio a aprendizagem de orientação a objetos, onde o professor cadastra as atividades a serem realizadas pelo aluno. Estas atividades cadastradas podem ser selecionadas pelo aluno para a realização das mesmas, auxiliando o aluno na criação de diagramas, produção de casos de uso, análise na classificação dos verbos e substantivos para geração de diagramas. Ao final gera o código fonte na linguagem Java.

O sistema LabSQL, é uma ferramenta para avaliação automáticas de consultas SQL, atribuindo uma nota a consulta SQL submetida pelo aluno. E utiliza lógica difusa para decidir conceitos finais dos estudantes críticos, isto é que estão muito perto do limite para alcançar outros conceitos melhores. As avaliações do LabSQL são elaborados pelo professor que atribui a cada uma o grau de dificuldade e prevê um tempo para execução. Ao final do semestre letivo a ferramenta avalia cada questão executada pelo aluno, obtendo o percentual de acertos da cada avaliação e por meio das informações fornecidas pelo LabSQL o sistema difuso faz inferência do conceito final do estudante. As entradas do sistema difuso são 4 variáveis (nota final do estudante, interesse e esforço, progressão e freqüência do estudante no ambiente LabSQL). A maquina de inferência processa as variáveis e gera a variável de saída (conceito final), a base de regra é composta por 50 regras (SILVA et al, 2008).

3 DESENVOLVIMENTO

Neste trabalho foram desenvolvidos dois softwares:

a) um objeto de aprendizagem para o desenvolvimento de atividade gerenciador de armazenamento e arquivos;
b) um STI para realizar o acompanhamento do aluno, e definir qual a próxima atividade a ser desenvolvida.

3.1 OBJETO DE APRENDIZAGEM PARA EXERCICÍOS DE GERENCIAMENTO DE ARMAZENAMENTO E ARQUIVOS

Neste capítulo são apresentados os requisitos, diagramas da UML, a implementação e exemplos de uso do objeto de aprendizagem produzido neste trabalho.

3.1.1 Requisitos

O objeto de aprendizagem deverá:

a) disponibilizar uma interface para que o aluno realize as atividades (Requisito Funcional - RF);
b) ser implementado no ambiente de desenvolvimento Netbeans 5.5 (Requisito Não-Funcional – RNF);
c) ser implementado utilizando a linguagem Java (RNF);
d) ser uma *applet* (RNF);
e) ser empacotado no modelo SCORM (RNF).

3.1.2 Especificações

Nesta seção são apresentados os casos de uso, classes, atividades e de seqüência do

objeto de aprendizagem para exercícios gerenciamento de armazenamento e arquivos de um SGBD.

3.1.2.1 Casos de uso

O diagrama de casos de uso é utilizado para descrever interações que usuários externos a aplicação tem com o software. A Figura 8 mostra o diagrama de casos de uso.

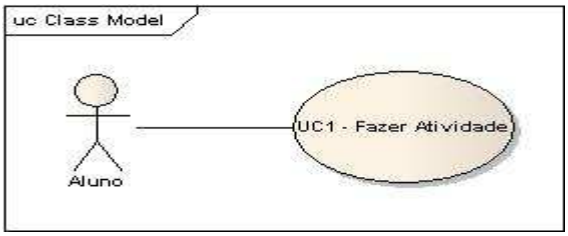


Figura 8 - Diagrama de casos de uso do objeto de aprendizagem

No Quadro 7 apresenta a descrição do caso de uso edita atividade.

UC1 – Fazer Atividade

Sumário: Aluno realizar a atividade.

Ator primário: Aluno.

Precondições: Nenhuma.

Fluxo Principal:
1. O aluno seleciona o bloco que deseja editar
2. O sistema apresenta o bloco.
3. O aluno seleciona registro ou byte inicial para edição do registro.
4. O sistema apresenta tela para edição do registro.
5. O aluno adiciona quantos campos cabeçalho e dados forem necessário.
6. O sistema apresenta os campos a serem preenchidos.
7. O aluno solicita a ordenação dos campos do registro.
8. O sistema ordena os campos do registro.
9. O aluno informa o byte inicial e aplica a alteração.
10. O sistema adiciona o registro ao bloco.

Fluxo alternativo (1): Novo Bloco
a) No passo 1, caso não tenha o bloco desejado.
b) O aluno adiciona novo bloco.
c) O sistema apresenta tela para cadastro do novo bloco.
d) O aluno cadastra o número do bloco, o número do disco , a quantidade de bytes e aplica.
e) O sistema adiciona o bloco e retorna ao passo 2.

Fluxo alternativo (2): Edita Bloco
a) No passo 2, caso o seja necessário alterar o bloco.
b) O aluno edita o bloco.
c) O sistema apresenta tela para editar o bloco.
d) O aluno edita o número do bloco, o número do disco , a quantidade de bytes e aplica.
e) O sistema altera o bloco e retorna ao passo 3.

Fluxo alternativo (3): Adiciona mais campos no registro
No passo 8, caso o aluno necessite adicionar mais campos retorna ao passo 5.

Fluxo alternativo (4): Adiciona mais registros no bloco
No passo 10, caso o aluno necessite adicionar ou editar mais registros retorna ao passo 3.

Fluxo alternativo (5): Término da atividade
a) Quando o aluno solicitar a finalização da atividade.
b) Caso atividade ainda não tenha sido encerrada o sistema solicita se deseja encerrar.
c) O aluno informa se deseja encerrar.
d) Se atividade for encerrada o sistema efetua a correção da mesma.
e) O sistema é encerrado.

Pós-condições: Atividade realizada.

Quadro 7 – Descrição do caso de uso `UC1 – Fazer atividade`

3.1.2.2 Diagrama de classes

O diagrama de pacotes é utilizado para ilustrar arquitetura de um sistema mostrando o agrupamento de suas classes. A Figura 9 mostra o diagrama de pacotes do objeto de aprendizagem.

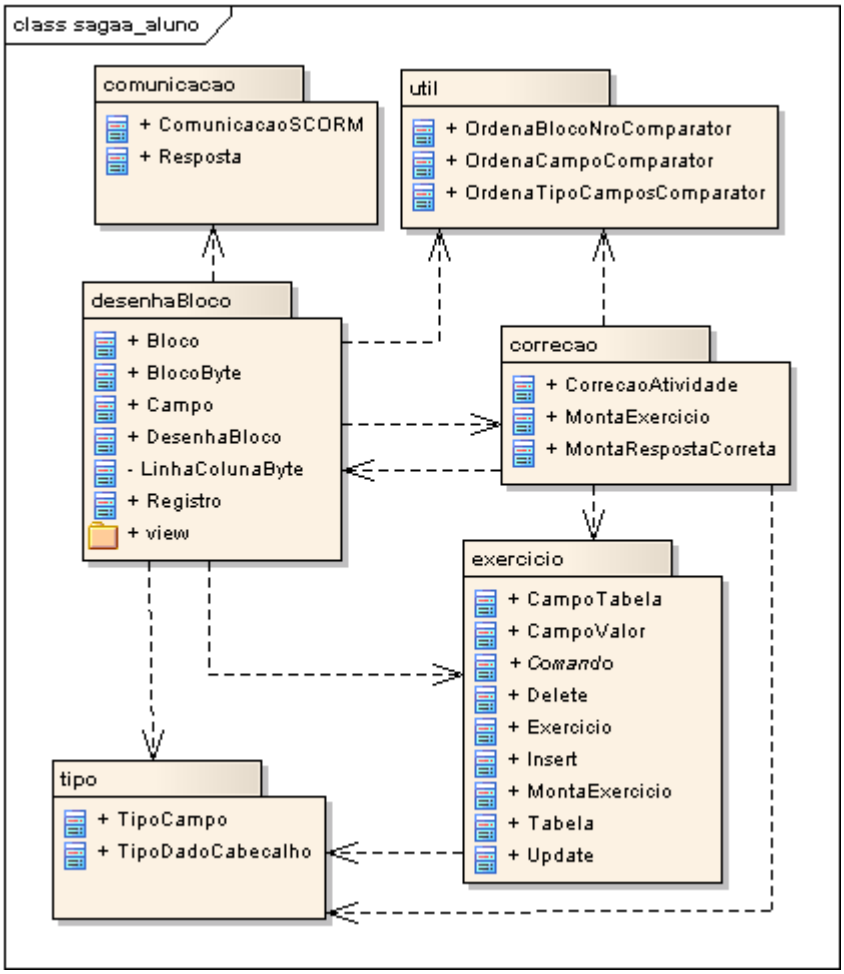


Figura 9 – Diagrama de pacotes de classes

No pacote *`view`* são agrupadas as classes de interface com o usuário. No pacote `desenhaBloco` são agrupadas as classes responsáveis pela estrutura de apresentação do bloco. No pacote `correcao` são agrupadas as classes de correção da atividade. O pacote `exercicio` agrupa as classe de estrutura do exercício a ser realizado. No pacote `util` contém classes de uso geral. No pacote `tipo` agrupa as classe de *Enumeration*.

A Figura 10 representa o diagrama de classes responsável pela estrutura de representação do bloco.

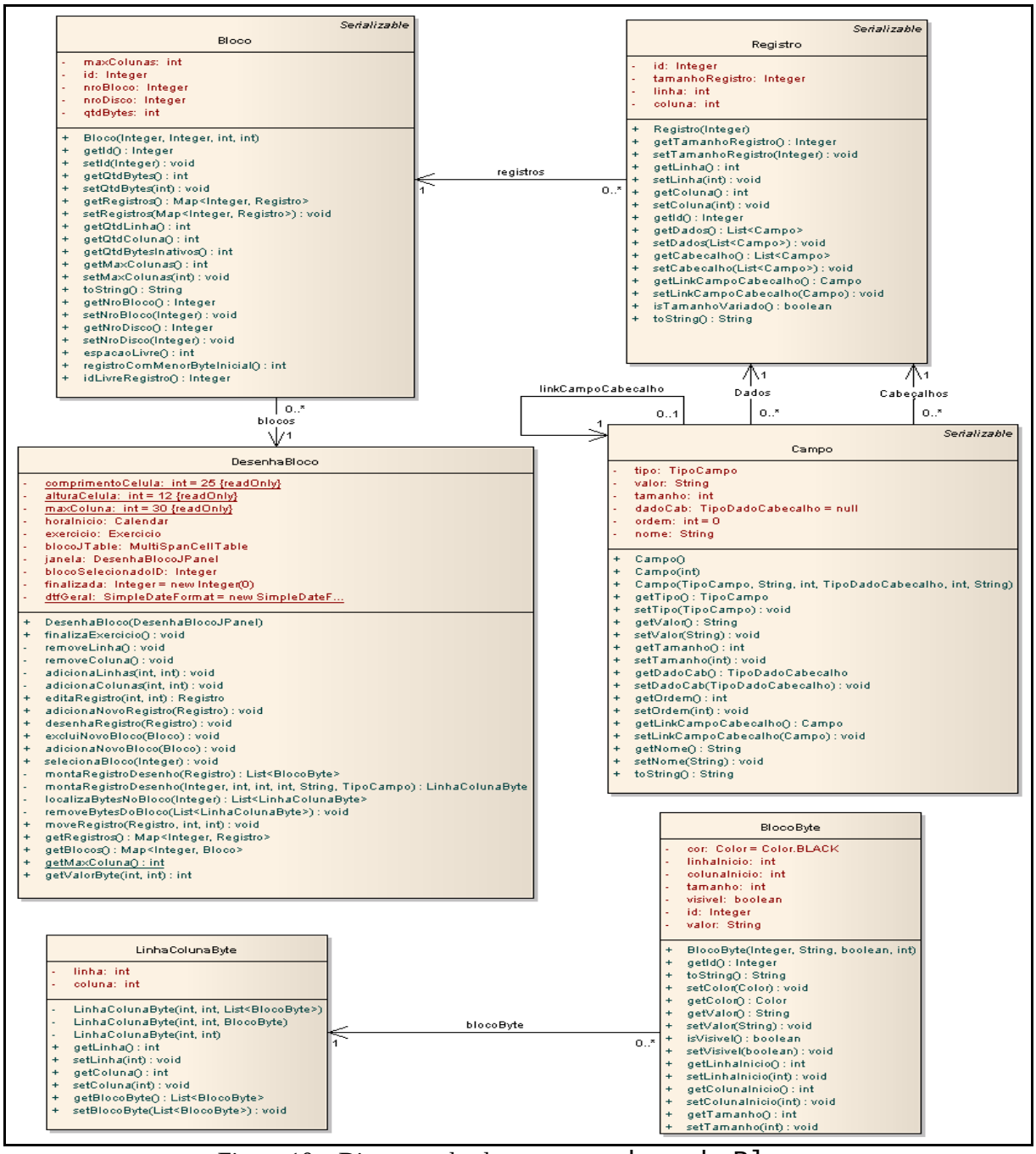


Figura 10 – Diagrama de classes pacote `desenhaBloco`

A classe `DesenhaBloco` é responsável por manter os blocos e gerenciar a atividade a ser realizada. As classes `Bloco`, `Registro` e `Campo` são as estruturas para armazenar as interação do usuário. As classes `LinhaColunaByte` e `BlocoByte` são estruturas auxiliares utilizadas para montar a visualização do usuário.

3.1.2.3 Diagrama de atividade

O diagrama de atividade na Figura 11, mostra o fluxo de controle para realização de atividades.

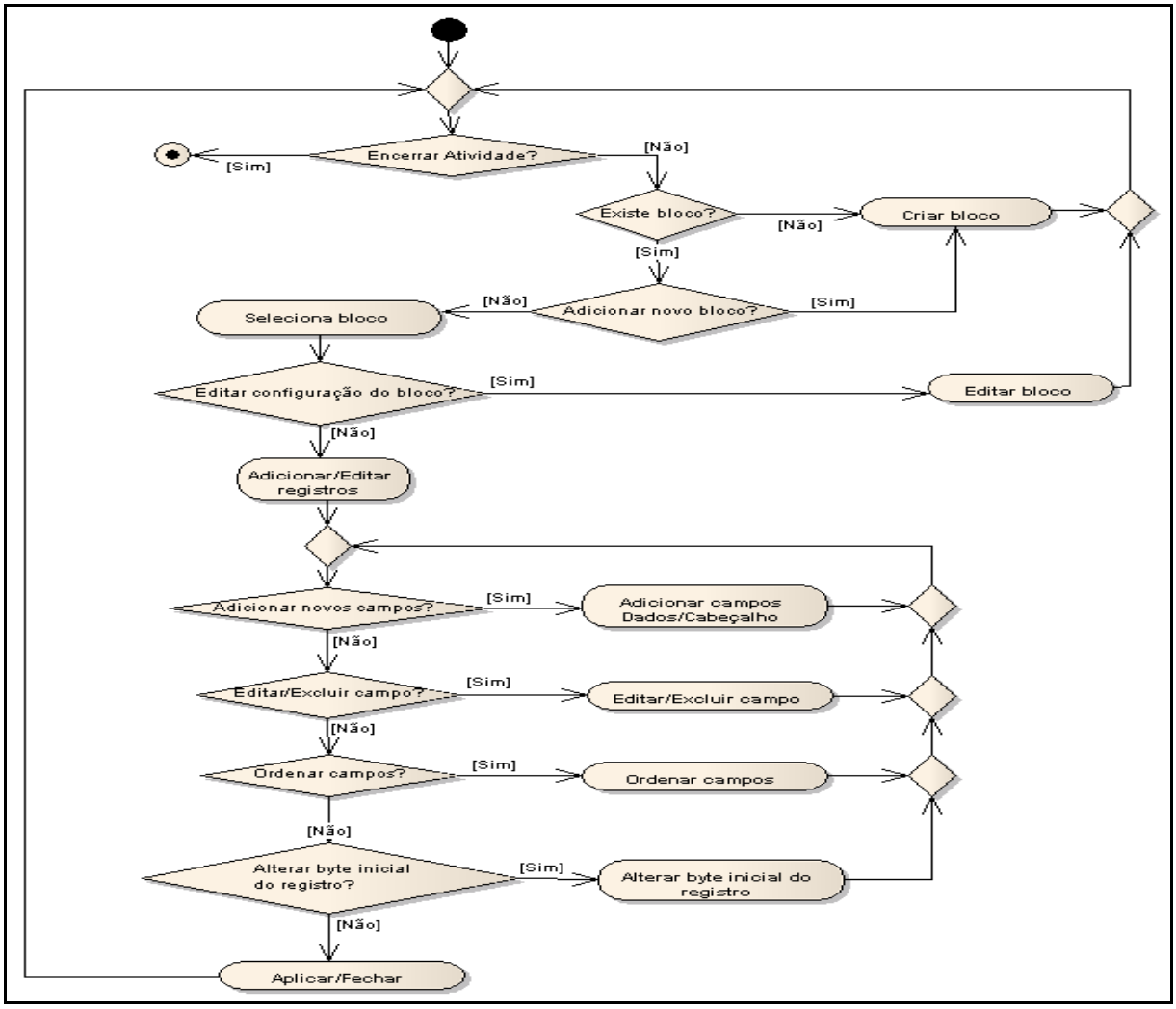


Figura 11 – Diagrama de atividade realização da atividade proposta

Inicialmente, o aluno pode optar por encerrar ou realizar a atividade. Caso venha realizar a atividade e a mesma ainda não tenha blocos é necessária a criação de um bloco (informando o número do bloco, o número do disco e o tamanho em bytes) e ao término da criação do bloco o aluno pode optar em encerrar atividade, criar novo bloco ou selecionar um bloco. Caso a atividade já possua blocos o aluno pode optar em criar um novo bloco ou selecionar um bloco.

Após a seleção do bloco, é possível alterar as configurações do mesmo ou adicionar/editar um registro. Caso se opte por editar as configurações do bloco é possível alterar o número do bloco, número do disco e tamanho. Ao término da alteração é possível

optar em encerrar a atividade, criar novo bloco ou selecionar um bloco. Caso a opção tenha sido por adicionar/editar um registro, pode-se adicionar, excluir ou ordenar campos, e alterar byte inicial do registro e para finalizar a edição do registro aplicar/fechar. Após terminar a edição do registro pode-se encerrar a atividade, selecionar ou criar novo bloco.

3.1.2.4 Diagrama de seqüência

O diagrama de seqüência é usado para representar a seqüência de processos. Na Figura 12 é mostrado o diagrama de seqüência para o processo de correção de atividade.

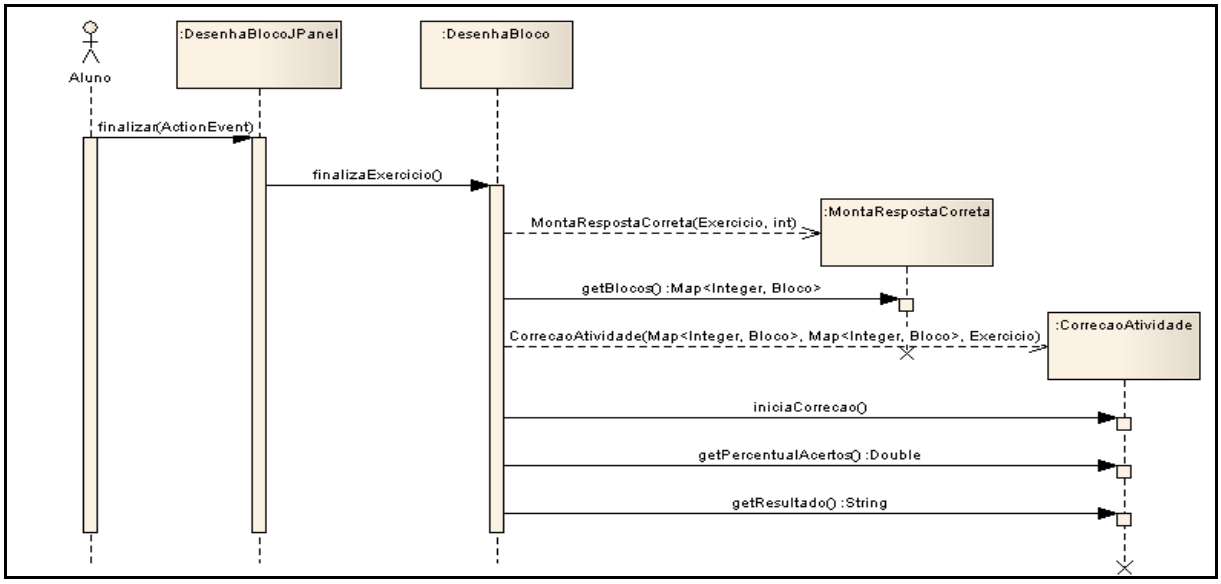


Figura 12 – Diagrama de seqüência da correção da atividade

O processo de correção acontece quando aluno clicar no botão de encerrar atividade, sendo chamado o método "`finalizaExercicio()`" da classe `DesenhaBloco`. Em seguida, é instanciada a classe `MontaRespostaCorreta`, onde será montado o gabarito para atividade que está sendo desenvolvida, e é solicitado o gabarito que foi gerado. O próximo passo é a instanciação da classe `CorrecaoAtividade`, e é requisitado que inicie a correção, em seguida solicita o percentual de acertos e o resultado da correção realizada.

3.1.3 Implementação

Para o desenvolvimento do objeto de aprendizagem foi utilizada a linguagem Java JDK 1.6 através da plataforma *Netbeans* 5.5. Foram utilizados componentes gráficos da

plataforma para interface com usuário.

Foi desenvolvido um *applet* para que o aluno possa realizar atividade do gerenciador de armazenamento e arquivo. Esta *applet* possibilita ao aluno a montagem de blocos de um arquivo de dados do SGBD, com seu registros e os registros com seus campos.

Na inicialização da *applet* é realizada a leitura de um XML que contém as informações da atividade a ser realizada. Este arquivo deve estar localizada no mesmo diretório onde encontra-se o arquivo HTML que faz a chamada da applet e possuir o seguinte nome "atividade.xml".

O exemplo para chamada do *Applet* pode ser observado no Quadro 1.

```
<applet archive="../common/SAGAA_ALUNO.jar"
             code="sagaa_aluno.desenhaBloco.view.DesenhaBlocoApplet"
             type="applet" jreversion="1.5" width="800" height="600"
             mayscript="mayscript">
            Erro ao Carregar Applet...
</applet>
```

Quadro 8 – Chamada do *Applet*

A interpretação do XML é feita utilizando a API DOM disponível a partir do JDK 1.4. Conforme vai sendo interpretado é mapeado para classe Java para tornar sua manipulação mais prática. A ordem com que as tabelas, *inserts*, *updates* e *deletes* são interpretados é mantido. As variáveis de peso informada no XML são somente utilizadas após o termino da correção para realizar o calculo de percentual de acertos.

A rotina de correção da atividade é realizada da seguinte forma: primeiramente é criado um gabarito. Este gabarito é uma estrutura idêntica ao que o aluno deve montar durante a realização da atividade, e tem como base para sua montagem a estrutura que foi criada a partir da interpretação do XML; a segunda parte é uma comparação entre o que ao aluno realizou e o gabarito montado, onde se inicia calculando o total de itens que serão avaliados.

Em seguida verifica-se a quantidade de blocos que foram criados pelo aluno. Após inicia-se a verificação de cada bloco pelo número do mesmo e quantidade de byte, caso esteja correto continua-se a correção do bloco e adiciona ao contador mais um bloco correto senão passa a verificar próximo bloco. Com o bloco correto inicia-se a correção do seu conteúdo, onde o registro deve-se iniciar no byte inicial esperado e possuir os campos com tipo, valor e tamanho corretos para que o registro esteja correto, para cada campo de dado ou de cabeçalho correto contabiliza-se um acerto para o respectivo tipo de campo. Ao final se divide o numero de acertos pelo numero de acertos esperados em cada item, multiplicando pelo peso que foi informado no XML da atividade. O XML que deve ser passado a *Applet* possui a estrutura apresentada Quadro 9.

```
<exercicio name="nome_da_atividade">
    <descricao>Enunciado da Atividade</descricao>
    <!--Informações necessária que possa ser feito a correção -->
    <tamanhobloco>64</tamanhobloco> <!-- byte -->
    <tamanhoMaxIDTabela>50</tamanhoMaxIDTabela> <!-- int -->
    <mapaDeslocamentoRegistro>2</mapaDeslocamentoRegistro> <!-- int -->
    <tamanhoMaxArquivo>36864</tamanhoMaxArquivo> <!-- byte -->
    <qtdDisco>1</qtdDisco><!-- int -->
    <tamanhoPonteiro></tamanhoPonteiro><!-- int -->
    <qtdMaxBloco></qtdMaxBloco><!-- int -->
    <tamanhoMapaVarchar></tamanhoMapaVarchar><!-- byte -->
    <IDTabelaBlob>255</IDTabelaBlob> <!-- int -->
<!--Valor utilizados na correção da atividade, a soma devem sempre fechar em 100-->
    <pesoQtdblocos>20</pesoQtdblocos><!-- int -->
    <pesoblocos>20</pesoblocos><!-- int -->
    <pesoRegistro>20</pesoRegistro><!-- int -->
    <pesoCampoDado>20</pesoCampoDado><!-- int -->
    <pesoCampoCabecalhoBloco>20</pesoCampoCabecalhoBloco><!-- int -->
     <!--Caso seja 'S' o sistema montará a resposta na inicialização  da Applet -->
     <atividadeExemplo>N</atividadeExemplo>
     <tabela>
        <nome>Nome da tabela</nome>
        <id>Id da tabela</id>
        <campo>
             <nome>Nome campo 1</nome> <!-- Obrigatório -->
             <tipo>Tipo do campo 1</tipo> <!-- Obrigatório -->
             <tamanho>Tamanho do campo quando de tamanho variável</tamanho>
             <restricao>Restrição do campo (PK)</restricao>
             <foreignKey>Nome da tabela:Nome do Campo</foreignKey>
        </campo>
     </tabela>
    <insert>
          <tabelaNome>Nome da tabela</tabelaNome>
          <values>
               <campo>Nome do campo</campo>
               <valor>Valor a ser inserido</valor>
           </values>
     </insert>
     <update>
          <tabelaNome>Nome da Tabela</tabelaNome>
            <set>
               <campo>Nome do campo a ser alterado</campo>
               <valor>Valor da alteração</valor>
            </set>
            <where>
                <campo>Nome do campo ser selecionado</campo>
                <valor>Valor a ser selecionado</valor>
                <condicao>Condição de seleção</condicao>
            </where>
      </update>
    <delete>
        <tabelaNome>Nome da tabela</tabelaNome>
        <where>
            <campo> Nome do campo ser selecionado </campo>
            <valor> Valor a ser selecionado</valor>
            <condicao> Condição de seleção </condicao>
        </where>
    </delete>
</exercicio>
```

Quadro 9 – Estrutura do XML

Para que o aluno tenha a possibilidade de ver mais tarde o que ele executou em atividades anteriores ou mesmo suspender uma atividade para continuar posteriormente, cada vez que atividade é encerrada é instanciada a classe Resposta onde um dos atributos é o conjunto de bloco que foram criados, e seus respectivos registros e os registros com seus

campos. A classe Resposta é convertida em um *XML* e retornado por uma função JavaScript, caso seja armazenada, quando a atividade for solicitada novamente e já tiver alguma resposta anterior é apresentado ao aluno o que tinha sido realizado.

Para que a atividade siga o padrão SCORM, esta deve ser empacotada em uma pasta com a estrutura apresentada na Figura 13. O nome desta pasta deverá ser o nome da atividade. O arquivo JAR gerado a partir do objeto de aprendizagem desenvolvido neste trabalho deve estar dentro da subpasta `common`. A atividade e os arquivos HTML utilizados devem estar dentro da subpasta `resourses`, onde a atividade deverá estar no formato do Quadro 9 e o nome do arquivo deverá ser `atividade.xml`. Na pasta principal deve constar o arquivo `imsmanifest.xml` com as configurações para esta atividade. Uma parte do arquivo é demonstrada no Quadro 10.

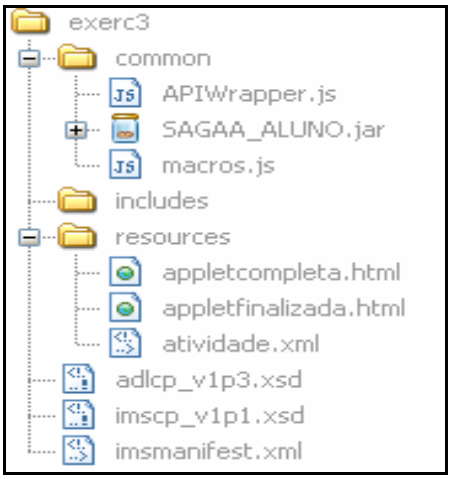


Figura 13 - Pasta com atividade

```
...
<organizations default="Applet">
    <organization identifier="Applet">
        <title>Exercicio_C1_E1_F</title>
        <item identifier="exercicioC1E1F" identifierref="exercicioC1E1F1">
                    <title>Exercicio_C1_E1_F</title>
         </item>
      </organization>
 </organizations>
 <resources>
    <resource identifier="exercicioC1E1F1" adlcp:scormType="sco"
            href="appletcompleta.html" type="webcontent" xml:base="resources/">
          <file href="appletcompleta.html" />
          <file href="appletfinalizada.html" />
     </resource>
 </resources>
...
```

Quadro 10 - Arquivo `imsmanifest.xml` de configuração da atividade

O elemento <identifier> deve ser um identificador único que não pode ser repetido novamente em outra atividade. O elemento <resources> descreve os arquivos que fazem parte do objeto, onde o atributo href define a primeira página do objeto.

As interações realizadas entre o objeto de aprendizagem e o LMS são feitas através de funções JavaScript, o envio de informações ao LMS é feito através da funções AddInteraction e UpdateInteraction. O primeiro serve para adicionar uma nova informação ao LMS, o segundo é utilizado quando deseja alterar o valor informação já existente no LMS. A solicitação de informações do LMS é realizada pela função GetInteraction. Estas funções são apresentadas no Quadro 11.

```
function AddInteraction(id, type, learnerResponse) {
   var count = retrieveDataValue("cmi.interactions._count");
   var res = storeDataValue("cmi.interactions."+count+".id", id);
   if (res != "true")
       return "invalid_id";
   storeDataValue("cmi.interactions."+count+".type", type);
   storeDataValue("cmi.interactions."+count+".learner_response", learnerResponse);
   return count;
 }

 function GetInteraction(id) {
   var count = retrieveDataValue("cmi.interactions._count");
   for (var i = 0; i < count; i++) {
       var resId = retrieveDataValue("cmi.interactions."+i+".id");
       if (resId == id) {
              return retrieveDataValue("cmi.interactions."+i+".learner_response");
       }
    }
    return "invalid_id";
 }

 function UpdateInteraction(id, learnerResponse) {
   var count = retrieveDataValue("cmi.interactions._count");
   for (var i = 0; i < count; i++) {
     var resId = retrieveDataValue("cmi.interactions."+i+".id");
     if (resId == id) {
        storeDataValue("cmi.interactions."+i+".learner_response", learnerResponse);
        return i;
     }
   }
   return "invalid_id";
}
```

Quadro 11 - Funções JavaScript de interação com LMS

3.1.4 Operacionalidade de implementação

A seguir são apresentadas as telas do objeto de aprendizagem com explicação de suas funcionalidades.

Na tela inicial do objeto de aprendizagem, apresentado na Figura 14, o aluno poderá visualizar o enunciado da atividade que deverá ser realizada. Para iniciar a atividade deve ser criado um novo bloco, com isso o bloco criado irá aparecer na caixa de seleção e a quantidade de bytes definida será desenhada. Com um bloco selecionado, pode ser iniciada a inserção, edição e exclusão dos registros deste bloco.

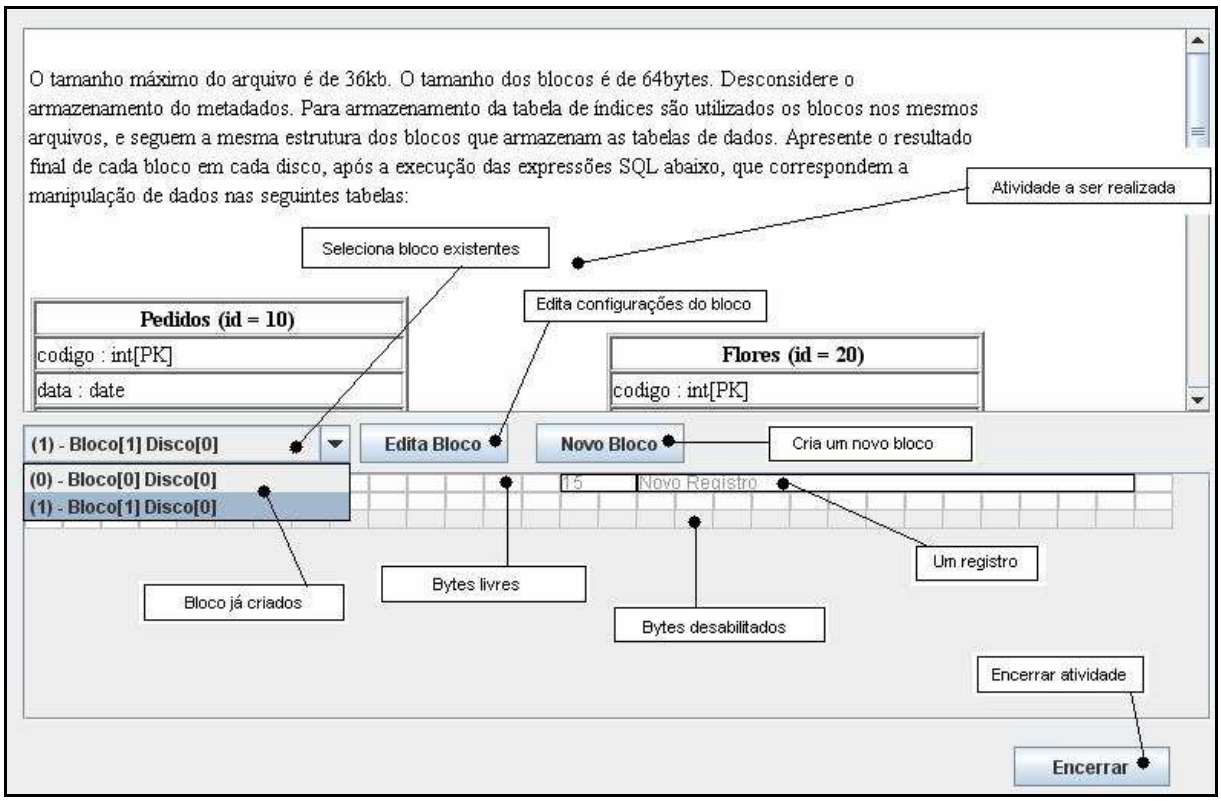


Figura 14 – Tela inicial do objeto de aprendizagem

Para criação de um novo bloco é necessário informar o número do bloco, número do disco e tamanho (bytes) do mesmo. Para editar estas configurações é necessário selecionar um bloco. A Figura 15, mostra a tela de configuração de um bloco.

Figura 15 – Tela de configuração do bloco

Para adicionar um novo registro é necessário selecionar um byte (clique com o botão esquerdo do mouse) do bloco, onde este byte será o byte inicial do registro podendo ser alterado na tela de configurações do registro. Para abrir a tela de configurações do registro em cima do byte selecionado clique com o botão direto do mouse. Para editar um registro é necessário selecionar o registro, clindo sobre o registro que se deseja selecionar com o botão esquerdo do mouse, e em seguida clicar com o botão direto do mouse e é aberta a tela de configuração do registro. A exclusão de um registro é realizada quando são excluídos todos os seus campos. Uma restrição é que os registros devem estar sempre no mesmo bloco caso não sejam do tipo BLOB. Na Figura 16 é apresentada a tela de configuração do registro.

Figura 16 – Tela de configuração do registro

Ao encerrar a atividade o aluno será questionado se deseja finalizar. Em caso positivo a atividade será corrigida. Na próxima vez que abrir está atividade não será possível salvar suas alterações e também não será realizada uma nova correção. Caso opte por não finalizar então não será realizada a correção e a próxima vez que abrir está atividade poderá continuar a fazê-lo. Estas informações sobre interações anteriores sobre a atividade, são solicitados ao LMS através de invocação de função JavaScript `GetInteraction` que se encontra disponível na página onde está a *applet*. O código do Quadro 12 apresenta como foi resolvido a comunicação entre *applet* e JavaScript.

```
public class ComunicacaoScorm{
    private DesenhaBlocoApplet applet;

    public ComunicacaoScorm(DesenhaBlocoApplet applet) {
        this.applet = applet;
    }

    public void retornaInformacoesAtividade(String cod, String valor) {
        String resposta = eval("AddInteraction('"+cod+"',
                          'long-fill-in','"+valor+"')");
        if (resposta.equals("invalid_id")) {
            String valor = eval("UpdateInteraction('"+cod+"', '"+valor+"')");
             ...
        }
        ...
    }

    public void recuperaRespostaAluno(String cod) {
        String resposta = eval("GetInteraction('"+cod+"')");
        ...
    }

    private String eval(String linhacomando) {
        Object o = JSObject.getWindow(this.applet).eval(linhacomando);
        if (o == null)
            return "";
        else
            return o.toString();
    }
}
```

Quadro 12 - Código de comunicação da *applet* com JavaScript

Para que fosse possível utilizar o objeto window do JavaScript e a sua função `eval()` é necessário possuir uma instancia da *applet.* Com isso os métodos `recuperaRespostaAluno` e `retornaInformacoesAtividade` montam chamadas a funções JavaScript definidas na página onde está a *applet* e chama o método `eval()`, que invocará a função `eval()` do objeto window executando a função passada como parâmetro.

3.2 SISTEMA TUTOR INTELIGENTE (STI)

Neste capítulo são apresentados os requisitos, diagramas da UML a implementação e exemplos de uso do STI.

3.2.1 Requisitos

O STI deverá:

a) disponibilizar uma interface para que o professor visualize o desempenho da turma e alunos (RF);
b) realizar avaliações constantes do aluno (RF);
c) permitir que o aluno visualize seu desempenho geral e por atividades (RF);
d) ser implementado no ambiente de desenvolvimento Netbeans 5.5 (Requisito Não-Funcional – RNF);
e) ser implementado em utilizando a linguagem Java (RNF);
f) ser uma aplicação *Web* (RNF);
g) utilizar o framework Java Server Faces (RNF);
h) utilizar o FuzzyJ Toolkit (RNF);
i) executar conteúdo SCORM (RNF).

3.2.2 Especificação

Nesta seção são apresentados o diagrama de caso de uso, a prototipação do sistema *fuzzy*, diagrama de classes, modelo de domínio e do aluno e diagrama de atividades.

3.2.2.1 Diagrama de casos de uso

Na Figura 1 apresenta o diagrama de casos de uso do STI.

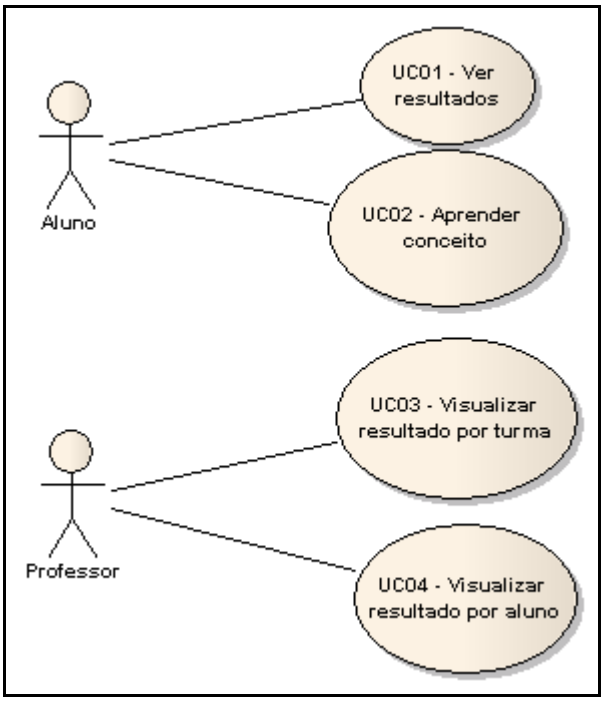


Figura 17 – Diagrama de casos de uso do STI

No Quadro 13 apresenta a descrição do caso de uso ver resultados.

UC01 – Ver resultados
Sumário: O aluno deseja visualizar seu desempenho nos conceito realizou.
Ator primário: Aluno.
Precondições: O aluno logado no sistema.
Fluxo Principal: 1. O aluno solicita ver resultados. 2. Sistema apresenta seus respectivos resultados no conceito em que realizou, e para cada atividade são apresentados o nível de dificuldade, o percentual de acertos e seu tempo de realização.

Quadro 13 – Descrição do caso de uso `UC01 – Ver resultados`

No Quadro 14 apresenta a descrição do caso de uso aprender conceito.

UC02 – Aprender conceito

Sumário: O aluno deseja aprender um conceito

Ator primário: Aluno.

Precondições: O aluno logado no sistema.

Fluxo Principal:
1. Aluno entra no sistema.
2. O sistema apresenta os conceitos que faltam o aluno aprender.
3. O aluno seleciona um conceito.
4. O sistema abre uma nova janela com o conteúdo sobre o assunto, assim como uma atividade ainda não realizada e de nível (fácil, médio e difícil), de acordo com o que o aluno já realizou anteriormente no conceito.
5. O aluno interage com o conteúdo e atividade.
6. O aluno encerra a interação.
7. O sistema fecha a janela.

Fluxo alternativo (1): Suspende a interação
a) No passo 6, o aluno não quer finalizar a interação, mas suspender para continuar em outro momento.
b) O sistema persiste o estado atual de interação.
c) Retorna ao passo 7.

Fluxo alternativo (2): Fazer atividade
a) No passo 6, caso seja uma atividade e tenha sido finalizada.
b) O sistema aplica a lógica fuzzy e grava o resultado obtido.
c) E define a próxima atividade a ser realizada pelo aluno.
d) Retorna ao passo 7.

Notas:
- o conteúdo e atividade são no modelo SCORM.
- a atividade é o objeto de aprendizagem (DAGAA) discutido na seção 3.1.

Pós-condições: Modelo do aluno atualizado.

Quadro 14 – Descrição do caso de uso `UC02 – Aprender conceito`

No Quadro 15 apresenta a descrição do caso de uso visualiza resultado por turma.

UC03 – Visualiza resultado por turma

Sumário: Ao professor visualizar o resultado de todos os alunos da turma, onde é

apresentado os conceitos que o aluno realizou e seu desempenho no mesmo.

Ator primário: Professor.

Precondições: O professor logado no sistema.

Fluxo Principal:
1. O professor solicita visualizar o resultado por turma.
2. O sistema apresenta as turmas.
3. O professor seleciona a turma desejada.
4. O sistema apresenta o resultado.

Quadro 15 – Descrição do caso de uso `UC03 – Visualiza resultado por turma`

No Quadro 16 apresenta a descrição do caso de uso visualiza resultado por aluno.

UC04 – Visualiza resultado por aluno
Sumário: Ao professor visualizar o resultado por aluno, onde são apresentado os conceitos e o desempenho obtidos nos conceitos realizados, e as atividades realizadas em cada conceito com as seguinte informações nível de dificuldade, os acertos e o tempo.
Ator primário: Professor.
Precondições: O professor logado no sistema.
Fluxo Principal: 1. O professor solicita visualizar o resultado por aluno. 2. O sistema apresenta os alunos. 3. O professor seleciona o aluno desejado. 4. O sistema apresenta o resultado.
Pós-condições: Professor ter noção de como anda o desempenho do aluno.

Quadro 16 - Descrição do caso de uso `UC04 – Visualiza resultado por aluno`

3.2.2.2 Prototipação do sistema *fuzzy*

Para projetar o sistema algumas variáveis foram definidas junto de seus conjuntos fuzzy. Os valores dos conjuntos foram definidos com teste realizados durante o processo de prototipação. Para o tempo de realização resolveu-se dividir o tempo proposto na atividade pelo tempo gasto para realização da atividade e multiplicá-lo por 10 (dez), pois se o resultado for 10 será tempo proposto pelo professor, sendo menor que 10 é um tempo inferior ao proposto e ser for maior será um tempo superior ao proposto.

O sistema *fuzzy* foi estruturado com três variáveis de entrada (complexidade, acertos e tempo) e duas variáveis de saída (desempenho e próximo nível), sendo adotado o modelo de inferência de Mamdani. A Figura 18 apresenta o modelo da solução adotada.

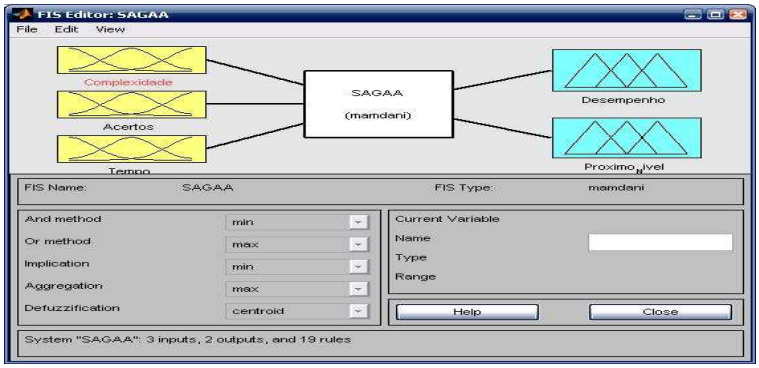


Figura 18 – Modelo de solução proposto

A variável **complexidade** define o nível de dificuldade da atividade, e pode assumir um dos três graus de pertinência fácil, médio ou difícil. A função de pertinência é apresentada na Figura 19.

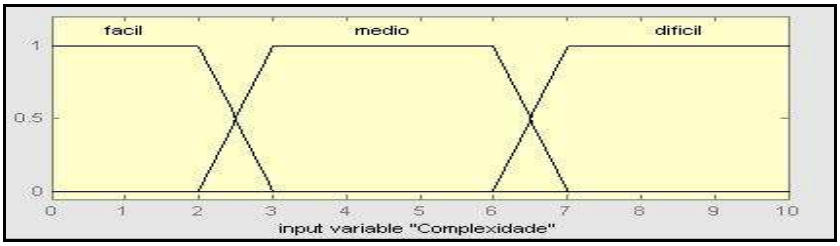


Figura 19 – Função de pertinência da variável complexidade

A variável **acertos** define o percentual de acertos que o aluno obteve na atividade em que realizou e podendo assumir os seguintes graus de pertinência baixo, razoável ou alto. Na Figura 20 é apresentada a função de pertinência desta variável.

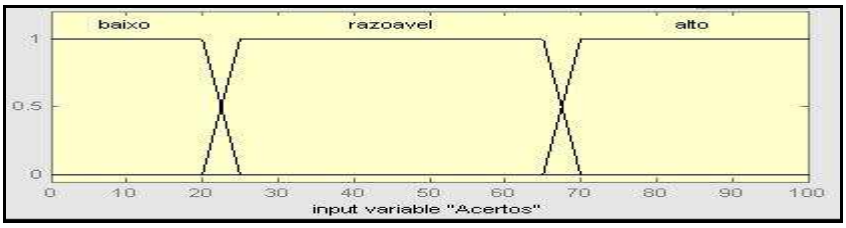


Figura 20 – Função de pertinência da variável acertos

A variável **tempo** define se o aluno desenvolveu a atividade dentro do tempo estimado pelo professor ou se obteve um tempo muito elevado. A fórmula para o cálculo da variável é mostrada no Quadro 17 e assume os seguintes graus de pertinência rápido, normal e devagar, conforme apresentado na Figura 21.

tempo = $\left(\frac{tr}{te}\right)$*10;

onde:

tr = Tempo gasto para realização da atividade.

te = Tempo estimado pelo professor para realização da atividade.

Quadro 17 – Calculo da variável tempo

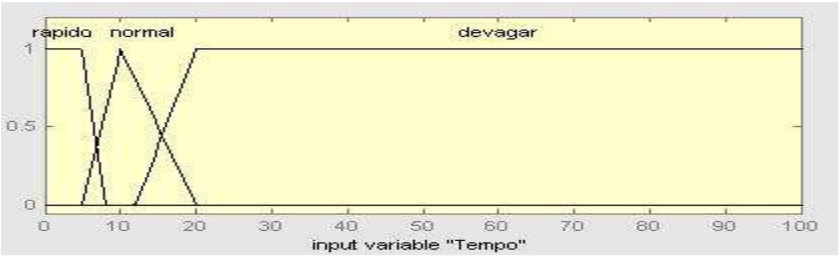


Figura 21 – Função de pertinência da variável tempo

A variável de saída **desempenho** define o conhecimento do aluno por conceito podendo assumir os seguintes graus de pertinência: desconhece, conhece pouco, conhece,

conhece bem, aprendeu e aprendeu bem, como apresentado na Figura 22.

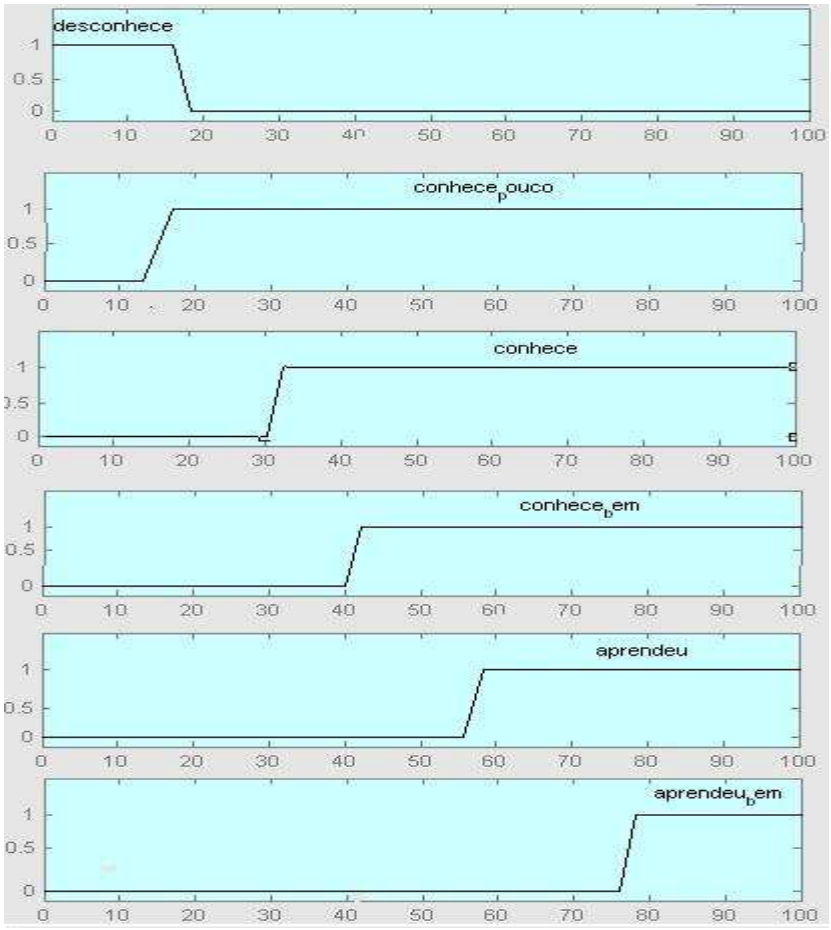


Figura 22 – Função de pertinência de variável desempenho

Pode-se observar que se o aluno aprendeu bem ele também aprendeu, conhece bem, conhece e conhece pouco, ou seja, pode possuir pertinência 1 (um) em mais de uma função de pertinência.

A variável de saída **próximo nível** é responsável por definir o próximo nível da atividade a ser realizada no conceito. Seu grau de pertinência é dado por fácil, médio e difícil. A função de pertinência desta variável é apresentada na Figura 23.

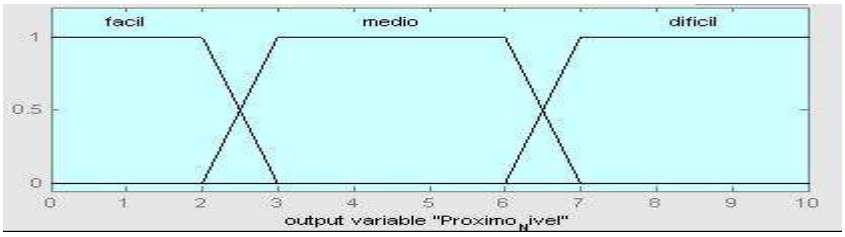


Figura 23 – Função de pertinência da variável próximo nível

As regras definidas são apresentas na Figura 24.

Complexidade	Devagar			Normal			Rápido			Tempo
	Baixo	Razoável	Alto	Baixo	Razoável	Alto	Baixo	Razoável	Alto	Acertos
Fácil	D / F	D / F	CP / F	D / F	D / F	C / M	D / F	CP / F	C / M	
Médio	CP / F	CP / F	C / M	CP / F	C / M	CB / D	CP / F	C / M	CB / D	
Difícil	C / F	CB / M	AB - D	C / F	A / D	AB - D	C / F	A / D	AB - D	

Desempenho
D - Desconhece
CP - Conhece Pouco
C - Conhece
CB - Conhece Bem
A - Aprendeu
AB - Aprendeu Bem

Próximo nível
F - Fácil
M - Médio
D - Difícil

Figura 24 - Tabela de regras

E para este modelo o Matlab gerou o arquivo mostrado no Anexo A, sendo utilizado como entrada para o mapeamento do sistema *fuzzy* deste trabalho.

3.2.2.3 Modelo de domínio e aluno

O modelo de domínio se utiliza de conteúdo e atividades. O conteúdo são atividades do modelo SCORM. Já as atividades são objetos de aprendizagem DAGAA como discutido na seção 3.1, onde também são armazenado o tempo estimado para realização e o nível de complexidade do mesmo.

O modelo de domínio é baseado na estrutura de conhecimento a ser ensinado. Este modelo está organizado em conceitos como apresentado na Figura 25, onde os conceitos C1 e C3 são pré-requisito para o conceito C2, ou seja, migrou para conceito C2 é necessário que tenha se atendido a condição em C1 e C3. Para seleção da atividade a ser apresentada no conceito, é levado em consideração o próximo nível de atividade. E cada conceito mantém uma lista de conteúdos e atividades.

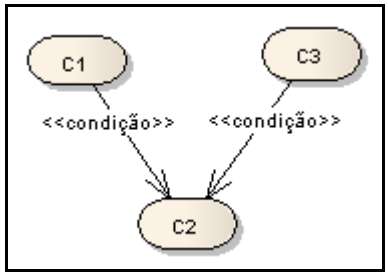


Figura 25 – Representação de conceitos

Para o ensino do gerenciador de armazenamento e arquivos foram definidos os seguintes conceitos para o ensino:

a) tipos e tamanho de campos (C1);

b) RAID (C2);
c) registros de tamanho fixo (C3);
d) registros com chaves primarias e chaves estrangeiras, onde inicia-se a ensino de ponteiros (C4);
e) registros com campos de tamanho variado (C5);
f) registros com valores verdadeiramente grandes os BLOB (C6);
g) registros de índices utilizando a estrutura de árvore B+ (C7).

Na Figura 26 apresenta a estrutura para o ensino dos conceitos do gerenciador de armazenamento e arquivos.

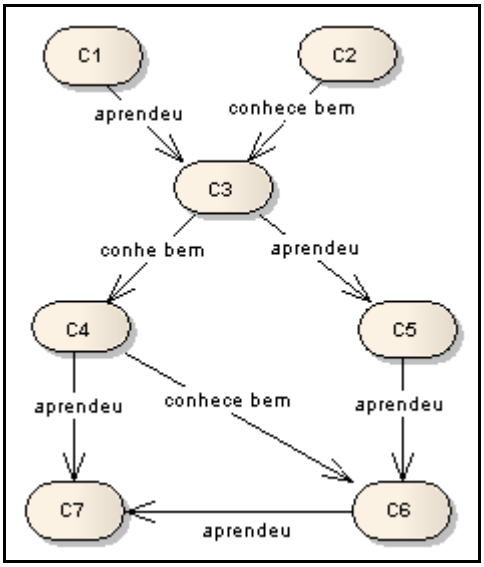


Figura 26 - Estrutura dos conceitos as serem ensinados

Para cada novo aluno na turma devem ser informados os conceitos que o mesmo iniciará. No modelo apresentado na Figura 26 devem ser informados os conceitos C1 e C2 que são os que não possuem pré-requisitos para serem iniciados, os demais conceitos serão atingidos conforme o aluno ir adquirindo o conhecimento. A evolução no conhecimento é determinada pela realização das atividades, que com base nas regras *fuzzy* definidas conclui-se o desempenho do aluno.

As atividades disponibilizadas ao aluno são por conceito, e podem ser de três níveis diferentes: fácil, médio e difícil. Este nível é definido conforme o próximo nível alcançado pelo aluno no conceito.

O modelo do aluno é uma instância do modelo de domínio, inicializada como desconhece para todos os conceitos e próximo nível como fácil. À medida que o aluno interage com o sistema o nível de conhecimento e próximo nível nos conceitos vão se atualizando, como exemplificados na Figura 27.

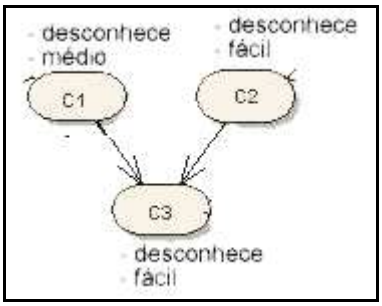


Figura 27 – Exemplo de um modelo de aluno

3.2.2.4 Diagrama de classes

Na Figura 28 são apresentados os principais pacotes de classes do STI.

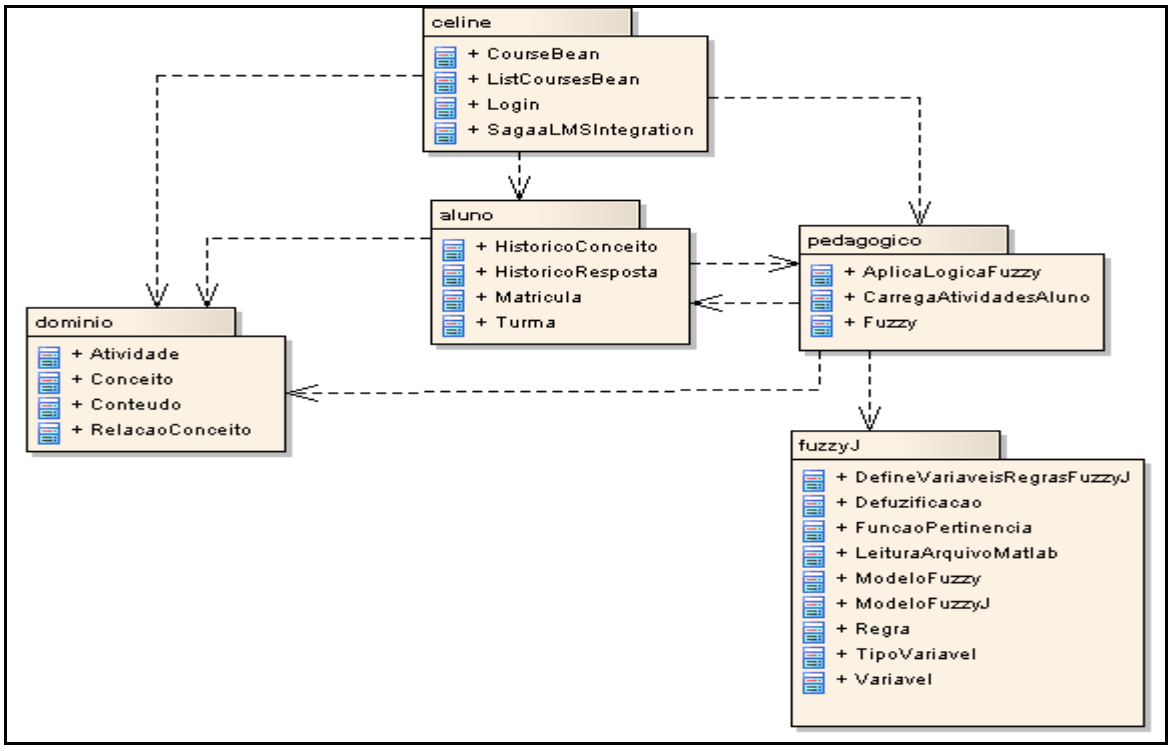


Figura 28 – Principais pacotes de classe do STI

No pacote `dominio` contém as informações sobre o conhecimento a ser ensinado. No pacote `aluno` são registradas informações de cada estudante. No pacote `pedagogico` é onde são decidas quais atividades serão expostas. O pacote `fuzzyJ` é responsável pela lógica *fuzzy* utilizada no pacote `pedagogico` para decisão do que expor. No pacote `celine` contém as classes necessárias para integração com o componente CELINE.

A Figura 29 apresenta o diagrama de classes do domínio da aplicação mostrando como os conceitos estão relacionados com outros conceitos e como as atividades e conteúdos estão relacionados ao conceito.

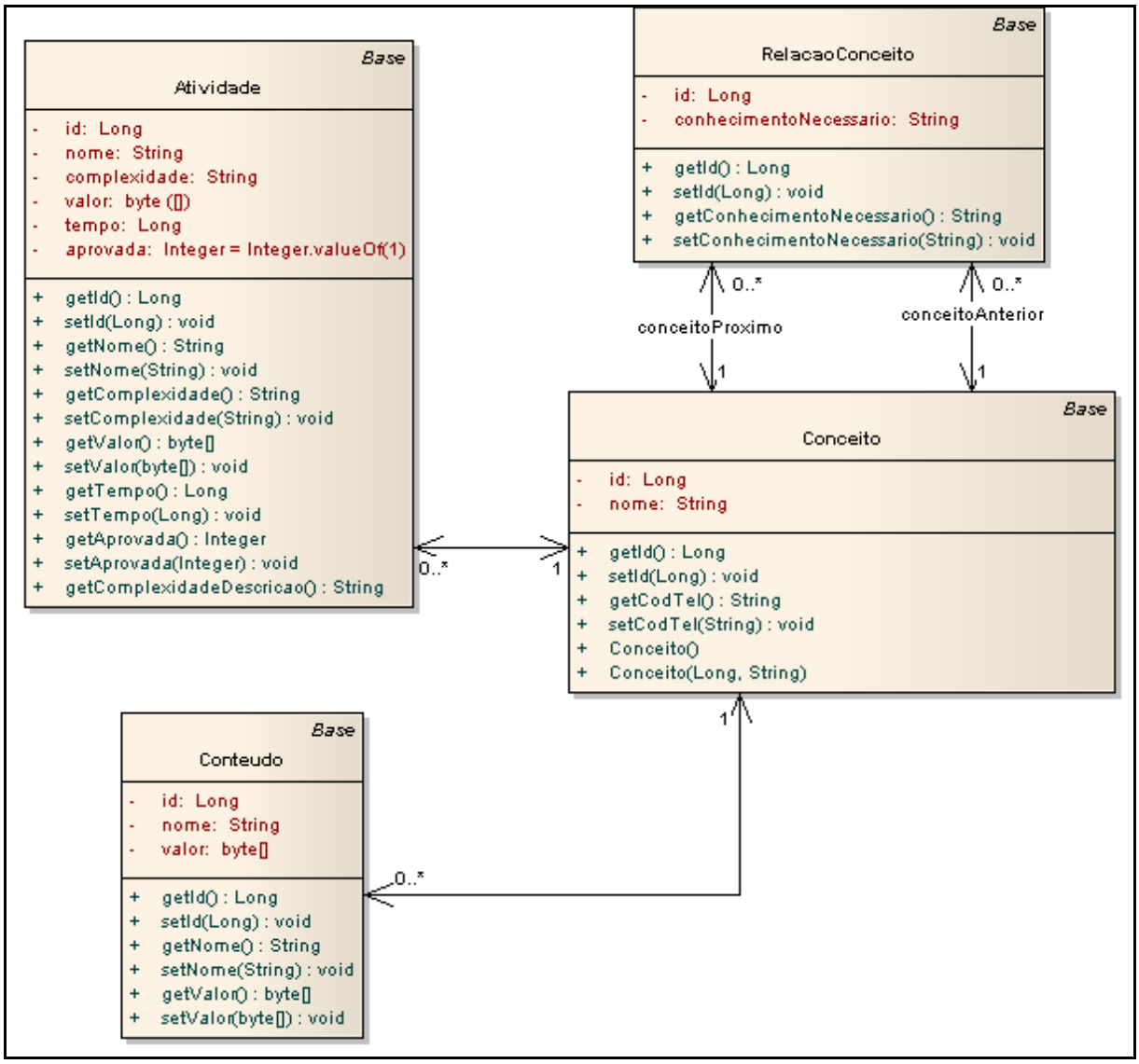


Figura 29 – Diagrama de classes do domínio do STI

A classe `Atividade` é onde são armazenadas as atividades cadastradas pelo professor sendo que cada atividade é relacionada a um conceito. A classe `Conteudo` é onde são armazenados os conteúdos relacionados a um conceito. A classe `Conceito` é onde são armazenados os conceito a serem repassados para o aluno, a classe `RelacaoConceito` é utilizada para relacionar os conceitos e definir qual o nível de desempenho esperado para que o aluno inicie no próximo conceito.

Na Figura 30 é apresentado o diagrama de classes do modelo do aluno, mostrando como são organizadas as informações sobre o aluno.

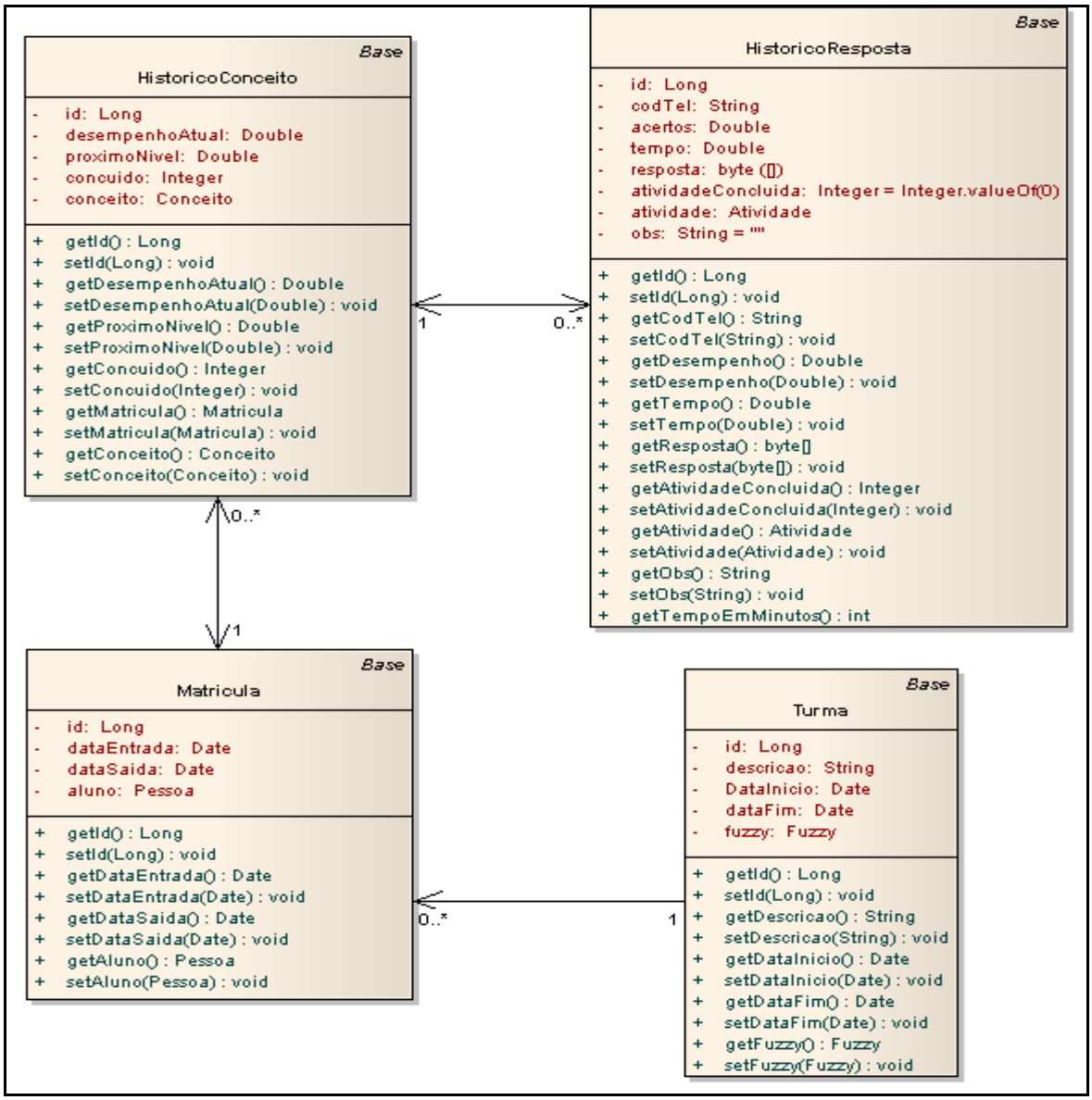


Figura 30 – Diagrama de classes do modelo do aluno

A classe `Turma` é responsável por manter um agrupamento dos alunos e possui o modelo *fuzzy* que será utilizado a estes alunos. A classe `Matricula` é o vinculo do aluno na turma e com os conceitos a serem ensinado ao mesmo. Na classe `HistoricoConceito` é mantido os conceito pertencentes a uma matrícula onde cada conceito contém o desempenho e o próximo nível que o aluno obteve neste conceito e se conseguiu concluir o conceito, ou seja, se atingiu o desempenho necessário a todos os próximos conceitos relacionados a este. A classe `HistoricoResposta` é onde são mantidas as atividades que o aluno veio a realizar no conceito, e contém o percentual de acertos, o tempo gasto para a realização, a resposta do aluno (esta informação apenas o objeto de aprendizagem conhece) e se atividade foi concluída.

3.2.2.5 Diagrama de atividades

Na Figura 31 é apresentado o diagrama de atividades do modelo pedagógico para atualizar o modelo do aluno.

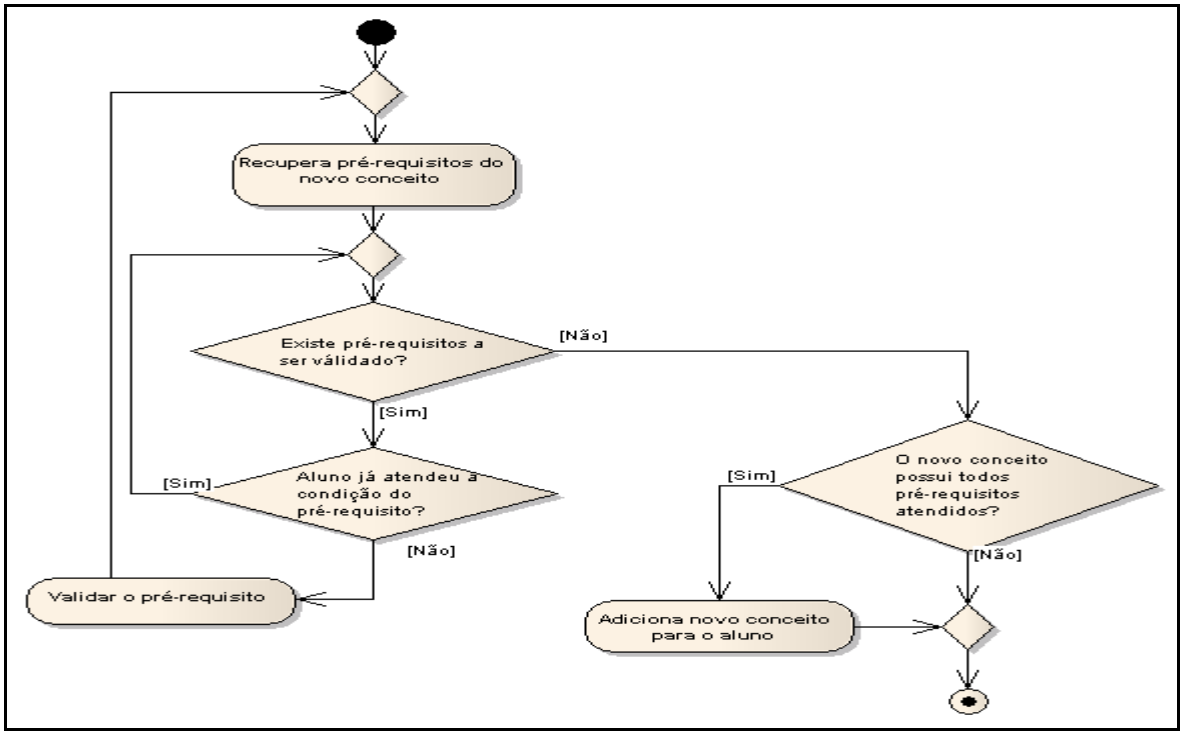


Figura 31 – Diagrama de atividade do modelo pedagógico

Inicialmente é recebido o conceito atual (conceito que está sendo realizado pelo aluno) e o novo conceito (conceito que atendeu a condição). Em seguida são recuperados todos os conceitos que são pré-requisitos deste novo conceito. Caso este novo conceito não possua pré-requisito será adicionado ao modelo do aluno.

Mas se este novo conceito possuir pré-requisitos, vai ser verificado se o aluno já possui este conceito, se possuir volta a verificar se existem mais pré-requisitos. Caso o aluno não possua o pré-requisito ainda no seu modelo, será feita uma chamada recursiva para este método passando o pré-requisito (como sendo o novo conceito), e validando se o este conceito não possui pré-requisitos não atendidos. Ao termino será verificado se todos os pré-requisitos foram atendidos, e se foram adiciona o novo conceito para o aluno.

3.2.3 Implementação

O STI batizado de SAGAA foi desenvolvido utilizando a linguagem Java JDK 1.6

através da plataforma *Netbeans* 5.5, e os componentes *FuzzyJ* para definir a inteligência do tutor e CELINE para organizar do conteúdo e atividades apresentados ao aluno.

3.2.3.1 Inteligência do sistema

A inteligência do STI é baseada num sistema *fuzzy*, tanto para verificar o desempenho do aluno como para decidir qual será a próxima atividade a disponibilizar para o mesmo.

As configurações do sistema *fuzzy* são realizadas com base no arquivo gerado pela ferramenta *Matlab*. Neste arquivo pode-se alterar o valor do grau de pertinência e a função de pertinência (as funções de pertinências suportadas são trimf, trapmf, zmf, smf, sigmf, gaussmf), e adicionar, alterar e remover regras, onde o único requisito é que as mesmas devem sempre serem unidas pelo operador "*and*".

O processo de fuzzificação é o mapeamento de números discretos em conjuntos *fuzzy*, onde estes conjuntos foram definidos como uma função de pertinência triangular, como apresentado na Figura 32. Onde é possível observar que o número discreto possui pertinência 1 (um). Cada uma das três variáveis de entrada são mapeadas para a função de pertinência triangular, a realização deste mapeamento no FuzzyJ é apresentado no Quadro 18.

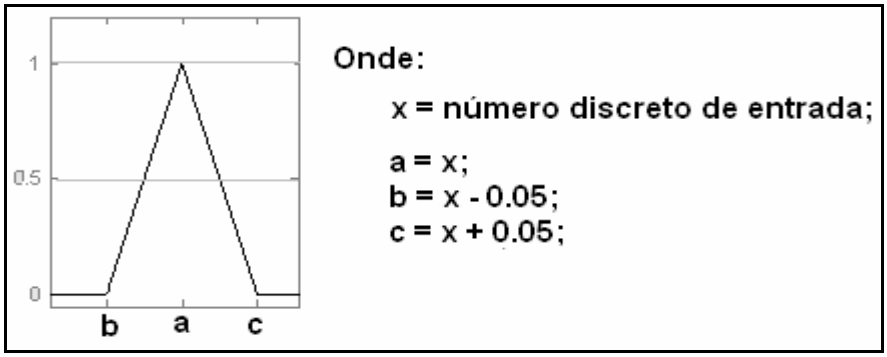


Figura 32 - Exemplo função de pertinência utilizada para fuzzificação

```
FuzzyValue umFuzzyValue = null;
try{
  if("Complexidade".equalsIgnoreCase(nomeVariavel)){
      umFuzzyValue = new FuzzyValue(variaveisFuzzy.get(nomeVariavel),
      new TriangleFuzzySet(complexidade-0.05, complexidade, complexidade + 0.05));
   }else if("Acertos".equalsIgnoreCase(nomeVariavel)){
      umFuzzyValue = new FuzzyValue(variaveisFuzzy.get(nomeVariavel),
      new TriangleFuzzySet(acertos -0.05, acertos, acertos + 0.05));
   }else if("Tempo".equalsIgnoreCase(nomeVariavel)){
      umFuzzyValue = new FuzzyValue(variaveisFuzzy.get(nomeVariavel),
           new TriangleFuzzySet(tempo -0.05, tempo, tempo + 0.05));
   }
}catch (Exception e){
   System.err.println("Erro ao definir variaveis de Leitura: "+e);
}
```

Quadro 18 – Rotina de fuzzificação

O processo de inferência é o responsável por avaliar as variáveis de entrada aplicando as regras da base de conhecimento e atribuir resposta ao processamento, este processo é apresentado no Quadro 19.

```
FuzzyValue[] resultado = null;
FuzzyValue[] resultadoAux = null;
FuzzyValueVector fuzzyValueVetorAux = null;
for(String nomeRegra : listaNomeRegra){
   FuzzyRule umaRegraFuzzy = regrasFuzzy.get(nomeRegra);
   umaRegraFuzzy.removeAllInputs();//Limpa as leituras existentes
   FuzzyValue[] antecedentes = umaRegraFuzzy.getAntecedents().toFuzzyValueArray();
   //para cada antecedente adiciciona a leiura
   for (int i = 0; i < antecedentes.length; i++) {
      umaRegraFuzzy.addInput(variaveisLeituraFuzzy.get(
                     antecedentes[i].getFuzzyVariable().getName()));
    }
    //Verifica se a regra se aplica
    try{
      if(umaRegraFuzzy.testRuleMatching()){
        fuzzyValueVetorAux = umaRegraFuzzy.execute();
          if(resultado == null){
            resultado = new FuzzyValue[fuzzyValueVetorAux.size()];
            for (int i = 0; i < resultado.length; i++) {
               resultado[i] = null;
            }
            resultadoAux = new FuzzyValue[fuzzyValueVetorAux.size()];
          }
          for (int i = 0; i < fuzzyValueVetorAux.size(); i++) {
             resultadoAux[i] = fuzzyValueVetorAux.fuzzyValueAt(i);
             if(resultado[i] == null){
                resultado[i] = resultadoAux[i];
             }else{
                 resultado[i] = resultado[i].fuzzyUnion(resultadoAux[i]);
             }
          }
       }
    }catch (Exception e){
       System.err.println("Erro ao aplicar Regras FuzzyJ: "+e);
    }
}
```

Quadro 19 – Rotina de inferência *fuzzy*

Nesse processo são percorridas todas as regras que foram cadastradas. Para cada regra são adicionados os valores dos antecedentes e é verificada se a regra é atendida. Se a regra foi atendida, e sendo a primeira regra atendida para esta inferência, apenas armazena o resultado obtido das duas variáveis de saída, caso não seja a primeira regra é realizada uma união entre o valor *fuzzy* armazenado e o valor *fuzzy* obtido das duas variáveis de saídas desta regra.

Após o processo de inferência, obtém-se um conjunto *fuzzy* como resposta. Neste caso não é conveniente como resposta final do sistema, sendo necessário uma representação numérica e este processo é chamado de defuzzificação. Nesta parte final do processo são retornados o elemento central da área definida pelo conjunto fuzzy, e o primeiro elemento entre aqueles que possuírem o maior grau de pertinência. A defuzzificação é demonstrada no Quadro 20.

```
double central = 0;
 Defuzificacao defuzificacao = null;
try {
  // calculate the deffuzified value
  central = resultado[i].momentDefuzzify();
  resultado[i] = resultado[i].fuzzyNormalize();
  double maxY = resultado[i].getMaxY();
  Double primeiroMax = resultado[i].getXforMembership(maxY);
  Defuzificacao = new Defuzificacao(Double.valueOf(central), primeiroMax, null);

 retorno.put(resultado[i].getFuzzyVariable().getName().toUpperCase(),defuzificacao);
} catch (Exception ex) {
   ex.printStackTrace();
}
```

Quadro 20 – Rotina de defuzzificação

A classe `Defuzificacao` é utilizada para retornar o valor médio e primeiro valor com maior pertinência da defuzzificação, e esta é armazenada em um `java.util.Map` onde a chave é o nome da variável de saída.

3.2.3.2 Sistema tutor

Esta seção apresenta o comportamento dos módulo integrados junto do módulo de interface onde são utilizados.

As *tags* fornecidas pelo CELINE não foram compatíveis com a tecnologia Java Server Faces, por essa razão fez-se necessário a utilização de *tags* do Java Server Faces para realizar a listagem dos conceitos a serem apresentados aos alunos.

A conexão utilizada pelo CELINE foi configurada no arquivo `celine-config.xml` onde foi informado o *driver* de conexão, a URL, usuário e senha do banco de dados utilizado, no Quadro 21 é apresentado o código do `celine-config`. No arquivo `WEB-INF\web.xml` foram adicionadas algumas linhas obrigatórias em toda aplicação, pois é neste documento que é definido tudo que o servidor precisa conhecer sobre sua aplicação.

```
<config>
    <courses-folder>courses</courses-folder>
    <error-page>error.do</error-page>
    <lmsIntegration>sagaa.celine.SagaaLMSIntegration</lmsIntegration>
    <database-source>
        <rdb>
            <driver>org.postgresql.Driver</driver>
            <url>jdbc:postgresql://localhost:5432/CELINE</url>
            <user>jeanbnu</user>
            <password>123456</password>
        </rdb>
    </database-source>
</config>
```

Quadro 21 - Código do `celine-config.xml`

Como o CELINE necessita de um usuário autenticado, após ter sido realizada a autenticação no SAGAA também é realizada a autenticação no CELINE. Para a autenticação no CELINE foi implementado, uma classe conforme o Quadro 22. Após a autenticação no SAGAA é invocado o método da classe `Login,` como pode ser visto no trecho de código no Quadro 23.

```
public class Login {
    public static boolean logIn(HttpServletRequest request, String name,
                                String passw) throws Exception {
        DAO dao = LMSConfig.getInstance().getDAO();
        User user = dao.login(name, passw);
        if (user != null) {
            UserAdministration.setUser(request, user);
            return true;
        }
        return false;
    }
}
```

Quadro 22 – Código que efetua `login` no componente CELINE

```
...
private boolean autenticaCeline(){
        boolean retorno = true;
        try {
            if (sagaa.celine.Login.logIn(getRequest(), login, senha) == false) {
                retorno = false;
            }
        } catch (Exception ex) {
            retorno =  false;
            ex.printStackTrace();
        }
        return retorno;
    }
...
```

Quadro 23 - Código que invoca método login

Para o módulo de interface foi utilizado o componente CELINE para permitir a apresentação de pacotes SCORM com um mecanismo para integração e apresentação com o sistema. Para interagir com o sistema é necessário desenvolver uma classe que realize a interface `LMSIntegration,` que foi chamada de `SagaaLMSIntegration.` O código é apresentado no Quadro 24.

```
@Override
public void selectCourses(User user, List<Course> courses) throws Exception {
   courses.clear();
   //Recupera os conceitos que o aluno pode visualizar
     ...
}

@Override
public ContentPackage openContentPackage(User user, String courseId)
       throws Exception {
   EasyContentPackage easy = new EasyContentPackage();
   LMSConfig lms = LMSConfig.getInstance();
   //Carregar da Tabela de conteúdo para este curso
     ...
   //Carregar Atividade deste curso
     ...
    return easy.build(title, courseId);
}

@Override
public void changeCourse(User user, ActivityTree course) throws Exception {
}

@Override
public boolean processInteraction(User user, Course course, int index, String type,
               String value) throws Exception {
    //Gravar informações recebidas da applet
       ...
    return true;
}

@Override
public String retrieveInteraction(User user, Course course, int index, String type)
             throws Exception {
   //Retorna informações solicitadas pela applet
       ...
   return null;
}
```

Quadro 24 - Código da classe `SagaaLMSIntegration`

Para apresentar os conceitos que o aluno pode acessar, foi desenvolvida a página do Quadro 25, que acessa o *form bean* `ListCoursesBean`. Para listar os conceitos o CELINE utiliza o método `selectCourses()` de `SagaaLMSIntegration`. A lista inicial de cursos é de acordo com a tabela `courses` do próprio componente. Porém, quem define quais cursos serão apresentados é o STI .

```
<h:dataTable value='#{ListCoursesBean.itens}' var='item' border="0" cellpadding="2"
    cellspacing="0" width="100%"rowClasses="primeiro,ultimo">
   <h:column>
      <h:outputLink value="conteudo.jsp?id=#{item.courseId}" target="_blank">
                  <h:outputText value="#{item.title}"/>
      </h:outputLink>
      </h:column>
</h:dataTable>
```

Quadro 25 - Página que lista os cursos

Quando o aluno seleciona um dos conceitos, o CELINE solicita ao SAGAA quais os conteúdos e atividades que o aluno poderá ter acesso através do método `openContentPackage()`.

Caso o usuário esteja interagindo com atividade os métodos `processInteraction()` e

`retrieveInteraction()`. O método `retrieveInteraction()` é utilizado para continuar uma atividade suspensa. O método `processInteraction()` é chamado tanto quando a atividade for suspensa quanto encerrada. A diferença que no encerramento da atividade além do que foi realizado é passado um parâmetro informando que atividade foi encerrada.

Com o encerramento da atividade os valores de tempo e acertos, juntamente com o nível de dificuldade da atividade, realiza-se o processo de fuzzificação e a inferência *fuzzy*. E como saída do processo se tem conjuntos fuzzy que são defuzzificados para número discreto. A inferência da lógica *fuzzy* depende da turma em que o aluno foi matriculado, pois é na turma que está definida qual é o grau de pertinência de cada variável e as regras utilizadas.

No Quadro 26 mostra o método de seleção dos próximos conceitos. O método recebe o conceito a validar e a matrícula do aluno. Deste conceito são percorridos todos os seu conceitos anteriores (conceitos pré-requisitos do conceito que está sendo avaliado). Caso o conceito anterior já esteja vinculado a matrícula do aluno e o mesmo ainda não tenha sido concluído, o conceito que está sendo avaliado é marcado como pendente. No caso o aluno ainda não possuir o conceito anterior, o conceito que está sendo avaliado é marcado como pendente e é feita uma chamada para este mesmo método só que passando o conceito anterior. Ao término de percorrer os conceitos anteriores é verificado se o conceito validado não foi marcado como pendente, e neste caso vincula a matrícula do aluno.

```
//Valida pré-requisito de conceitos
private void validaECriaConceito(Conceito proxConceito, Long matriculaID){
boolean conceitoPendente = false;
List<RelacaoConceito> conceitosAnteriores =
     new ArrayList<RelacaoConceito>(proxConceito.getConceitoAnterior());
        for (RelacaoConceito umRC : conceitosAnteriores) {
            Conceito conceitoAnt =
                 ConceitoDAO.recuperaPorID(umRC.getConceitoAnterior().getId());
            List<HistoricoConceito> listaHIC =
                   HistoricoConceitoDAO.recuperaHistoricoConceitoPorMatriculaID
                   (matriculaID, null, conceitoAnt.getId());
            if(listaHIC != null && !listaHIC.isEmpty()){
                if(listaHIC.get(0).getConcuido().equals(Integer.valueOf(0))){
                    conceitoPendente = true;
                }
            }else{
                conceitoPendente = true;
                validaECriaConceito(conceitoAnt, matriculaID);
            }
        }
        if(!conceitoPendente){
            HistoricoConceito.criaHistoricoConceito(matriculaID, proxConceito);
        }
    }
```

Quadro 26 – Rotina de seleção de conceitos

A seleção das atividades do conceito é realizada quando não houver nenhuma atividade em aberto no conceito e este não esteja finalizado são recuperadas as atividades ainda não realizadas pelo aluno, levando em consideração o próximo nível da atividade. Caso

seja retornada mais de uma atividade é selecionada aleatoriamente uma delas. No Quadro 27 é apresentado o método onde é selecionado a próxima atividade a ser realizada no conceito.

```
private Atividade localizaProximaAtividade(Long conceitoID, TipoComplexidade
       complexidade){
     Random generator = new Random();
     Atividade atividade = null;
     List<Atividade> lista = AtividadeDAO.
          querySelAtividadeAindaNaoRealizadaNaMatricula(matriculaID,
              complexidade.toString(), conceitoID, Integer.valueOf(1), 1);
      if(lista != null && !lista.isEmpty()){
          int qtd = lista.size();
          int tentativas = 0;

          int index = 0;
          if(qtd > 1){
              index = generator.nextInt(qtd);
              index = index == qtd ? index -1 : index;
          }
          atividade = lista.get(index);
      }
      return atividade;
    }
```

Quadro 27 – Definição da próxima atividade

A apresentação dos conteúdos e atividades é de responsabilidade do `LMSIntegration`, onde a responsabilidade do STI é informar quais conteúdos e atividades serão apresentadas. Quantos aos conteúdos apresentados serão todos que estiverem relacionados ao conceito, já as atividades serão as já realizadas pelo aluno e se houver a atividade a ser realizada.

3.2.4 Operacionalidade

Nesta seção são apresentadas a interface do usuário do SAGAA.

Para iniciar deve-se informar o *login* e senha como apresentado na Figura 33.

Figura 33 – Página de autenticação no SAGAA

Conforme o perfil cadastrado para este usuário será montado o menu.

O professor pode visualizar o desempenho de seu alunos de duas formas. Uma onde será mostrado do desempenho do aluno individualmente contendo o desempenho no conceitos e todas as atividade realizadas pelo aluno, com seu respectivos desempenhos. E a outra forma

é a visualização dos alunos na turma, onde apresenta todos os alunos matriculados na turma e seu desempenho por conceito não contém quais atividades foram realizadas.

Na Figura 34 apresenta a página para pesquisar alunos para visualizar o desempenho do aluno desejado.

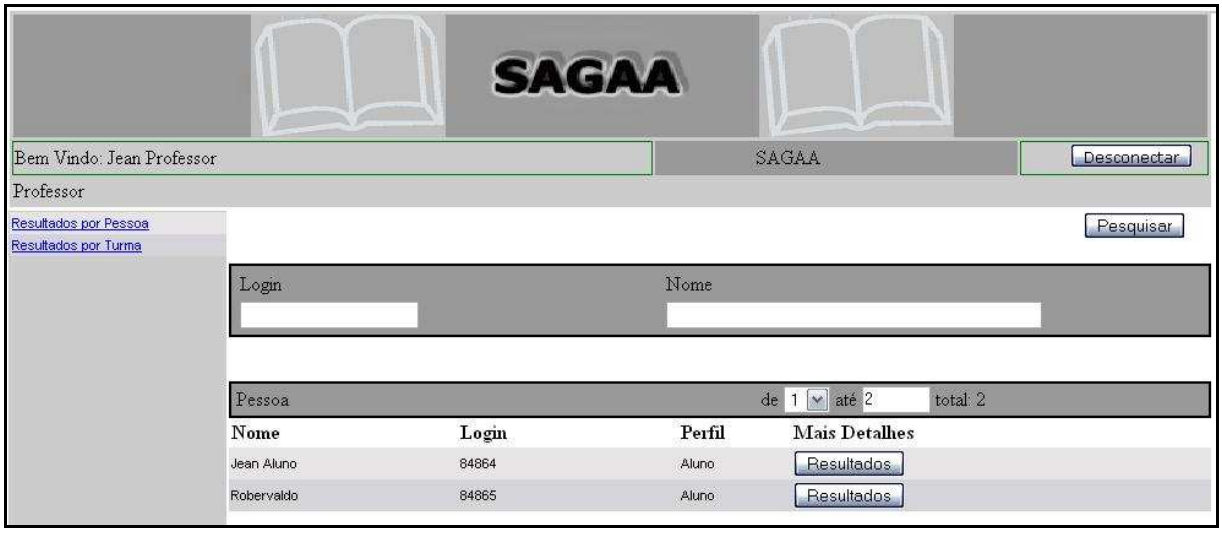


Figura 34 – Página de pesquisa de alunos para resultado por aluno

Nesta página pode ser adicionado o filtro pelo *login* ou nome do aluno. Após clicar no botão *Pesquisar*. São apresentados os alunos conforme o filtro informado. Para visualizar o desempenho do aluno deve-se clicar no botão *Resultados* do aluno desejado e será apresentada uma nova página com estes resultado. Como mostrada na Figura 35.

Figura 35 – Página com o resultado de um aluno

Esta página apresenta o nome do aluno que foi selecionado, a turma que se refere os conceitos e para o conceito ainda apresenta o desempenho obtido. Para cada conceito lista todas as atividades realizadas e para aquelas que ainda não foram concluídas apresenta a mensagem “Atividade não Concluída”. Para as demais atividades apresenta a complexidade, acertos e o tempo que o aluno obteve para realização.

Na Figura 36 apresenta a página de pesquisa de turma para visualizar o resultado da turma.

Figura 36 - Página de pesquisa de turmas

Pode ser inserido o filtro descrição e clicando em *Pesquisar* são apresentadas as turma do professor logado. Para visualizar o resultados deve-se clicar no botão *Ver Resultados* da turma desejada. A página com os resultados por turma é mostrada na Figura 37.

Figura 37 – Página de resultados obtidos na turma

Esta página apresenta a turma selecionada e todos os alunos da turma. Para cada aluno são informados os conceitos que o mesmo tem permissão e o desempenho atingido.

Para o perfil do aluno é possível visualizar os resultados obtidos, selecionar o conceito que deseja estudar e para cada conceito, visualizar o conteúdo fornecido. A página de resultados é apresentada na Figura 38.

Figura 38 - Página de resultado visualizada pelo aluno

Ao selecionar um conceito para ser estudado é aberta uma nova janela, onde são listados no menu os conteúdos e atividades a serem realizadas. Na Figura 39 é apresentada página com o conteúdo do conceito selecionado.

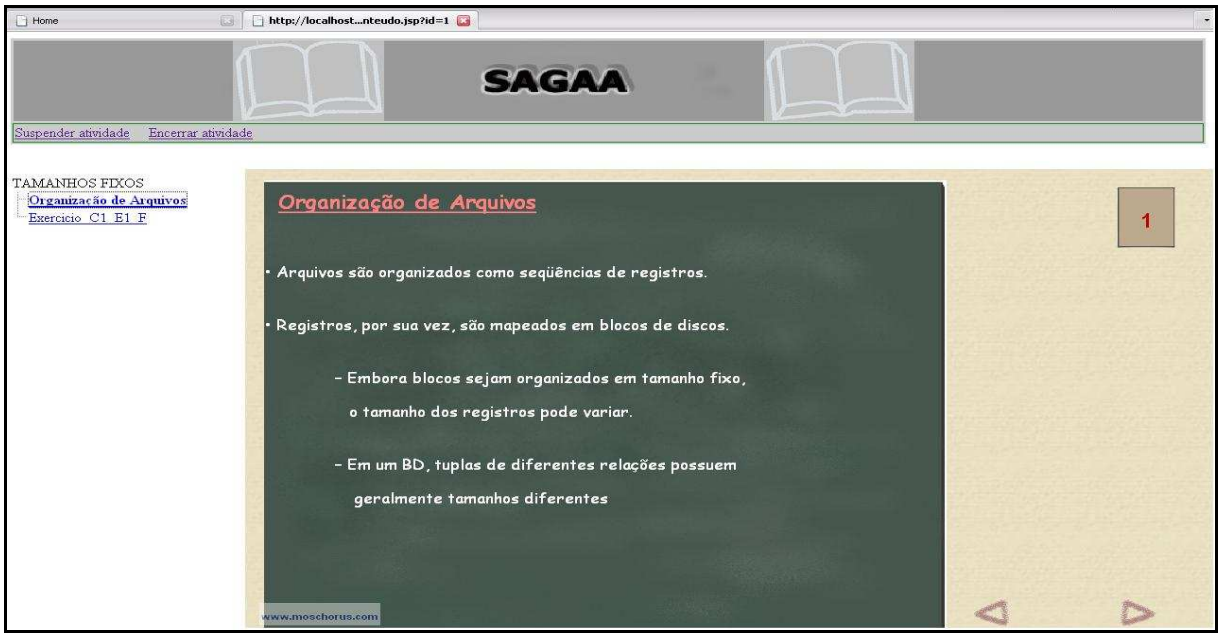


Figura 39 - Página com o conteúdo do conceito selecionado

Na Figura 40 é mostrada uma atividade a ser realizada pelo aluno no conceito em que o mesmo selecionou.

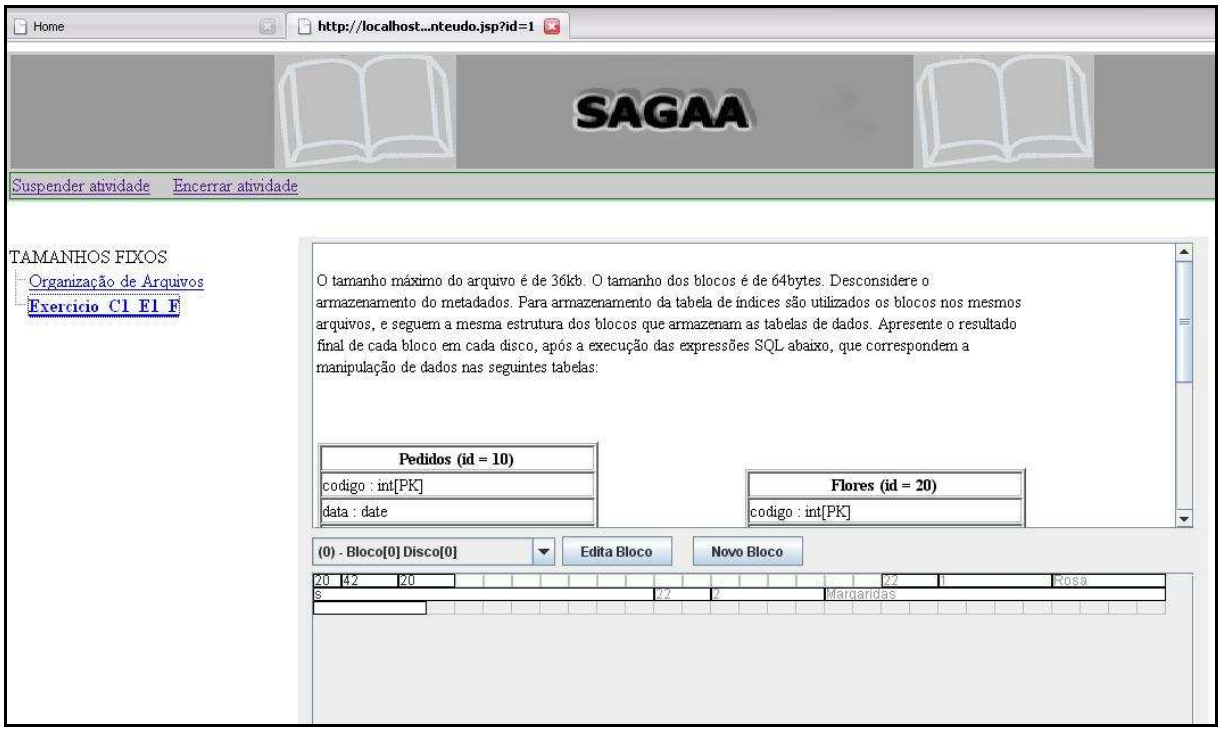


Figura 40 - Página com a atividade a ser realizada pelo aluno

Para suspender ou encerrar o estudo existem dois link na parte superior da página. Ao clicar em um desses *links* a janela será fechada. O *link* “Suspender atividade” possibilita a continuação da atividade posteriormente, e o link “Encerrar atividade” finaliza atividade e o sistema irá definir uma nova atividade ou habilitar novos conceitos dependendo do desempenho obtido.

3.3 ESTUDO DE CASO

Para demonstrar a funcionalidade do STI e o funcionamento do objeto de aprendizagem desenvolvidos no presente trabalho, foi construído um estudo de caso com três conceitos que são relacionados ao gerenciador de armazenamento e arquivo de um SGBD. Foram utilizados quatro perfis diferentes de aluno: um que acerta menos de 10%, outro que acerta 35%, um que acerta 50% e um que acerta 100%.

3.3.1 Os conceitos

Para o estudo de caso foi criada uma estrutura de conceitos conforme a Figura 41.

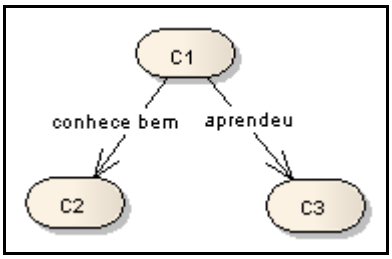


Figura 41 – Relacionamento entre os conceitos

O conceito C1 refere-se aos registros de tamanho fixo; C2 aos registros de tamanho variado; e C3 aos campos de chaves estrangeiras.

O aprendizado inicia com o conceito C1. O conceito C2 fica disponível quando o aluno tiver atingido o desempenho “conhece bem” do C1. O conceito C3 é apresentado quando o aluno obtiver o desempenho “aprendeu” no conceito C1.

3.3.2 As Atividades

Foram criadas duas atividades de cada nível para cada conceito.

No conceito C1 foram definidas duas atividades para o nível fácil com o tempo estimado de 15 minutos para serem resolvidas pelo aluno e só possuem comandos de inserção. No nível médio as atividades foram previstas com um tempo de 18 minutos e possuem comandos de inserção e alteração. No nível difícil as atividades devem consumir 20 minutos e possuem comandos de inserção, alteração e exclusão.

No conceito C2 as atividades definidas para o nível fácil devem consumir um tempo de 15 minutos e somente com comandos de inserção. O nível médio contém atividades são com o tempo de 20 minutos e com comandos de inserção e exclusão. No nível difícil são as atividades com o tempo de 25 minutos e com os comandos de inserção, alteração e exclusão.

Para o conceito C3 foram criadas duas atividades do nível fácil com o tempo de 15 minutos e um campo referenciando registros de outra tabela. No nível médio foram duas atividades com o tempo de 20 minutos e dois campos referenciando registros de outra tabela. No nível difícil as atividades prevêem 25 minutos e mais de dois campos referenciando registros de outra tabela.

No Anexo B encontram-se os enunciados de cada atividade.

3.3.3 Os alunos

Para que o aluno possa executar as atividades deve-se matriculá-lo na turma, informando quais serão os conceitos iniciais (conceitos que não possuem nenhum pré-requisito para ser realizado).

Para exemplificar são cadastrados quatro alunos, onde cada um possuirá um papel diferente ao desenvolver as atividades, sendo que todos iniciarão pelo conceito C1.

No Quadro 28 são apresentadas as atividades realizadas pelo aluno que acerta menos de 10%, onde o mesmo obteve menos de 10% de acertos na 2 (duas) atividade de nível fácil que realizou.

Exercício	Acertos	Tempo	Complexidade	Desempenho	Próximo nível
C1 F1	baixo	normal	fácil	desconhece	fácil
C1 F2	baixo	normal	fácil	desconhece	fácil

Quadro 28 - Atividades realizadas pelo aluno que acertou menos de 10%

O aluno realizou as duas atividades de nível fácil e não obteve os resultados para que seja selecionada a atividade de nível médio. E assim não havendo mais atividades deste nível cadastradas para este conceito, o STI não poderá selecionar uma nova atividade. Na Figura 42 é apresentado o modelo do aluno após a realização destas atividades. O aluno não atendeu a condição para iniciar em C2 ou C3, por isto seu modelo é apenas C1 não sendo gravado as informações sobre os conceitos C2 e C3.

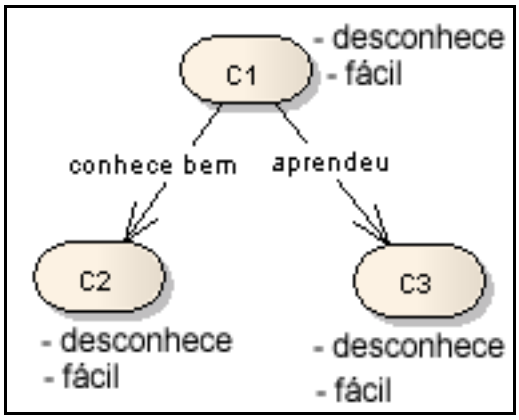


Figura 42 – Modelo do aluno que acerta pouco

No Quadro 29 são apresentados os resultados obtidos pelo aluno que acerta 35%.

Exercício	Acertos	Tempo	Complexidade	Desempenho	Próximo nível
C1 F1	alto	normal	fácil	conhece	médio
C1 M1	razoável	normal	médio	conhece	médio
C1 M2	razoável	normal	médio	conhece	médio

Quadro 29 - Atividades realizadas pelo aluno que acertou 35%

O aluno realizou a atividade de nível fácil e obteve médio como próximo nível, onde foram realizadas as duas atividades de nível médio cadastradas, obtendo um percentual aproximado 35% de acertos em cada uma destas atividades (C1 M1 e C1 M2), e com isso não conseguiu obter um desempenho para o próximo nível. Não havendo mais atividades cadastradas para o próximo nível atingido pelo aluno, o STI não selecionará uma nova atividade. A Figura 43 apresenta o modelo do aluno após a realização das atividades. Este aluno também não atingiu a condição necessária para que fosse apresentado os conceitos C2 e C3 e assim gravando informações sobre estes conceitos.

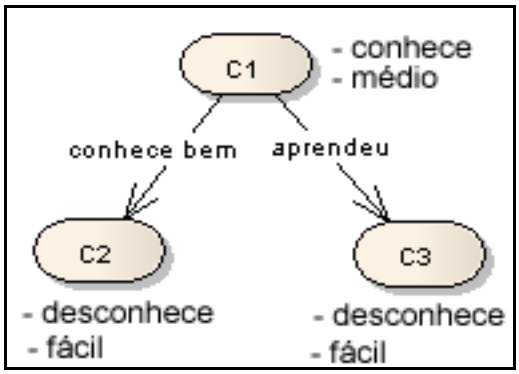


Figura 43 - Modelo do aluno que acertou 35%

No Quadro 30 são apresentadas as atividades realizadas pelo aluno que acerta 50%, onde para cada atividade de nível difícil o mesmo tem um percentual de acertos aproximado de 50%.

Exercício	Acertos	Tempo	Complexidade	Desempenho	Próximo nível
C1 F1	alto	normal	fácil	conhece	médio
C1 M1	alto	normal	médio	conhece bem	difícil
C1 D1	razoável	devagar	difícil	conhece bem	médio
C1 M2	alto	normal	médio	conhece bem	difícil
C1 D2	razoável	devagar	difícil	conhece bem	médio
C2 F1	alto	normal	fácil	conhece	médio
C2 M1	alto	normal	médio	conhece bem	difícil
C2 D1	razoável	devagar	difícil	conhece bem	médio
C2 M2	alto	normal	médio	conhece bem	difícil
C2 D2	razoável	devagar	difícil	conhece bem	médio

Quadro 30 - Atividades realizadas pelo aluno que acertou 50%

Este aluno atingiu a condição para o conceito C2 após a realização da primeira atividade de nível médio realizada no conceito C1, mas mesmo após realizar as demais atividades em C1 não conseguiu obter o desempenho para iniciar em C3. Na Figura 44 apresenta o modelo do aluno após a realização das atividades no conceito C1 e C2.

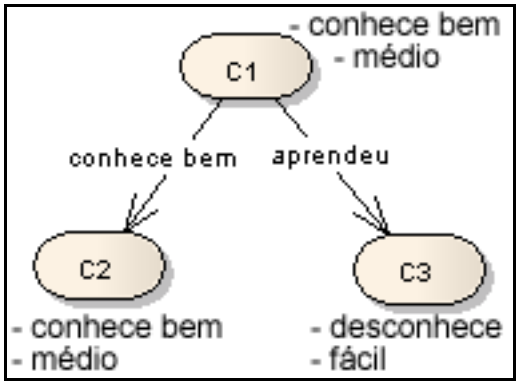


Figura 44 - Modelo do aluno que acertou 50%

No Quadro 31 são apresentadas as atividades realizadas pelo aluno que acerta 100%, ou seja, o aluno acerta totalmente as atividade que realiza.

Exercício	Acertos	Tempo	Complexidade	Desempenho	Próximo nível
C1 F1	alto	normal	fácil	conhece	médio
C1 M1	alto	normal	médio	conhece bem	difícil
C1 D1	alto	normal	difícil	aprendeu bem	difícil
C2 F1	alto	normal	fácil	conhece	médio
C2 M1	alto	normal	médio	conhece bem	difícil
C2 D1	alto	normal	difícil	aprendeu bem	difícil
C3 F1	alto	normal	fácil	conhece	médio
C3 M1	alto	normal	médio	conhece bem	difícil
C3 D2	alto	normal	difícil	aprendeu bem	difícil

Quadro 31 - Atividades realizadas pelo aluno que acertou 100%

Neste caso o aluno atingirá a condição do conceito C2 após realizar a atividade de nível médio em C1, e C3 após a realização da atividade de nível difícil em C1. A Figura 45 apresenta o modelo do aluno após realização das atividades.

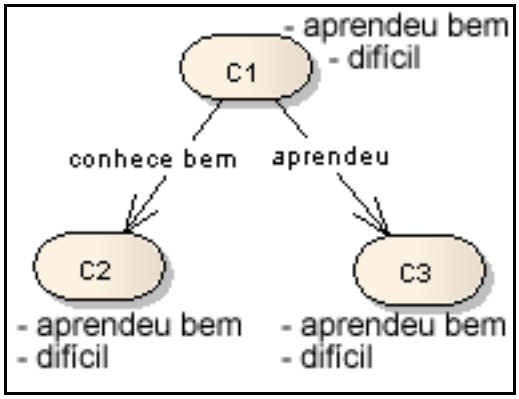


Figura 45 - Modelo do aluno que acertou 100%

3.4 RESULTADOS E DISCUSSÃO

Este trabalho atingiu o objetivo de desenvolver um ambiente com um comportamento inteligente utilizando a lógica *fuzzy* e disponibilizar um objeto de aprendizagem para o aluno desenvolver suas atividades. Apesar de não terem sido realizados testes com alunos, o ambiente se mostrou de simples utilização, mas é indicado para alunos que estão realizando a disciplina de Banco de Dados II, como uma ferramenta de apoio na realização de atividades.

A correção da atividade pelo objeto de aprendizagem é feita através de comparação de

igualdade o que limita uma avaliação, pois é na avaliação parcial que se fazem as conclusões dos conceitos não conhecidos.

O STI desenvolvido superou os objetivos que se tinha inicialmente, ao possibilitar a flexibilidade da construção do modelo de domínio, assim como o uso de variável lingüística para restringir o acesso a conceitos que o aluno ainda não possua conhecimento suficiente. E a flexibilidade ao poder adicionar e organizar conteúdo e atividades no STI com o uso de pacotes SCORM. E de o STI possibilita a alteração dos parâmetros do sistema *fuzzy* pelo especialista, tanto como a alteração das funções de pertinência quando a adição e remoção de regras.

Em relação ao trabalho de conclusão de curso de Forest (2004), o STI aqui desenvolvido teve uma grande diferença, pois apresenta ao aluno um resultado sobre o seu desempenho na realização da atividade, e com base no seu desempenho é que será selecionada a próxima atividade a ser desenvolvida. As atividades são organizadas por conceito, o aluno não verá conceitos onde o mesmo não tenha conhecimento necessário para realizá-lo.

Sobre a ferramenta LabSQL, o trabalho aqui desenvolvido realiza a inferência a cada interação do aluno e com isso o aluno fica ciente do seu desempenho a cada atividade realizada. O trabalho aqui desenvolvido, além de disponibilizar o desempenho do aluno, ainda define qual será o próximo nível de atividade que será realizada pelo mesmo e este processo de inferência ocorre uma única vez, ou seja, há 2 (duas) variáveis de saída para as mesmas variáveis de entrada.

4 CONCLUSÕES

Através do desenvolvimento deste trabalho foi possível adquirir conhecimento sobre lógica fuzzy, conhecimento este não visto em momento algum durante a graduação, além de relembrar conceitos de implementação de BD.

A utilização da lógica fuzzy mostrou-se eficaz para a modelagem do STI proposto, pois além de sua simplicidade e reduzido custo computacional, fornece informações importantes para identificar o conhecimento adquirido pelo aluno, bem como o identifica qual será a melhor atividade a apresentar para o mesmo.

Não foi atendido o requisito de disponibilizar uma interface ao professor cadastrar as atividades. A justificativa para este fato foi a adição do requisito de execução de pacotes SCORM, o que da um maior valor ao trabalho por permitir uma maior flexibilidade na adição de conteúdo a ser disponibilizado ao aluno. Porém, a disponibilização desta interface para montagem do pacote SCORM não estava prevista no cronograma original. Vale ressaltar que a montagem manual deste pacote não é uma função trivial, pois há a necessidade de se codificar o arquivo XML da atividade e organizar a estrutura do pacote.

O requisito permitir ao professor e aluno gerar atividade não foi mais atendida, pois inicialmente tinha se a idéia de que o aluno pudesse solicitar uma atividade ou mesmo o professor determinar uma atividade a um aluno, e não sendo mais atendido pelo motivo que a disponibilização das atividades é realizada pelo STI.

Este trabalho serviu como validação do componente CELINE, pois o desenvolvimento deste componente está em andamento como projeto de dissertação do orientador.

Este trabalho contribuiu na especificação da comunicação entre um ambiente inteligente de aprendizagem e pacotes SCORM. A persistência dos dados produzidos pelo CELINE apresentou problemas e por essa razão não existe continuidade de atividades suspensas.

Por fim, este trabalho teve como seu objetivo maior fornecer subsídios automatizados para o professor poder acompanhar o desempenho do aluno no desenvolvimento das atividades, assim como uma ferramenta de estudo sobre o assunto na disciplina.

4.1 EXTENSÕES

Segue abaixo sugestões para trabalhos futuros:

a) permitir ao professor importar os pacotes SCORM;
b) desenvolver uma ferramenta de criação dos pacotes do OA.
c) corrigir as atividades do gerenciador de armazenamento e arquivo com maior flexibilidade, e que aponte com maior exatidão os erros dos alunos;
d) desenvolver uma interface para configuração dos parâmetros do sistema *fuzzy*;
e) implementar uma interface para cadastro de conceitos;
f) o STI mostrar quais conceitos o aluno deverá realizar e qual o desempenho que deve ser atingido;
g) automatizar testes para que se possa explorar melhor a modelagem do sistema *fuzzy*;
h) desenvolver outros objetos de aprendizagem.

REFERÊNCIAS BIBLIOGRÁFICAS

COSTA, K. et al. Acompanhamento do estudante em ambientes de aprendizagem utilizando Lógica Fuzzy. In: Congresso da Sociedade Brasileira de Computação, 26., 2005, Campo Grande. **Anais eletrônicos...** Pará: UFPA, 2005. p. 41-49 Disponível em: <http://www.labead.ufpa.br/gped/publicacaoes >. Acesso em: 18 set. 2007.

COSTA, M. **Sistema tutor inteligente**. Rio Janeiro, [2002?]. Disponível em: <http://www.nce.ufrj.br/ginape/publicacoes/trabalhos/MacarioMaterial/Sti.htm >. Acesso em: 20 set. 2007.

ELMASRI, R.; NAVATHE, S. B. **Sistema de banco de dados.** 4. ed. Tradução Marília Guimarães Pinheiro. São Paulo: Pearson Education do Brasil, 2005.

FOREST, K. W. **Protótipo de um sistema tutor de orientação objetos.** 2004. 68 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

FRIEDMAN-HILL E. **Jess, the rule engine for the Java™ Platform**. Estados Unidos, [2008]. Disponível em: < http://herzberg.ca.sandia.gov/ >. Acesso em: 20 abr. 2008.

FUJII, N. N.; SILVEIRA, I. F. Individualizando o ensino de estatística através do uso de objetos de aprendizagem adaptativos. In: Simpósio Brasileiro de Informática na Educação, 15., 2006, Brasília. **Anais eletrônicos...** Brasília: UNICSUL, 2006. Não pág. Disponível em: < http://www.sbc.org.br/bibliotecadigital/download.php?paper=758>. Acesso em: 18 set. 2007.

GARCIA-MOLINA, H.; ULLMAN, J. D.; WIDOM, J. **Implementação de sistema de banco de dados.** Tradução Vandenberg D. de Souza. Rio Janeiro: Campus, 2001.

LIMA, D. R.; ROSATELLI, M. C**.** Um sistema tutor inteligente para um ambiente virtual de ensino aprendizagem. In: Congresso da Sociedade Brasileira de Computação, 13., 2003, Campinas. **Anais do XXIII Congresso da Sociedade Brasileira de Computação**. Campinas: SBC, 2003. p. 129-129.

LOPES, W. A; JAFELICE, R. S. W; BARROS, L. C**.** Modelagem fuzzy de diagnóstico médico e monitoramento do tratamento da pneumonia. In: Congresso Nacional de Matemática Aplicada e Computacional, 28., 2005, São Paulo. **Anais do XXVIII CNMAC - Congresso Nacional de Matemática Aplicada e Computacional**. São Paulo: SBMAC, 2005. p. 77-96 Disponível em: <http:// www.ime.unicamp.br/~biomat/bio15final.pdf >. Acesso em: 10 maio. 2008.

MOREIRA, T. D. R. G. Sistemas tutores inteligentes. In: OLIVEIRA JR., AGUIAR H (Coord.). **Inteligência Computacional**: Aplicada à Administração, Economia e Engenharia em Matlab. São Paulo: Thomson, 2007. p. 265-280.

OLIVEIRA JR. et al. Lógica *fuzzy*. In: OLIVEIRA JR., AGUIAR H (Coord.). **Inteligência Computacional**: A Aplicada à Administração, Economia e Engenharia em Matlab. São Paulo: Thomson, 2007. p. 1-66.

ORCHARD, R. **NRC *FuzzyJ toolkit for the Java™ platform* user's guide**. Canadá, [2006]. Disponível em: < http://www.iit.nrc.ca/IR_public/fuzzy/fuzzyJToolkit.html >. Acesso em: 20 fev. 2008.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de banco de dados.** 3. ed. Tradução Marília Guimarães Pinheiro, Cláudio César Canhette. São Paulo: Pearson Education, 1999.

SILVA, H. A. N. et al. Um sistema baseado na lógica difusa para decidir os conceitos finais dos estudantes críticos. In: Congresso da Sociedade Brasileira de Computação, 38., 2008, Belém do Pará. **Anais eletrônicos...** São Paulo: SBC, 2008. Não pág. Disponível em: <http:// http://www.prodepa.gov.br/sbc2008/anais/pdf/arq0193.pdf >. Acesso em: 14 julho. 2008.

TANSCHEIT, R. Sistemas *fuzzy*. In: OLIVEIRA JR., AGUIAR H (Coord.). **Inteligência Computacional**: Aplicada à Administração, Economia e Engenharia em Matlab. São Paulo: Thomson, 2007. p. 229-264.

VAHLDICK, A.; RAABE, A. L. A. **CELINE – Componente para player SCORM em aplicações Java**. Blumenau, [2008?]. Disponível em: < http://www.inf.furb.br/~adilsonv/tcc/CELINE.pdf>. Acesso em: 20 maio 2008.

ANEXO A – Conteúdo do arquivo gerado pelo Matlab.

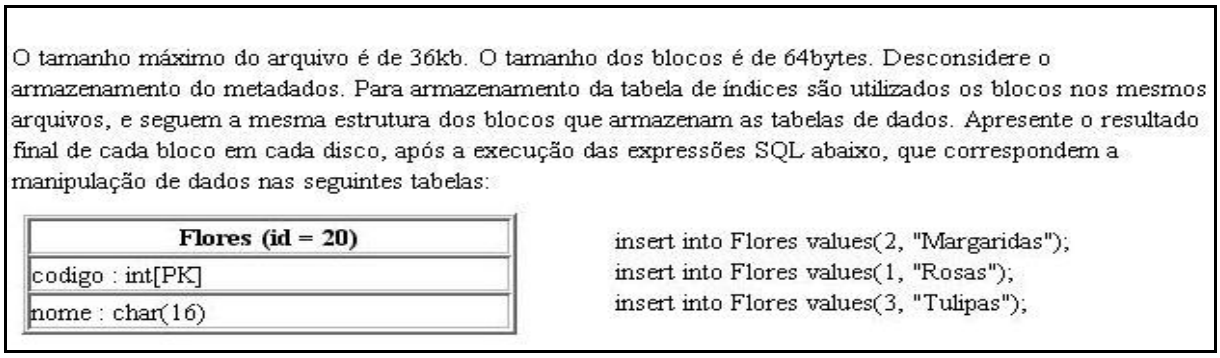
No Quadro 32 é apresentado o conteúdo do arquivo gerado pelo Matlab.

```
[System]
Name='SAGAA'
Type='mamdani'
Version=2.0
NumInputs=3
NumOutputs=2
NumRules=19
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='Complexidade'
Range=[0 10]
NumMFs=3
MF1='facil':'trapmf',[0 0 2 3]
MF2='medio':'trapmf',[2 3 6 7]
MF3='dificil':'trapmf',[6 7 10 10]

[Input2]
Name='Acertos'
Range=[0 100]
NumMFs=3
MF1='baixo':'trapmf',[0 0 20 25]
MF2='razoavel':'trapmf',[20 25 65 70]
MF3='alto':'trapmf',[65 70 100 100]

[Input3]
Name='Tempo'
Range=[0 100]
NumMFs=3
MF1='rapido':'trapmf',[0 0 5 8]
MF2='normal':'trimf',[5 10 20]
MF3='devagar':'trapmf',[12 20 100 100]

[Output1]
Name='Desempenho'
Range=[0 100]
NumMFs=6
MF1='desconhece':'trapmf',[0 0 16.5 18]
MF2='conhece':'trapmf',[30 32.1 100 100]
MF3='aprendeu_bem':'trapmf',[76 78.1 100 100]
MF4='conhece_pouco':'trapmf',[14 16.6 100 100]
MF5='aprendeu':'trapmf',[56 58.1 100 100]
MF6='conhece_bem':'trapmf',[40 42.1 100 100]

[Output2]
Name='Proximo_Nivel'
Range=[0 10]
NumMFs=3
MF1='facil':'trapmf',[0 0 2 3]
MF2='medio':'trapmf',[2 3 6 7]
MF3='dificil':'trapmf',[6 7 10 10]

[Rules]
1 1 0, 1 1 (1) : 1
1 2 3, 1 1 (1) : 1
1 2 2, 1 1 (1) : 1
1 2 1, 4 1 (1) : 1
1 3 3, 4 1 (1) : 1
1 3 2, 2 2 (1) : 1
1 3 1, 2 2 (1) : 1
2 1 0, 4 1 (1) : 1
2 2 3, 4 1 (1) : 1
2 2 2, 2 2 (1) : 1
2 2 1, 2 2 (1) : 1
2 3 3, 2 2 (1) : 1
2 3 2, 6 3 (1) : 1
2 3 1, 6 3 (1) : 1
3 1 0, 2 1 (1) : 1
3 2 3, 6 2 (1) : 1
3 2 2, 5 3 (1) : 1
3 2 1, 5 3 (1) : 1
3 3 0, 3 3 (1) : 1
```

Quadro 32 – Arquivo gerado pelo Matlab

ANEXO B – Enunciado das atividades.

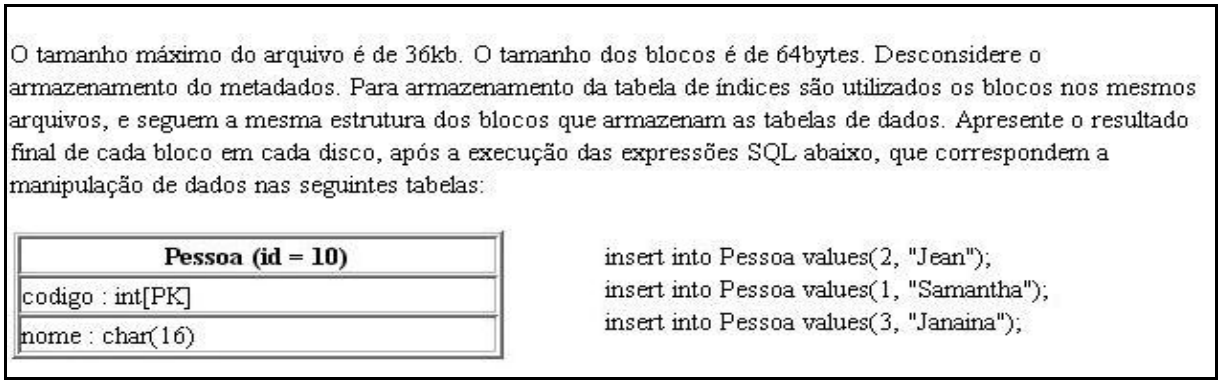
A Figura 46 e Figura 47 apresentam as atividade de nível fácil do conceito C1.


O tamanho máximo do arquivo é de 36kb. O tamanho dos blocos é de 64bytes. Desconsidere o armazenamento do metadados. Para armazenamento da tabela de índices são utilizados os blocos nos mesmos arquivos, e seguem a mesma estrutura dos blocos que armazenam as tabelas de dados. Apresente o resultado final de cada bloco em cada disco, após a execução das expressões SQL abaixo, que correspondem a manipulação de dados nas seguintes tabelas:

Flores (id = 20)
codigo : int[PK]
nome : char(16)

```
insert into Flores values(2, "Margaridas");
insert into Flores values(1, "Rosas");
insert into Flores values(3, "Tulipas");
```

Figura 46 - Enunciado da atividade C1 F1

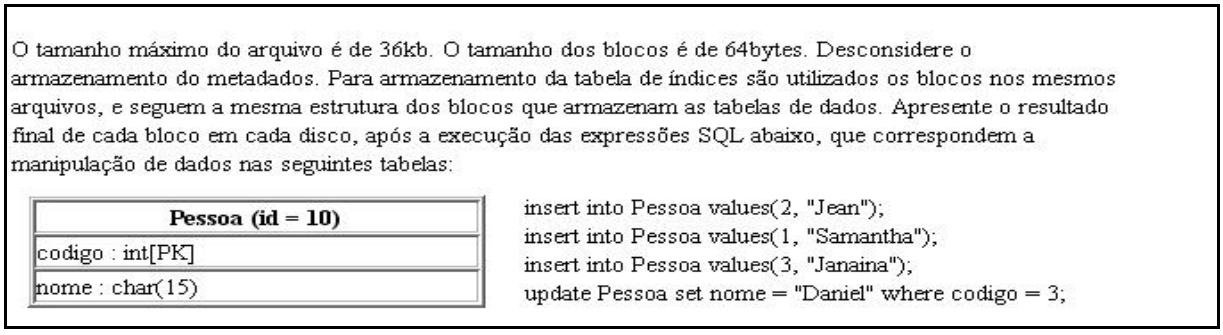

O tamanho máximo do arquivo é de 36kb. O tamanho dos blocos é de 64bytes. Desconsidere o armazenamento do metadados. Para armazenamento da tabela de índices são utilizados os blocos nos mesmos arquivos, e seguem a mesma estrutura dos blocos que armazenam as tabelas de dados. Apresente o resultado final de cada bloco em cada disco, após a execução das expressões SQL abaixo, que correspondem a manipulação de dados nas seguintes tabelas:

Pessoa (id = 10)
codigo : int[PK]
nome : char(16)

```
insert into Pessoa values(2, "Jean");
insert into Pessoa values(1, "Samantha");
insert into Pessoa values(3, "Janaina");
```

Figura 47 - Enunciado da atividade C1 F2

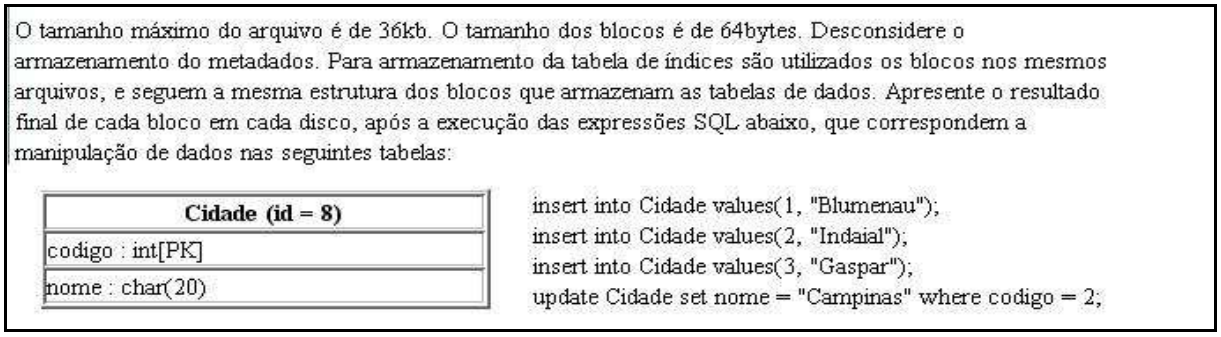
A Figura 48 e Figura 49 apresentam as atividade de nível médio do conceito C1.


O tamanho máximo do arquivo é de 36kb. O tamanho dos blocos é de 64bytes. Desconsidere o armazenamento do metadados. Para armazenamento da tabela de índices são utilizados os blocos nos mesmos arquivos, e seguem a mesma estrutura dos blocos que armazenam as tabelas de dados. Apresente o resultado final de cada bloco em cada disco, após a execução das expressões SQL abaixo, que correspondem a manipulação de dados nas seguintes tabelas:

Pessoa (id = 10)
codigo : int[PK]
nome : char(15)

```
insert into Pessoa values(2, "Jean");
insert into Pessoa values(1, "Samantha");
insert into Pessoa values(3, "Janaina");
update Pessoa set nome = "Daniel" where codigo = 3;
```

Figura 48 - Enunciado da atividade C1 M1


O tamanho máximo do arquivo é de 36kb. O tamanho dos blocos é de 64bytes. Desconsidere o armazenamento do metadados. Para armazenamento da tabela de índices são utilizados os blocos nos mesmos arquivos, e seguem a mesma estrutura dos blocos que armazenam as tabelas de dados. Apresente o resultado final de cada bloco em cada disco, após a execução das expressões SQL abaixo, que correspondem a manipulação de dados nas seguintes tabelas:

Cidade (id = 8)
codigo : int[PK]
nome : char(20)

```
insert into Cidade values(1, "Blumenau");
insert into Cidade values(2, "Indaial");
insert into Cidade values(3, "Gaspar");
update Cidade set nome = "Campinas" where codigo = 2;
```

Figura 49 - Enunciado da atividade C1 M2

A Figura 50 e Figura 51 apresentam as atividade de nível difícil do conceito C1.

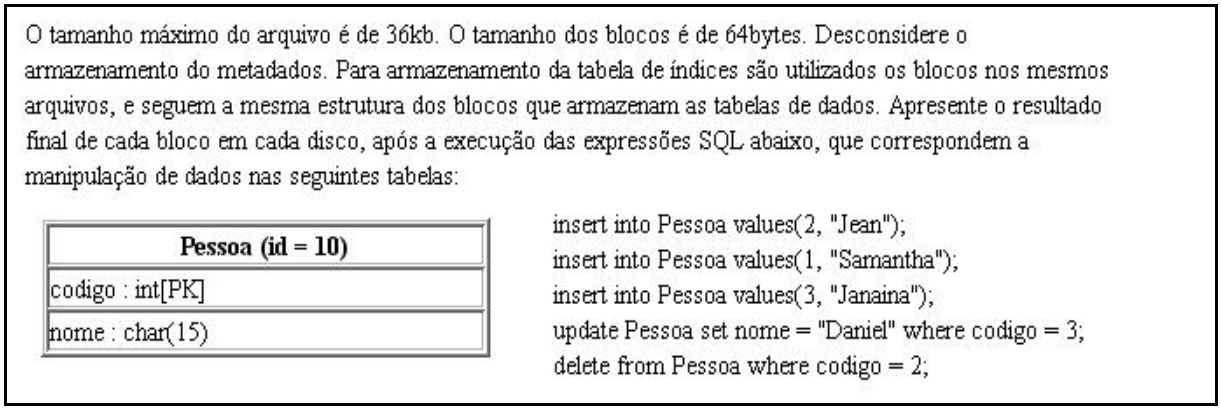


Figura 50 - Enunciado da atividade C1 D1

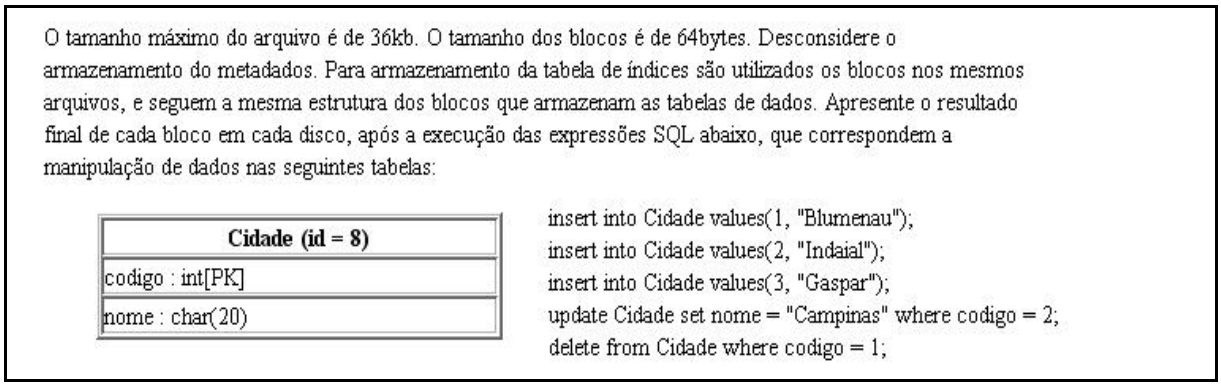


Figura 51 - Enunciado da atividade C1 D2

A Figura 52 e Figura 53 apresentam as atividade de nível fácil do conceito C2.

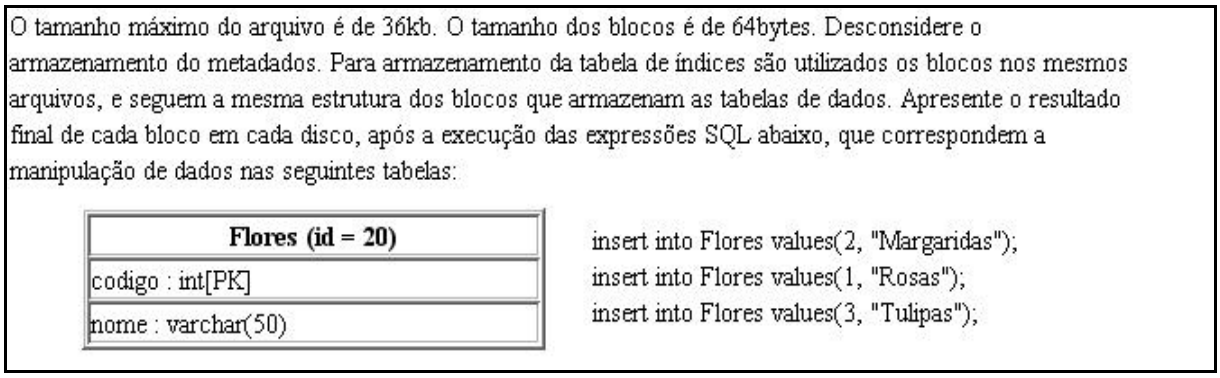


Figura 52 - Enunciado da atividade C2 F1

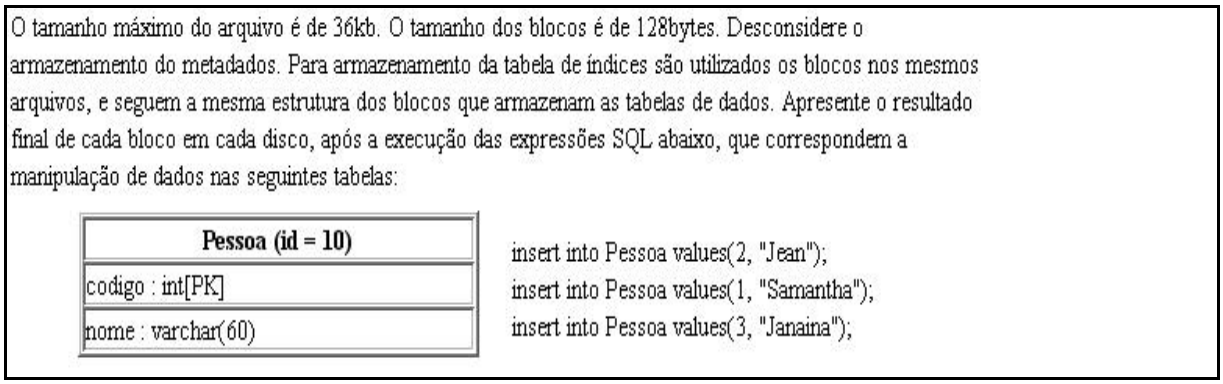


Figura 53 - Enunciado da atividade C2 F2

A Figura 54 e Figura 55 apresentam as atividade de nível médio do conceito C2.

O tamanho máximo do arquivo é de 36kb. O tamanho dos blocos é de 64bytes. Desconsidere o armazenamento do metadados. Para armazenamento da tabela de índices são utilizados os blocos nos mesmos arquivos, e seguem a mesma estrutura dos blocos que armazenam as tabelas de dados. Apresente o resultado final de cada bloco em cada disco, após a execução das expressões SQL abaixo, que correspondem a manipulação de dados nas seguintes tabelas:

Pessoa (id = 10)
codigo : int[PK]
nome : varchar(50)

```
insert into Pessoa values(2, "Jean");
insert into Pessoa values(1, "Samantha");
insert into Pessoa values(3, "Janaina");
delete from Pessoa where codigo = 2;
```

Figura 54 - Enunciado da atividade C2 M1

O tamanho máximo do arquivo é de 36kb. O tamanho dos blocos é de 128bytes. Desconsidere o armazenamento do metadados. Para armazenamento da tabela de índices são utilizados os blocos nos mesmos arquivos, e seguem a mesma estrutura dos blocos que armazenam as tabelas de dados. Apresente o resultado final de cada bloco em cada disco, após a execução das expressões SQL abaixo, que correspondem a manipulação de dados nas seguintes tabelas:

Cidade (id = 8)
codigo : int[PK]
nome : varchar(20)

```
insert into Cidade values(1, "Blumenau");
insert into Cidade values(2, "Indaial");
insert into Cidade values(3, "Gaspar");
delete from Cidade where codigo = 1;
```

Figura 55 - Enunciado da atividade C2 M2

A Figura 56 e Figura 57 apresentam as atividade de nível difícil do conceito C2.

O tamanho máximo do arquivo é de 36kb. O tamanho dos blocos é de 256bytes. Desconsidere o armazenamento do metadados. Para armazenamento da tabela de índices são utilizados os blocos nos mesmos arquivos, e seguem a mesma estrutura dos blocos que armazenam as tabelas de dados. Apresente o resultado final de cada bloco em cada disco, após a execução das expressões SQL abaixo, que correspondem a manipulação de dados nas seguintes tabelas:

Pessoa (id = 10)
codigo : int[PK]
nome : varchar(80)

```
insert into Pessoa values(2, "Jean");
insert into Pessoa values(1, "Samantha");
insert into Pessoa values(3, "Janaina");
update Pessoa set nome = "Daniel" where codigo = 1;
delete from Pessoa where codigo = 2;
```

Figura 56 - Enunciado da atividade C2 D1

O tamanho máximo do arquivo é de 36kb. O tamanho dos blocos é de 64bytes. Desconsidere o armazenamento do metadados. Para armazenamento da tabela de índices são utilizados os blocos nos mesmos arquivos, e seguem a mesma estrutura dos blocos que armazenam as tabelas de dados. Apresente o resultado final de cada bloco em cada disco, após a execução das expressões SQL abaixo, que correspondem a manipulação de dados nas seguintes tabelas:

Cidade (id = 8)
codigo : int[PK]
nome : varchar(30)

```
insert into Cidade values(1, "Blumenau");
insert into Cidade values(2, "Indaial");
insert into Cidade values(3, "Gaspar");
update Cidade set nome = "Campinas" where codigo = 2;
delete from Cidade where codigo = 1;
```

Figura 57 - Enunciado da atividade C2 D2

A Figura 58 e Figura 59 apresentam as atividade de nível fácil do conceito C3.

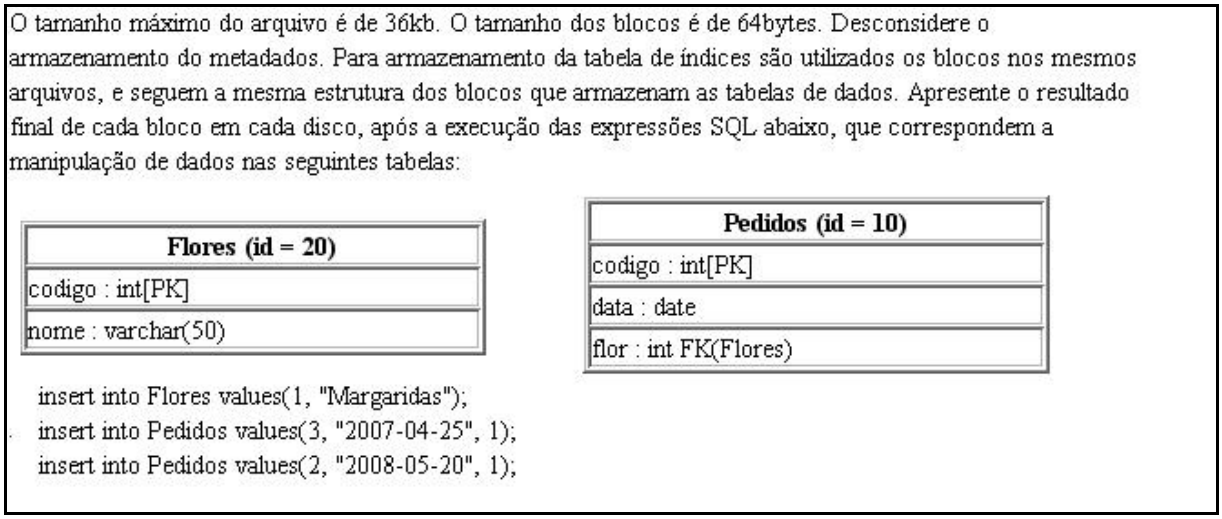


Figura 58 - Enunciado da atividade C3 F1

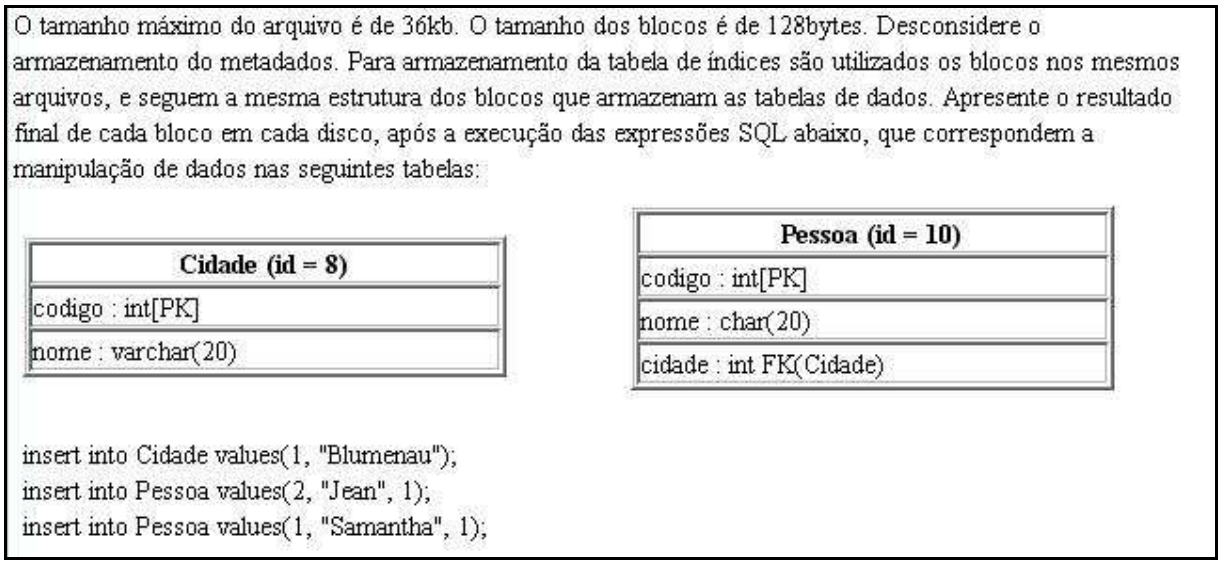


Figura 59 - Enunciado da atividade C3 F2

A Figura 60 e Figura 61 apresentam as atividade de nível médio do conceito C3.

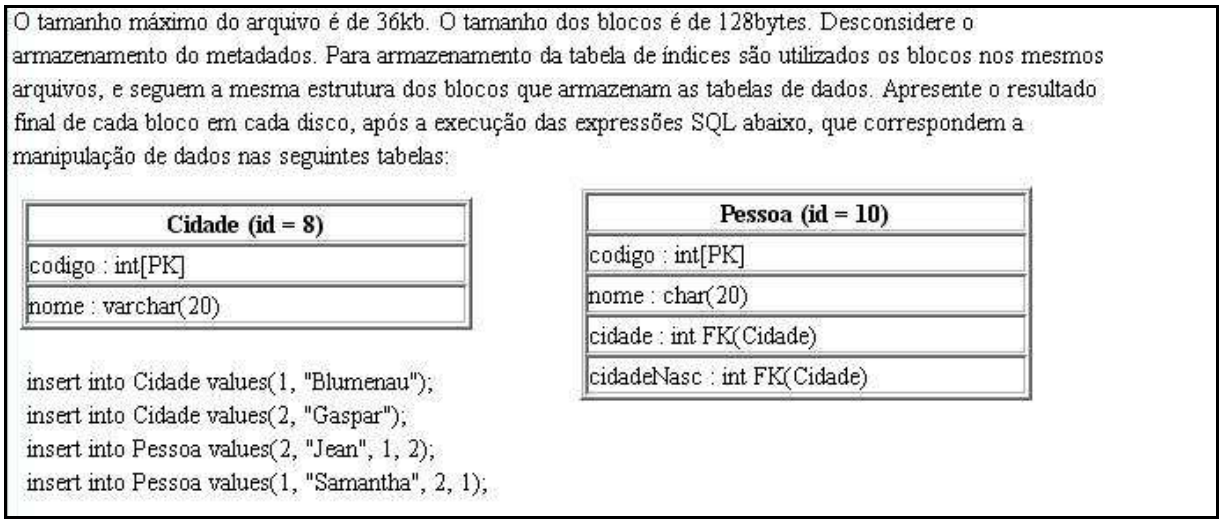


Figura 60 - Enunciado da atividade C3 M1

O tamanho máximo do arquivo é de 36kb. O tamanho dos blocos é de 64bytes. Desconsidere o armazenamento do metadados. Para armazenamento da tabela de índices são utilizados os blocos nos mesmos arquivos, e seguem a mesma estrutura dos blocos que armazenam as tabelas de dados. Apresente o resultado final de cada bloco em cada disco, após a execução das expressões SQL abaixo, que correspondem a manipulação de dados nas seguintes tabelas:

Flores (id = 6)
codigo : int[PK]
nome : varchar(50)

Pessoa (id = 8)
codigo : int[PK]
nome : char(20)

Pedidos (id = 10)
codigo : int[PK]
data : date
flor : int FK(Flores)
pessoa : int FK(Pessoa)

```
insert into Pessoa values(1, "Jean");
insert into Flores values(1, "Margaridas");
insert into Pedidos values(3, "2007-04-25", 1, 1);
insert into Pedidos values(2, "2008-05-20", 1, 1);
```

Figura 61 - Enunciado da atividade C3 M2

A Figura 62 e Figura 63 apresentam as atividade de nível difícil do conceito C3.

O tamanho máximo do arquivo é de 36kb. O tamanho dos blocos é de 128bytes. Desconsidere o armazenamento do metadados. Para armazenamento da tabela de índices são utilizados os blocos nos mesmos arquivos, e seguem a mesma estrutura dos blocos que armazenam as tabelas de dados. Apresente o resultado final de cada bloco em cada disco, após a execução das expressões SQL abaixo, que correspondem a manipulação de dados nas seguintes tabelas:

Cidade (id = 5)
codigo : int[PK]
nome : varchar(20)

Carro (id = 8)
codigo : int[PK]
nome : char(20)

Pessoa (id = 10)
codigo : int[PK]
nome : char(20)
cidade : int FK(Cidade)
cidadeNasc : int FK(Cidade)
carro : int FK(Carro)

```
insert into Cidade values(1, "Blumenau");
insert into Cidade values(2, "Gaspar");
insert into Carro values(1, "Fiesta");
insert into Carro values(2, "Corsa");
insert into Pessoa values(2, "Jean", 1, 2, 2);
insert into Pessoa values(1, "Samantha", 2, 1, 1);
```

Figura 62 - Enunciado da atividade C3 D1

O tamanho máximo do arquivo é de 36kb. O tamanho dos blocos é de 128bytes. Desconsidere o armazenamento do metadados. Para armazenamento da tabela de índices são utilizados os blocos nos mesmos arquivos, e seguem a mesma estrutura dos blocos que armazenam as tabelas de dados. Apresente o resultado final de cada bloco em cada disco, após a execução das expressões SQL abaixo, que correspondem a manipulação de dados nas seguintes tabelas:

Disciplina (id = 5)
codigo : int[PK]
nome : char(10)

Pessoa (id = 10)
codigo : int[PK]
nome : char(20)
matematica : int FK(Disciplina)
fisica : int FK(Disciplina)
portugues : int FK(Disciplina)

```
insert into Disciplina values(1, "Mat. 001");
insert into Disciplina values(2, "Mat. 002");
insert into Disciplina values(3, "Fis. 009");
insert into Disciplina values(4, "Por. 006");
insert into Pessoa values(2, "Jean", 1, 3, 4);
insert into Pessoa values(1, "Samantha", 2, 3, 4);
```

Figura 63 - Enunciado da atividade C3 D2