UNIVERSIDADE PRESBITERIANA MACKENZIE

GUILHERME BARBOSA RATOLA

Ferramenta para Acompanhamento do Processo de Ensino-Aprendizagem de Programação de Computadores Utilizando o Método de Aprendizagem por Competências

São Paulo
2010

GUILHERME BARBOSA RATOLA

Ferramenta para Acompanhamento do Processo de Ensino-Aprendizagem de Programação de Computadores Utilizando o Método de Aprendizagem por Competências

Dissertação apresentada à Universidade Presbiteriana Mackenzie como requisito parcial para a obtenção do título de Mestre em Engenharia Elétrica na área de Engenharia da Computação.

Orientador: Prof. Dr. Nizam Omar

São Paulo
2010

R236f Ratola, Guilherme Barbosa.
Ferramenta para acompanhamento do processo de ensino-aprendizagem de programação de computadores utilizando o método de aprendizagem por competências / Guilherme Barbosa Ratola. – 2010.
83 p. : il. ; 30 cm.
Dissertação (Mestrado em Engenharia Elétrica) – Escola de Engenharia, Universidade Presbiteriana Mackenzie, São Paulo, 2010.
Orientação: Nizam Omar
Bibliografia: p. 65-68
1.Ambiente de Aprendizagem. 2. Programação de computadores. 3. Método de aprendizagem por competências. 4. Linguagem de programação Java. II. Título.
CDD 005.13

GUILHERME BARBOSA RATOLA

Ferramenta para Acompanhamento do Processo de Ensino-Aprendizagem de Programação de Computadores Utilizando o Método de Aprendizagem por Competências

Dissertação apresentada à Universidade Presbiteriana Mackenzie como requisito parcial para a obtenção do título de Mestre em Engenharia Elétrica na área de Engenharia da Computação.

Aprovado em 02/02/2010

BANCA EXAMINADORA

_____

Prof. Dr. Nizam Omar
Universidade Presbiteriana Mackenzie

_____

Prof. Dr. Leandro Nunes de Castro
Universidade Presbiteriana Mackenzie

_____

Prof. Dr. Edson Pinheiro Pimentel
Universidade Federal do ABC

À minha mãe.

Agradecimentos

1. Ao Professor Dr. Nizam Omar pela orientação e incentivo.
2. Aos meus avôs, mais do que meus companheiros.
3. A todos os Professores do Programa de Mestrado do Mackenzie, somado do Professor Dr. Edson Pimentel, pela colaboração e apoio.
4. A CAPES pelo suporte ao desenvolvimento desta pesquisa.
5. Ao MACKPESQUISA pelo suporte ao desenvolvimento desta pesquisa.

Resumo

O aprendizado de programação de computadores é fundamental na formação de um profissional de computação. No entanto, esta atividade é de aprendizagem difícil, contínua e permanente, uma vez que a cada dia novas linguagens e ambientes surgem no mercado de trabalho. Nos cursos da área de computação, muitos estudantes apresentam dificuldades para aprender a programar, pois a atividade requer alto nível de competência e o aprendizado e domínio de conceitos complexos. O desenvolvimento de ambientes computacionais que apóiem a aprendizagem de programação é de grande valia nesse cenário, que envolve estudantes, professores e profissionais. Muitas pesquisas e trabalhos têm sido desenvolvidos nessa área, na medida em que fornecem contribuições importantes na construção de ambientes computacionais. Este trabalho apresenta uma ferramenta de suporte ao processo de ensino-aprendizagem de programação a alunos de graduação, priorizando a interdisciplinaridade, com uma abordagem baseada no método de aprendizagem por competências. Apresenta-se aqui também o mecanismo de Pirâmide Conceitual como uma alternativa ao planejamento de aprendizagem para o aluno. Como suporte ao ensino de programação, este trabalho utiliza a linguagem de programação Java, motivado pela alta demanda de utilização nos cursos superiores. Os estudantes são classificados por seus conhecimentos prévios em cada um dos conceitos e é esperado que a partir dessas informações e com a medida do *Conceptual Gap* com relação a Pirâmide Conceitual de determinado conceito seja possível desenvolver um plano de aprendizagem para esse aluno. Espera-se que a utilização desta ferramenta, auxiliando o processo de aprendizagem, contribua no sentido de tornar a representação de conceitos, exercícios e competências relacionados às disciplinas de um curso de Computação adequada.

Palavras-chave: *Ambiente de Aprendizagem, Programação de Computadores, Método de Aprendizagem por Competências, Linguagem de Programação Java.*

Abstract

The learning of computer programming is fundamental in the formation of a computing professional. However, its learning is difficult and longlife, considering that every day new languages and environments appear in the labor market. In the Computer Science courses, many students have difficulties programming, the activity requires the learning of complex concepts. The development of computational environments to support the learning of programming is of great value in this scenario, which involves students, teachers and professionals. Many researches and studies have been developed and they provide important contributions in the construction of learning computing environments. This dissertation proposes the development of a tool to support the process of teaching and learning of programming to students, prioritizing interdisciplinary, with an approach based on the Method of Learning by Competence. It presents also a mechanism of Concept Pyramid as an alternative to the planning of learning for a student. To support teaching of computer programming, the tool, presented in this work, uses the Java programming language, motivated by the high demand for use in undergraduate courses. The students are classified by their prior knowledge of each concept and it is expected that from this information and the measure of the student's Conceptual Gap regarding Conceptual Pyramid of a concept is possible to develop a learning plan for each student. It is expected that the use of this tool, helping the process of teaching and learning in the classroom, contribute to make the representation of concepts, exercises and skills related to each discipline from an appropriate course.

Key-words: *Learning Environment, Computer Programming, Method of Learning for Competence, Java Programming Language.*

Lista de Figuras

Lista de Tabelas

SUMÁRIO

1 INTRODUÇÃO

Este capítulo tem o objetivo de apresentar o problema, discorrer a respeito de como se encontra o cenário atual deste problema, apresentar o objetivo e a relevância da proposta desta pesquisa.

1.1 O PROBLEMA

A aprendizagem de programação de computadores é essencial para as carreiras ligadas à Informática. No entanto, é amplamente conhecido que aprender a programar, mesmo em um nível básico, é uma atividade difícil (PARVEZ; BLANK, 2007).

Nos cursos da área de computação, muitos estudantes apresentam dificuldades para programar, pois essa atividade requer o aprendizado de conceitos novos e uma forma de raciocínio diferente, o que acaba por tornar alto o índice de reprovações e desistências.

Segundo Castro et al. (2002) dois problemas se destacam nesse cenário: Por um lado, pouca ênfase é dada às habilidades para entender e solucionar problemas nesses cursos. Por outro, o estudante ainda tem de transformar cada solução num conjunto de comandos a serem executados pelo computador. Essa transformação é possivelmente a principal dificuldade de todo iniciante em programação. A necessidade de descrever comandos com rigorosa precisão torna-se assim uma característica muito artificial e até mesmo desgastante para o estudante.

Aprender programação não é somente aprender uma linguagem de programação. É fundamental que conceitos iniciais, como por exemplo, declarações de variáveis e tipos de variáveis, operadores e parâmetros, controles de repetição e condição, tenham uma atenção especial. Segundo Xavier, Garcia e Silva (2004) conceitos de programação necessitam de um tempo de maturação para seu total domínio e uso.

Já Neto (2006) parte do pressuposto que um dos principais aspectos na aprendizagem de programação é a organização das habilidades para a resolução de problemas, elemento comum à construção de conhecimento em qualquer outro domínio, assim surgindo a necessidade de investigar como tais habilidades são construídas, o que envolve identificar e entender as dificuldades encontradas pelos estudantes de programação.

Dessa forma, para realizarem tarefas, tanto profissionais como acadêmicas, e no âmbito acadêmico serem aprovados em disciplinas nos cursos de computação, é necessário que os estudantes dominem suficientemente um conjunto de conceitos, esboçando sinais de competência nas atividades que estão desempenhando.

1.2 OBJETIVO

O principal objetivo deste trabalho é o desenvolvimento de uma ferramenta computacional que dê suporte ao processo de ensino-aprendizagem de programação, incorporando em sua metodologia, técnicas exploradas por esta pesquisa, que serviram como alicerces para o desenvolvimento deste trabalho, tais como:

- O método de aprendizagem por competência, pressupondo que um aprendiz para realizar determinada atividade deve possuir um conjunto de habilidades e técnicas. Nesse trabalho, o método de aprendizagem por competência é utilizado com um módulo a ser incorporado na arquitetura modular de um ambiente de aprendizagem. Esse módulo permite o cadastramento de competências e o seu relacionamento com disciplinas e conceitos. O levantamento de Competências, disciplinas e conceitos é contribuição deste trabalho mediante análise de grades curriculares;
- A categorização de conceitos de programação Java, baseada na definição de James Gosling, criador da linguagem. A partir da analise de James Gosling, este trabalho faz uma categorização de conceitos entre conceitos dependentes e conceitos independentes, estruturados e orientados a objetos e os níveis de prioridades que cada conceito tem com relação a outro conceito;
- E a Pirâmide Conceitual como alternativa ao planejamento da aprendizagem e identificador de lacunas conceituais A Pirâmide Conceitual é elaborada a partir do levantamento de Disciplinas, Competências e Conceitos. Para cada Disciplina, temos um conjunto de Competências e para cada Competência temos um conjunto de conceitos, formando pequenas pirâmides e ao final formando uma grande Pirâmide Conceitual.

Este trabalho dá continuidade ao desenvolvido na Dissertação de Mestrado de Locatelli (2007), que apresentou a proposta de uma aplicação web, tendo como foco principal a avaliação continuada da aprendizagem em um curso específico de graduação de computação seguindo os princípios da aprendizagem significativa.

1.3 RELEVÂNCIA DO SISTEMA

Conforme Xavier, Garcia e Silva (2004) conceitos de programação são peças fundamentais na formação de um profissional de Tecnologia da Informação. No entanto, são

conceitos que apresentam maior nível de dificuldade de aprendizado, tanto por apresentar um novo paradigma como a aplicação prática dessas novas teorias. A conseqüência da ausência de conceitos aprendidos previamente são as lacunas de aprendizagem que se formam ao longo do tempo. Estas lacunas dificultam a aprendizagem e levam até à desistência.

No entanto, muitas questões metodológicas têm sido discutidas a fim de melhorar o processo de ensino e aprendizagem (PRETO; GUEDES; CORDEIRO, 2007). Para combater essas lacunas, as Tecnologias de Informação e Comunicação - TICs estão cada dia mais presentes no processo de ensino-aprendizagem, uma vez que propiciam recursos importantes para a elaboração e distribuição de conteúdos educacionais, dando flexibilidade a esse processo (CARVALHO et al., 2007). Ferramentas computacionais como o C-Tutor (SONG et al., 1997) e SICAS (GOMES; MENDES, 2000), por exemplo, têm sido desenvolvidas, com o propósito de facilitar o aprendizado de lógica e linguagens de programação. Segundo Barros e Delgado (2006), tais ferramentas devem permitir ao estudante adquirir melhor e mais rapidamente, as habilidades necessárias para programação.

Essas ferramentas podem ser classificadas em:

- Ambientes integrados de programação, por exemplo, o sotware IBM, Rational Application Developer (RATIONAL APPLICATION DEVELOPER, 2009);
- Editores avançados de programas, como as ferramentas Eclipse (Eclipse, 2010) e Netbeans (NETBEANS, 2009);
- Compiladores, tais como o GCC (GCC, 2009) e GCJ (GCJ, 2009) e;
- Depuradores automáticos.

O estudo de Ambientes Virtuais de Aprendizagem é de extrema relevância, pois são programas de computadores que auxiliam na montagem de cursos de Educação a Distância, gerenciamento de conteúdo por parte dos professores, administração dos cursos e permite acompanhar o andamento da aprendizagem de um aluno. O fator interessante e atraente de um Ambiente Virtual de Aprendizagem é a interatividade e o trabalho coletivo, formando fóruns e comunidades virtuais de aprendizagem.

Entretanto, nos Ambientes Computacionais de Aprendizagem, estabelecer a medida do conhecimento do estudante em determinado conteúdo é útil para que o sistema possa fazer uso de técnicas para adaptar o conteúdo, possibilitando que a aprendizagem seja significativa (PIMENTEL; OMAR, 2007). As avaliações contínuas que ocorrem durante o processo de ensino e aprendizagem são essenciais para o mapeamento do conhecimento do estudante, de

modo que o seu modelo de aprendizagem possa ser estabelecido. Um ambiente de avaliação da aprendizagem com objetivos de mensurar o grau de aprendizagem e dirigir esse aprendizado serve como norteador do processo de ensino-aprendizagem distinguindo a forma de avaliação de uma tradicional avaliação somativa para uma avaliação formativa.

1.4 ESTRUTURA DOS CAPÍTULOS

Para melhor entendimento das pesquisas realizadas por este trabalho, descreve-se no Capítulo 2 sobre as principais referências que contribuíram e serviram de motivação e suporte para a realização desta pesquisa.

No Capítulo 3, apresenta-se a arquitetura da ferramenta desenvolvida por Locatelli (2007), bem como o conceito de nível de aquisição de conhecimento e as implementações sugeridas baseada em uma arquitetura modular. Na seqüência do capítulo 3, apresenta-se a aplicação do Método de Aprendizagem por Competências no contexto educacional, com foco na aprendizagem de programação e a sua implementação, como um módulo adicional na ferramenta abordada por este trabalho.

Já no Capítulo 4, apresenta-se um estudo sobre a Pirâmide Conceitual como mecanismo para auxiliar no planejamento de aprendizagem e identificador de lacunas conceituais perante as disciplinas voltadas para programação de um curso de computação.

No Capítulo 5, este trabalho apresenta o estudo de caso realizado durante o segundo semestre de 2009, baseado nas atividades de estágio docente obrigatório do programa de Mestrado em Engenharia Elétrica da Universidade Presbiteriana Mackenzie.

E enfim, o Capítulo 6 descreve os resultados e a conclusão obtida deste trabalho e propostas de trabalhos futuros a serem desenvolvidos.

2 REFERENCIAL TEÓRICO

A relevância deste trabalho está em buscar técnicas que pudessem colaborar com um ambiente computacional de aprendizagem já existente. A partir daí, espera-se fornecer um conjunto de informações que auxiliem na aprendizagem de programação de computadores. Sendo assim, O Capítulo 2 apresenta as principais técnicas estudadas e trabalhadas ao longo da construção desta pesquisa.

2.1 AMBIENTES DE APRENDIZAGEM

A área de Informática na Educação - IE vem mostrando que é possível o desenvolvimento de um ambiente de aprendizagem que permita que cada aluno seja devidamente auxiliado e direcionado a caminhos personalizados de aprendizagem que reduzam e propiciem identificar lacunas conceituais que se formam ao longo dessa aprendizagem.

Os principais resultados da IE advêm das pesquisas em Sistemas de Tutoria Inteligentes, que são programas de computadores que dão suporte a aprendizagem utilizando técnicas de Inteligência Artificial. Segundo Gavidia e De Andrade (2003), a estrutura de um Sistema Tutor Inteligente possui informações relevantes sobre o conhecimento do aluno, podendo:

- Direcioná-lo adequadamente;
- Propiciar comunicação mais facilitada entre aluno e tutor;
- Oferecer tarefas de aprendizagem não pré-determinadas e *feedback* mais detalhado da situação corrente de um aluno.

Sua arquitetura separa o assunto que vai ser ensinado das técnicas de ensino e proporciona um ensino adequado a cada aluno através dos seus quatro modelos: do aluno, onde guarda as informações referentes às características do aluno; do tutor, no qual possui o conhecimento de como se utilizar estratégias de ensino em função de um perfil de aluno; do domínio, onde o conhecimento do assunto fica armazenado em formato adequado, por exemplo, o de regras de produção; e da interface, caracterizada como a mediação entre aluno e tutor.

No trabalho desenvolvido por Locatelli (2007) foi apresentado o desenvolvimento de uma ferramenta, denominada ACAvA, que se encaixa nessa categoria de ambiente de aprendizagem, e que acompanha o nível de aquisição de conhecimento de um aluno para conceitos diversos. O aplicativo permite que professores e alunos acompanhem a evolução do

aprendizado em uma turma, mediante exercícios e medidas de NAC (nível de aquisição de conhecimento) e a modelagem de conceitos de uma disciplina mediante o uso de mapas conceituais, através do cadastro de disciplinas, conceitos e o relacionamento entre eles.

A arquitetura do modelo proposto por Locatelli (2007) se baseia no paradigma instrucionista de aprendizagem proposto por Pimentel e Omar (2007) que sugere uma organização modular da seguinte forma: módulos de administração, comunicação, acompanhamento, aprendiz, avaliação e conhecimento.

No entanto, para um aluno ser aprovado em uma determinada disciplina, ele precisa de um conjunto de habilidades e competências que o possibilite aprender e assimilar o conteúdo da matéria necessário para a atividade profissional que irá exercer. Nesse momento existe a união dos trabalhos de Locatelli (2007) e Feitosa (2009), proposta por esta pesquisa.

Feitosa (2009) propõe um modelo computacional que represente as pessoas por suas competências profissionais e comportamentais, de forma a sugerir um padrão de medida para que alunos, empresas e instituições de ensino tenham um mesmo referencial para aferição de conhecimentos, habilidades e atitudes de determinada pessoa.

O trabalho de Pimentel, Feitosa e Omar (2008) apresenta uma arquitetura para integrar um Ambiente de Aprendizagem a um Ambiente de Certificação, conforme a Figura 1:

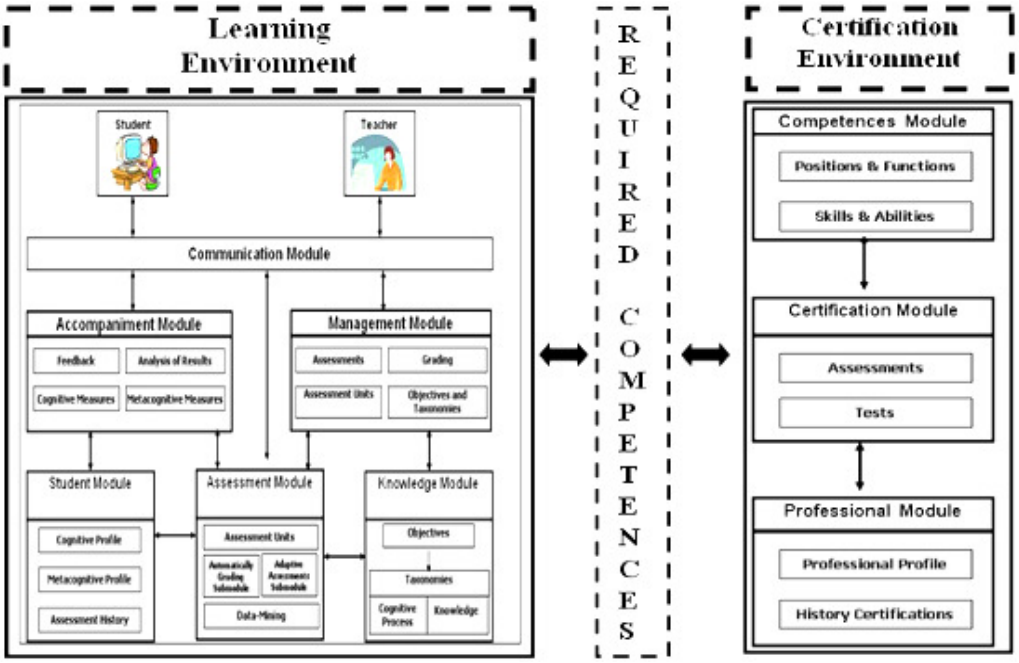

Figura 1. Arquitetura de um Ambiente de Aprendizagem e de um Ambiente de Certificação (PIMENTEL; FEITOSA; OMAR, 2008).

A Figura 1 apresenta, ao lado esquerdo, a arquitetura modular de um ambiente de aprendizagem com os seguintes módulos: Módulo de Comunicação, Módulo de Administração, Módulo de Avaliação, Módulo de Acompanhamento, Módulo do Aprendiz e o Módulo de Conhecimento.

Já ao lado direto é apresentado o ambiente de certificação que se integra com o ambiente de aprendizagem através das competências requeridas. O ambiente de certificação apresenta os módulos de Competência, Certificação e Profissional.

A partir daí, maturou-se a proposta de implementar e desenvolver o módulo de competências dentro da ferramenta proposta por Locatelli (2007). Uma segunda versão do ambiente passou a ser estudada e desenvolvida com base na inclusão do módulo de competências no ACAvA e tomando como suporte as disciplinas de programação do curso de Ciência da Computação da Universidade Presbiteriana Mackenzie, envolvendo a análise do seu conteúdo programático, bem como seus conteúdos, levantando matérias e temas relevantes no âmbito computacional. A notação utilizada nesta pesquisa para referenciar a nova versão do ACAvA é ACAvA 2.0. Um fator muito importante nesse cenário é o conteúdo, pois a motivação de um aluno em utilizar uma ferramenta de apoio está no conteúdo.

E como sugere Perrenoud (2000), definiu-se um conjunto de objetivos principais, que nesse contexto são as disciplinas que envolvem programação e um conjunto de competências mais específicas a trabalhar em formação contínua, como parte do desenvolvimento do módulo de competências.

2.2 PARADIGMAS DE LINGUAGENS

Como parte do desenvolvimento e com uma importância elevada neste cenário, estudou-se os Paradigmas de Linguagens de Programação existentes, dando destaque para o Paradigma de Orientação a Objetos e o de Programação Estruturada.

Um Paradigma de Programação estabelece a estruturação que o programador vai utilizar para o seu programa. Em alguns casos, uma linguagem de programação pode suportar mais de um paradigma. Porém, o que diferencia um paradigma do outro, muitas vezes, são suas limitações. Contudo, a habilidade e competência em manusear determinado paradigma vão determinar a eficácia do estudante em utilizar cada paradigma.

Abaixo seguem alguns exemplos de paradigmas existentes:

- Programação estruturada;
- Programação imperativa;
- Programação de passagem de mensagens;
- Programação procedural;
- Programação orientada a fluxos;
- Programação escalar;
- Programação restritiva;
- Programação orientada a aspecto;
- Programação orientada a regras;
- Programação orientada a tabelas;
- Programação orientada a fluxo de dados;
- Programação orientada a políticas;
- Programação orientada a testes;
- Programação genérica;
- Programação multiparadigma.

Ao longo da vida profissional de um programador ele se deparará com diversos desses paradigmas. O aprendizado de programação aqui proposto permite que esse processo seja incremental sobre os conceitos necessários para a aprendizagem de um paradigma a partir dos conceitos previamente dominados. O Paradigma abordado ao longo deste trabalho é o de Orientação a Objetos. Nesse paradigma objetiva-se identificar o melhor conjunto de objetos para descrever um sistema de software, havendo interação entre esses objetos a partir da troca de mensagens entre eles. Tais objetos são frutos da implementação de Classes, que vai definir o comportamento dos objetos.

Alguns conceitos de Orientação a Objetos, como Objeto e Classe são fundamentais, no entanto outros conceitos são essenciais também e compõem esse paradigma, como: Subclasse, Atributos, Métodos, Herança, Associação, Polimorfismo, Encapsulamento e Abstração. Segundo Ricarte (2001) a vantagem da programação orientada a objetos em relação a outros paradigmas de programação que também permitem a definição de estruturas e operações sobre essas estruturas está no conceito de herança, mecanismo através do qual definições existentes podem ser facilmente estendidas.

Já Fujii (1997) considera que um dos grandes diferencias da programação orientada a objetos está na produtividade de programação devido a reutilização de código, o rápido

desenvolvimento de aplicativos devido a possibilidade de união de aplicativos de diferentes programas e a redução do custo de manutenção.

São exemplos de linguagens de programação Orientada a Objetos:

- C++;
- C#;
- Java;
- Object Pascal;
- Python;
- Ruby;
- Smalltalk.

2.3 LINGUAGEM DE PROGRAMAÇÃO JAVA

Escolhida como suporte para o processo de ensino e aprendizagem em muitas universidades e amplamente utilizada no mercado de tecnologia de informação, a linguagem de programação Java, escolhida como ferramenta de trabalho por esta pesquisa, foi desenvolvida em meados dos anos 90 pela equipe de James Gosling, da empresa Sun Microsystems. Lançada em 1995, a plataforma Java foi adotada rapidamente, mais do que qualquer outra linguagem. Em 2004, atingiu a marca de três milhões de desenvolvedores.

Java possui características muito peculiares, tais como:

- Orientação a objetos;
- Portabilidade, indicando sua independência de plataforma;
- Sintaxe semelhante à das linguagens C/C++;
- Inúmeras bibliotecas ou APIs;
- Possibilidade de criação de programas multitarefas;
- Facilidade ao acesso de recursos de rede;
- Segurança;
- Diversos *frameworks* para facilitar o desenvolvimento de aplicações;
- Vários ambientes de desenvolvimento.

Os programas Java são traduzidos para uma representação conhecida por *bytecode*. Os *bytecodes* são interpretados pela JVM (*Java Virtual Machine*), o que proporciona uma compilação dinâmica.

A Sun Microsystems oferece alguns tipos de certificações na linguagem Java, sendo que cada certificação testa os conhecimentos dos aprendizes em determinado domínio dentro da plataforma.

No contexto educacional, o uso da linguagem Java como suporte ao ensino de programação ao longo dos cursos tem gerado alguns problemas devido à necessidade do aprendizado de conceitos de programação que são independentes da linguagem. De certa forma, não havendo a devida separação desses conceitos, fortes lacunas podem se formar no caminho da aprendizagem ideal. Na seção que se segue faz-se um estudo sobre a apresentação da mesma em sites como o do James Gosling e da empresa Sun Microsystems, estabelecendo e analisando as relações entre os conceitos e a sua importância naquele contexto.

2.3.1 Divisão de conceitos por Gosling

Como suporte às disciplinas, competências, conceitos e exercícios e adotada como ferramenta de aprendizagem em muitos cursos de computação, a linguagem de programação Java é estudada nesta seção, que realiza um levantamento, análise e categorização da linguagem a partir do site de seu criador, James Gosling.

James Gosling (GOSLING, 2009) apresenta hierarquicamente sua abordagem sobre programação Java em diversas camadas. Ele organiza os conceitos de maneira crescente, apresentando os conceitos mais fundamentais em primeiro lugar, seguindo para conceitos mais complexos gradativamente.

Inicialmente, são apresentados conceitos básicos da Linguagem de Programação Java, como a definição de palavras-chave, como se realizam comentários ao longo do código, tipos de dados, entre outros. Na seqüência, o nível de dificuldade de entendimento dos conceitos vai aumentando com a vinculação entre os mesmos e o relacionamento com as características peculiares à linguagem e sua filosofia.

Nessa divisão há a exploração de um conceito principal, denominado como “conceito-alvo”. Por exemplo, o conceito “Controle de Repetição” é o conceito-alvo, no qual sua definição deve ser apresentada, com prioridade maior e em seguida, suas derivações, como os comandos if, while e for, por exemplo. A Tabela 1 apresenta uma descrição adaptada dessa divisão:

Tabela 1. Divisão Conceitual de Gosling (GOSLING, 2009).

Conceitos-Alvo	Conceitos que formam o conceito-alvo
Linguagem	Palavras-Chave, Comentários, Javadoc, Main, Garbage, Collection, Variáveis, Escopo de Variáveis e Jar.
Tipos de Dados	Boolean, Int, Short, Char, Float, Double, Long, String, Date e Calendar.
Operadores	Atribuição, Incremental, Decremental, Aritméticos e Lógicos.
Controles de Repetição	While, For
Classes	Definição de Classe, Construtores, Métodos, Return, Recursividade, This, Objeto, Polimorfismo e Import.
Collections	Array, Iterator, LinkedList, HashMap, TreeMap, Sort, Stack, Deque, Vector

Ao redor de um conceito-alvo existem outros conceitos, que são a base de sustentação da aprendizagem deste conceito-alvo. A Figura 2 apresenta como exemplo o conceito-alvo "Classe" representado através de um Mapa Conceitual indicando o vínculo entre os conceitos, ou seja, a maneira como as idéias estão organizadas.

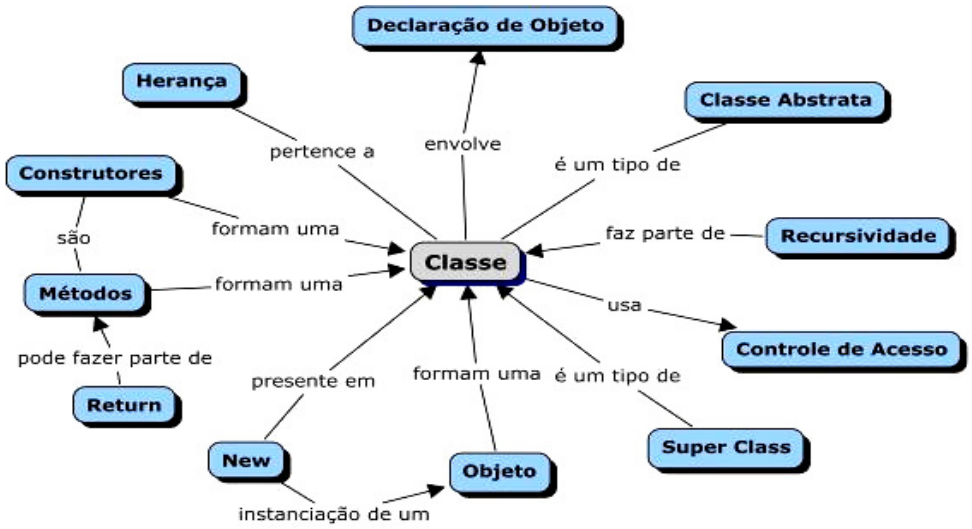

Figura 2. Representação de Conceitos que cercam o conceito-alvo Classe.

De forma recursiva, estes conceitos que cercam o conceito-alvo "Classe" evoluem.

A partir da divisão adaptada de Gosling, esta pesquisa define três fases de categorização:

- Classificar os conceitos como dependentes ou independentes;
- Determinar os níveis de prioridades;

- Classificar os conceitos como Estruturado ou Orientado a Objetos;

Portanto, para a continuação da categorização, definiu-se conceito-alvo como um conceito principal que envolve outros conceitos para o seu entendimento. Se um aprendiz deseja aprender o conceito de Método, por exemplo, ele deve inicialmente entender o que é Função, Variáveis, Tipos de Dados, Tipos de Retorno, Declaração de Variáveis e Operadores.

2.3.2 Conceitos dependentes e conceitos independentes

O universo da linguagem de programação Java é muito extenso. Porém, no contexto semântico de programação de computadores, os conceitos utilizados em Java fazem intersecção com os conceitos de muitas outras linguagens de programação.

Esse trabalho classifica como “conceitos dependentes”, aqueles que dependem de uma ou várias linguagens de programação, mas não de forma genérica, ou seja, são conceitos pertencentes ao universo de determinadas linguagens. Já os “conceitos independentes” são aqueles que pertencem ao escopo geral das linguagens de programação, ou seja, não são conceitos exclusivos de uma determinada linguagem.

Conceitos como os Tipos de dados, pertencem a classe de conceitos independentes, pois em todas as linguagem de programação é necessário saber do que se trata um Tipo de dado para programar. Porém, existem Tipos de dados que dependem de características da linguagem de programação. Os tipos de dados Date e Calendar seguem essa linha. Os dados dependentes são do tipo:

- Calendar.

Já os Tipos de Dados Independentes seriam:

- Boolean, Int, Char, Float, Double;

Já os conceitos envolvidos com o conceito-alvo de Operações, seguem o caminho dos conceitos independentes. São utilizados por todas as linguagens e não dependem exclusivamente da linguagem Java. Os conceitos independentes envolvidos com o conceito-alvo Operadores são: Atribuição, Aritméticos e Lógicos.

2.3.3 Níveis de prioridades

Um conceito deve ser ensinado, considerando sua prioridade, dentro de uma hierarquia. Esse conceito assimilado inicialmente deve ser a base de sustentação de um próximo conceito. Segundo Moreira (1999), os novos conhecimentos que se adquirem relacionam-se com o conhecimento prévio que o aluno possui.

Dessa forma, a aprendizagem significativa, principal conceito da teoria de aprendizagem de Ausubel (1968), ocorre quando o novo conhecimento se apóia em conceitos relevantes preexistentes na estrutura cognitiva do aluno. Estruturas cognitivas são como estruturas hierárquicas de conceitos, que são representações de experiências sensoriais do indivíduo. A ocorrência da aprendizagem significativa implica no crescimento e modificação do conceito subsunçor. A partir de um conceito geral o conhecimento pode ser construído de modo a relacioná-lo com novos conceitos facilitando a compreensão das novas informações, o que dá significado real ao conhecimento adquirido.

Dessa forma, este trabalho propõe que níveis de prioridades sejam atribuídos aos conceitos, de modo que, em uma estrutura geral, possa seguir-se um planejamento no momento da aprendizagem. A Tabela 2 apresenta a definição de prioridades.

Tabela 2. Definição de níveis de prioridades.

Prioridades	Definição
1	Conceitos Independentes de Linguagem, mais gerais. Definições. Conceitualização.
2	Conceitos Independentes da Linguagem mais específicos.
3	Conceitos Independentes da Linguagem com maior complexidade(começa a envolver dependência).
4	Conceitos Dependentes, implicitamente vinculados as características da linguagem(como a Orientação a Objetos).
5	Conceitos Independentes vinculados às características da linguagem.

A Tabela 3, por sua vez, apresenta a aplicação desta categorização de níveis de prioridades considerando a divisão que Gosling faz a respeito das características peculiares da linguagem de programação Java.

Tabela 3. Conceitos-alvo Linguagem e Estrutura de dados e os conceitos que os cercam.

Linguagem		Estrutura de Dados	
Conceitos	**Prioridades**	**Conceitos**	**Prioridades**
Palavras-chave	5	Array	3
Comentários	1	LinkedList	3
Javadoc	4	Pilha	3
Main	3	Fila	3
Variáveis	1	HashMap	3
Garbage Collection	5	Arvores	3
Escopo de Variáveis	1	Vetor	3
Jar	5	Collections	4

O conceito-alvo “Linguagem”, que lhe tem atribuído a prioridade 1, envolve diversas prioridades diferentes, pertencentes aos conceitos que lhe cercam. Isso ocorre porque uma linguagem de programação, no caso Java, envolve conceitos que são independentes da linguagem de programação, ou seja, conceitos Estruturados. Evidentemente, Java também possui suas características próprias.

Os conceitos-alvo também assumem um nível de prioridade, pois o que é considerado conceito-alvo em determinado momento da aprendizagem, pode ter seu status alterado para um conceito que cerca um conceito-alvo. Dessa forma, no caso do conceito “Classe”, ele assume o nível de prioridade 2.

Da mesma forma, o conceito “Recursividade”, considerado Estruturado, pois independe da linguagem de programação, tanto faz Java ou C, possuem se explorado, inúmeros conceitos que compõem sua formação. Nesse caso, o conceito de Método, é fundamental para que se possa ter total entendimento do que significa Recursividade.

2.3.4 Conceitos estruturados e de orientação a objetos

A linguagem de programação Java segue o paradigma de Orientação a Objetos. No entanto sugere-se a distinção entre os conceitos Estruturados e Conceitos de Orientação a Objetos. Apesar de a programação Java envolver fundamentalmente Orientação a Objetos, é interessante notar como, mesmo assim, envolve conceitos Estruturados. Um exemplo disso é o conceito de Classe, que traz consigo os conceitos de Método e Return. Os dois conceitos, Classe e Método, estão presentes no paradigma Estruturado e no paradigma de Orientação a Objetos. Não são caracterizados por exclusividade de nenhum dos dois paradigmas, formando uma intersecção entre os dois paradigmas.

Já o conceito de Declaração de Objetos, por sua vez, é peculiar ao paradigma de Orientação a Objetos. Os conceitos que se encaixam no paradigma Estruturado para o conceito-alvo Classe são:

- Métodos, Recursividade, final, null e Controle de acesso.

Já para o paradigma Orientado a objetos, temos:

- Definição de Classes, Construtores, This, Objeto, Declaração de Objetos, New, Herança e Super Class.

Java possui peculiaridades, principalmente por trabalhar com o paradigma de Orientação a Objetos, portanto, conceitos como criação de objetos, instanciações, herança, referências, pertencem à classe, aqui definida, de Orientação a Objetos. Os conceitos pertencentes ao grupo de conceitos Estruturados são:

- Abstract, const, finally, int, return, throws, assert, continue, float, short, default, for, long, static, try, break, do, void, byte, double, if, else, final.

Já os conceitos pertencentes ao grupo de Orientados a objetos são:

- Interface, new, super, implements, extends, instanceof, this.

2.3.5 Análise de exemplos do site da Sun

A certificação tem diversas importâncias para um profissional de Tecnologia da Informação, tais como: melhor posicionamento no mercado de trabalho; testar o quanto o profissional conhece da tecnologia; e ser um selo de qualidade, tornando-se um diferencial.

Este trabalho apresenta uma análise baseada no site da SUN (SUN, 1995) que oferece a apresentação de conceitos e exercícios de preparação para a certificação de programador Java, baseado na maneira como foram divididas as seções de exercícios e exemplos em tópicos e subtópicos.

O conteúdo de preparação para certificação Java é estruturado e apresentado da seguinte forma:

- Declarações, Inicializações e Escopo;
- Controle de Fluxo;
- API;
- Concorrência;
- Conceitos de Orientação a Objetos;
- Collections/Generics;

- Conceitos Fundamentais.

O comportamento da primeira seção, que engloba os assuntos Declarações, Inicializações e Escopo, se dá da seguinte forma: As seções apresentadas e descritas acima são divididas em itens. A primeira seção possui 6 itens, conforme descrito na Tabela 4.

Tabela 4. Conteúdo dos itens que envolvem a primeira seção.

Item 1	Declaração de Classes; Métodos e Classes Abstratas; Nested Classes; Interfaces; Enum Types; Packages; Utilizando Package Members(imports, static imports).
Item 2	Definindo uma Interface; Implementando uma Interface; Classes e Métodos Abstratos.
Item 3	Tipos de Dados; Arrays; Enum Types; Instâncias e Class Members; Variáveis; Declarando Variáveis.
Item 4	Instâncias e Class Members; Passando Informações para Métodos e Construtores.
Item 5	Sobrescrevendo Métodos; Definindo Métodos; Retornando valores de um método
Item 6	Criando construtores para uma classe; Utilizando a palavra-chave "Super"; Criando Objetos; Nested Classes.

Observa-se nesse caso que os conceitos estão sempre se relacionando. O conceito de “Nested Classes”, que significa a possibilidade criar uma classe dentro de outra, por exemplo, pertence tanto ao Item 1, quanto ao Item 6. Analisando os conceitos que formam os itens 1 e 6, concluímos que é coerente o conceito “Nested Classes” pertencer aos dois itens, pois ambos abordam conceitos ligados fortemente ao conceito de “Classes”. Esse mesmo caso repete-se ao conceito de “Instâncias e Class Members”, que compõem os itens 3 e 4. Definimos aqui, esses conceitos, pertencentes a mais de um item, como “conceitos conectores”, ou seja, eles são os responsáveis pelo relacionamento entre os itens de uma seção, conforme demonstrado na Figura 3:

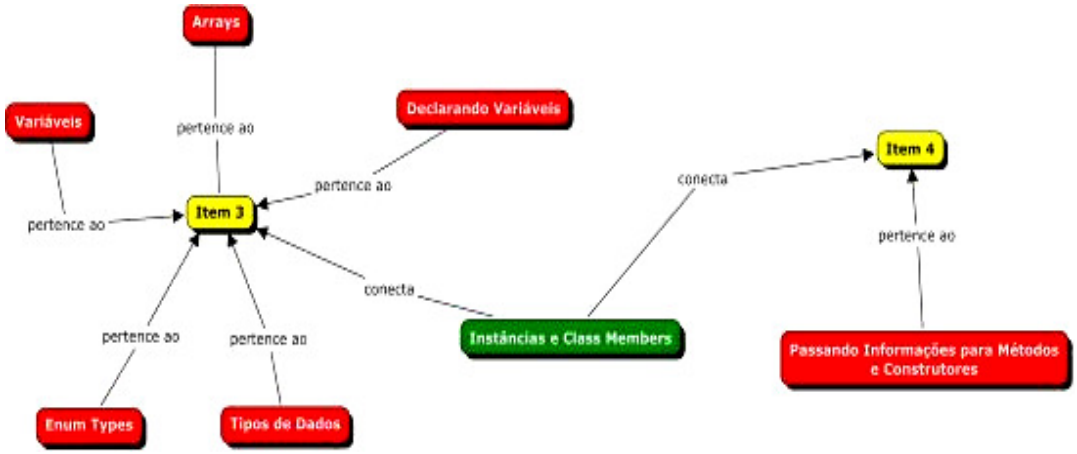

Figura 3. Mapa Conceitual explorando a existência de um conceito "conector".

Essa definição aplica-se a primeira seção de conteúdos, pois, gradativamente, as outras seções se tornam mais especificas, minimizando os casos de conceitos repetidos em mais de um item.

Ainda nesta primeira seção, exercícios, envolvendo os diversos conceitos relacionados aos itens, são sugeridos, para efetuar o aprendizado destes conceitos, conforme descrito nas questões e respostas abaixo:

1. *O termo "instance variable" é outro nome para ___.*
2. *O termo "class variable" é outro nome para___.*
3. *A variável local armazena estado temporário; ela é declarada dentro de um___.*
4. *Uma variável declarada dentro de um abre e fecha parenteses de uma assinatura de método é chamada de ____.*

E são apresentadas as respostas, conforme descrito abaixo:

1. *O termo "instance variable" é outro nome para **non-static field**.*
2. *O termo "class variable" é outro nome para **static field**.*
3. *Uma variável local armazena estados temporários; ela é declarada dentro de um **method**.*
4. *Uma variável declarada dentro de um abre e fecha parenteses de uma assinatura de método é chamada de **parameter**.*

Conclui-se que os exercícios apresentados são fundamentais na aprendizagem dos conceitos, na medida em que, eles proporcionam que o aprendiz aprenda a relacionar os

diversos conceitos de programação. Um exemplo claro disso é a questão 6, na qual envolve o relacionamento entre o conceito de “Tipos de Dados” e “Package”, ou seja, criando uma relação entre o Item 3 e o Item 1 da Tabela 4.

2.4 MAPAS CONCEITUAIS

A teoria que envolve os Mapas Conceituais, originalmente baseada na teoria da aprendizagem significativa de Ausubel, trata-se de uma representação de um conjunto de conceitos elaborados de tal forma que os relacionamentos entre eles sejam evidentes, através de frases que determinam suas relações.

Os mapas conceituais, segundo Moreira e Mansini (2001) trabalham também como ferramenta de avaliação. Porém não é avaliação no sentido de atribuir nota, mas sim no sentido de se obter informações sobre o tipo de estrutura que o aluno vê para um dado conjunto de conceitos. Dessa forma pode-se construir o mapa conceitual de um aluno através de respostas em entrevistas.

A finalidade do uso dos mapas conceituais na ferramenta proposta neste trabalho é possibilitar que o professor, ao identificar problemas de aprendizagem em um conceito, desenvolva um levantamento de todos os pré-requisitos que são necessários para a aprendizagem significativa deste conceito. Foi de extrema valia a construção de mapas conceituais nesta pesquisa, possibilitando visualizar como os conceitos de programação estão relacionados.

Para a elaboração e construção dos mapas conceituais das disciplinas, competências e conceitos utilizou-se a ferramenta Cmaptools (CMAPTOOLS, 2009).

No ACAvA desenvolvido por Locatelli (2007), a relação entre conceitos é inicialmente estabelecida mediante o cadastro de conceitos no módulo de conhecimento, conforme a Figura 4:

Figura 4. Cadastro de conceitos no ACAvA.

No ACAvA 2.0, o cadastro de conceitos vem acompanhado de um link, que aqui chamamos de link tutorial, o qual refere-se a um encaminhamento a algum site explicativo ou Objeto de Aprendizagem, considerado adequado pelo professor daquele conceito.

Após o cadastramento de conceitos é possível estabelecer a relação entre os conceitos ao acessar o menu "Associar Conceito-Dependência", conforme indica a Figura 5:

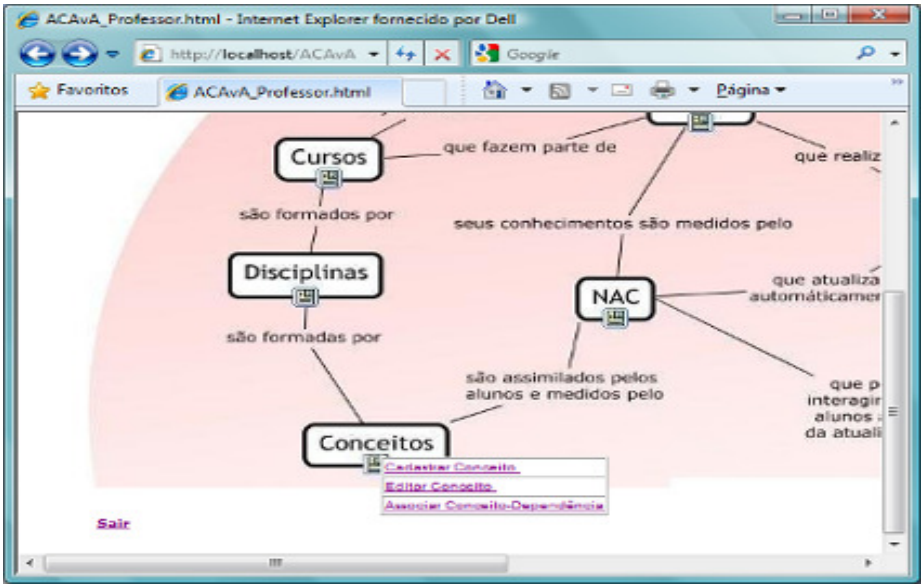

Figura 5. Menu de acesso para relacionar os conceitos.

Para efetivamente definir as relações entre os conceitos, escolhe-se um conceito em uma lista, e determinam-se quais são os conceitos pré-associados e pós-associados, como a Figura 6 apresenta:

Figura 6. Estabelecendo conceitos pré e pós-associados.

Além do cadastramento de conceitos, a ferramenta permite o cadastramento de cursos, disciplinas, competências, alunos e exercícios.

2.5 ONTOLOGIAS

Existem várias definições para o termo ontologia, sob diversos pontos de vista. Porém, há um consenso de que ontologia trata-se de um modelo de dados que representa um conjunto de conceitos dentro de um domínio e os relacionamentos entre estes. Para Russel e Norvig (1995), ontologia é uma teoria particular, expressa por um vocabulário da natureza do ser e do existir. Já Gruber (1993) definiu ontologia como sendo a especificação de uma conceitualização.

A finalidade da elaboração de ontologias nesse trabalho se dá pelo motivo da construção da ontologia das disciplinas que envolvem programação de computadores do curso de Ciência da Computação. Por exemplo, o currículo de uma disciplina deve ser modelado e organizado de forma a facilitar o aprendizado e as decisões instrucionais do sistema (SANTOS; BARROS; VALENTE, 2001).

Segundo Locatelli (2007) para a elaboração de uma ontologia é necessária a definição de etapas bem estruturadas:

- Construir um modelo de domínio, que representa um conjunto de itens de informação presentes em determinado contexto;
- Especificar requisitos caracterizados por identificar claramente os seus propósitos e os usos esperados pela ontologia;
- Realizar a captura da ontologia que estabelece os conceitos e relações do universo do discurso e a aquisição de dados;
- Formalizar a ontologia, definindo sua representação;
- Definir as classes, identificar sua hierarquia e definição de suas propriedades. As propriedades podem possuir atributos (também conhecidos como *slots*) que descrevem as propriedades que poderão ser particulares para cada instância de uma classe e restrições (também chamados de *facets* ou *role restrictions*).

Para a elaboração das Ontologias, utilizou-se a ferramenta Protégé (PROTÉGÉ, 2009), que é um editor de Ontologias, open source, desenvolvido pela Universidade de Stanford.

Inicialmente definiu-se um dicionário de Termos, de forma que o sistema possa reconhecer termos de mesmo significado, porém com sintaxes diferentes:

- Curso = Especialidade;
- Disciplina = Matéria;
- Competência = Habilidades, técnicas;
- Conceito = Definição, tópico;

Uma visão geral da ontologia pode ser exibida através do Protégé na aba *Class Browser*, lugar em que são criadas as classes, como exibe a Figura 7:

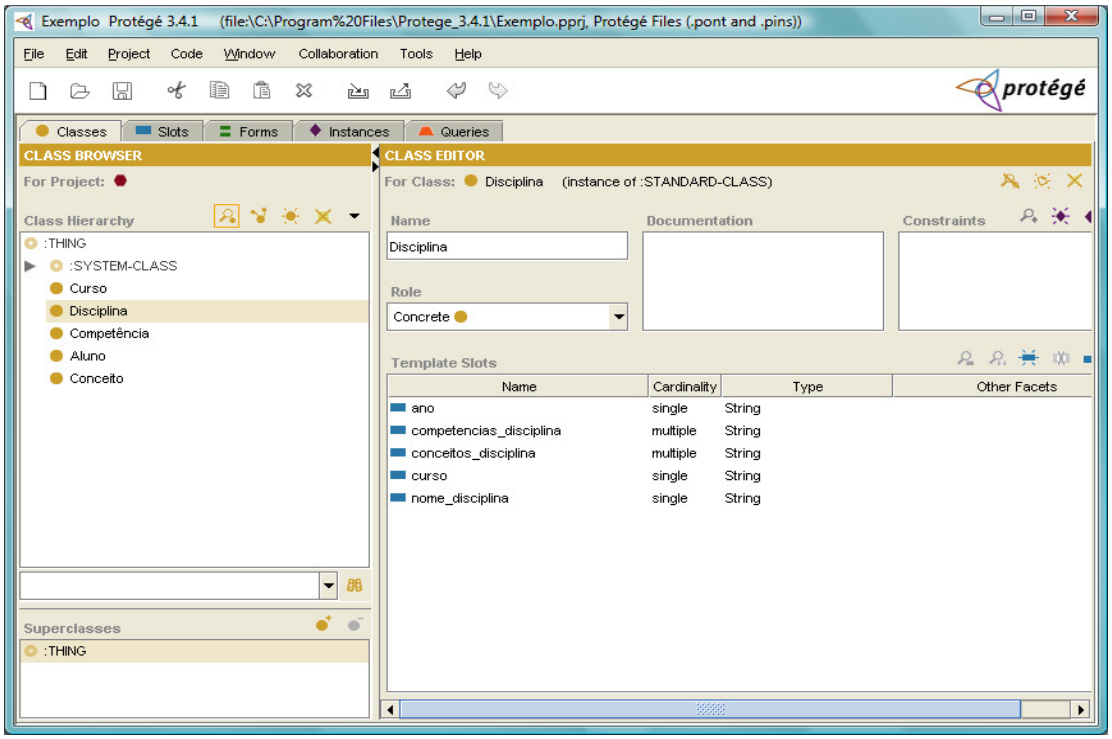

Figura 7. Definindo Atributos para a Classe Disciplina.

Foram criadas 5 classes: Curso, Disciplina, Competência, Aluno e Conceito. Para cada uma, um conjunto de atributos é definido, como nome, ano, por exemplo. Porém, os atributos que indicam vínculo ou dependência entre uma classe e outra são os atributos: “depende_de” e “pre-requisito_para”. O atributo “depende_de” significa que existem conceitos que dependem de outros conceitos. Já o atributo “pre-requisito_para” significa que determinado conceito é requisito para outro conceito.

Após a definição de classes e atributos, criam-se as chamadas “Instâncias”, que são objetos específicos de determinada classe. A Figura 8 é um exemplo da criação de uma instância da classe Disciplina para a disciplina Desenvolvimento Orientado a Objetos.

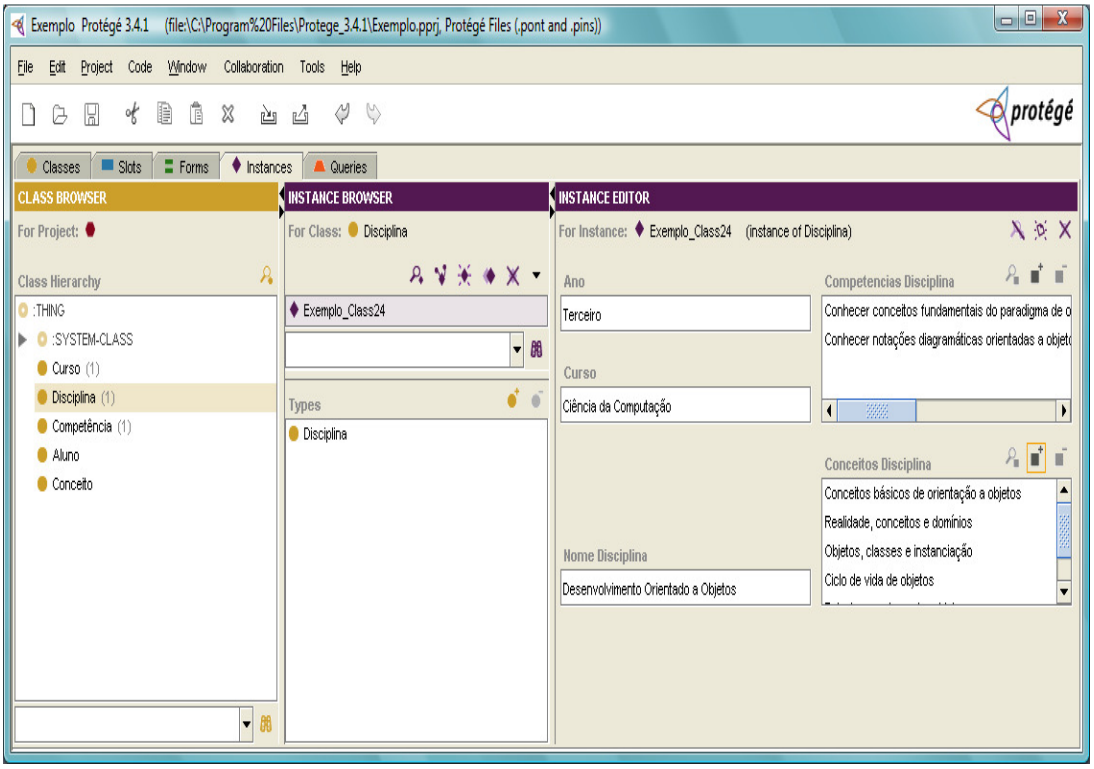

Figura 8. Estabelecendo conceitos pré e pós-associados.

Dessa forma, para cada classe Curso, Disciplina, Competência e Conceito foi instanciado um objeto para representar as disciplinas, competências e conceitos das matérias relacionadas a programação do curso de Ciência da Computação. A construção da ontologia serviu como base para a população da base de dados a fim de representar este cenário.

2.6 TÉCNICAS DE PLANEJAMENTO

Embora este trabalho tenha como principal objetivo a construção do módulo de competências no ambiente proposto por Locatelli (2007), estudou-se também uma solução para o planejamento de aprendizagem de um aluno. Esta pesquisa define no Capítulo 5 o mecanismo de Pirâmide Conceitual que foi influenciada por técnicas de Planejamento em Inteligência Artificial. Segundo Santanchè et al. (2007) a técnica de Planejamento consiste em um conjunto de técnicas que permite a busca de uma solução eficiente para solucionar diversos problemas aplicados a diferentes domínios. Planejamento é um tipo de solucionador de problema onde os agentes fazem o uso de ações a procura de soluções em um espaço de planos, ou seja, envolve determinar um conjunto ordenado de ações que quando executadas por um ou mais agentes a partir de um estado inicial que satisfaça as circunstâncias dadas,

resulte num estado final que satisfaça a meta. Um problema de planejamento geralmente é representado na forma de uma tripla (A, I, G), respectivamente *Action*, *Initial State* e *Goal*, onde *Action* representa as ações que podem ser executadas, formadas por pré-condições e efeitos, *Initial State* é o estado inicial de um determinado problema e *Goal* a meta a ser alcançada.

3 APLICANDO O MÉTODO DE APRENDIZAGEM POR COMPETÊNCIAS EM UM AMBIENTE DE APRENDIZAGEM VIRTUAL

Este trabalho utiliza essencialmente o escopo de um curso de computação, enfatizando as disciplinas que envolvem programação de computadores. Neste capítulo, inicialmente, apresenta-se o ACAvA de Locatelli (2007) e sua arquitetura modular. Em seguida apresenta-se a proposta de desenvolvimento do método de aprendizagem por competências na versão do ACAvA 2.0 através do levantamento das disciplinas, competências, conceitos e exercícios paralelamente a implementação, no sentido de construção de novas páginas PHP, tabelas em banco de dados e diagramas UML, deste novo módulo.

3.1 ACAvA - AMBIENTE COMPUTACIONAL DE APRENDIZAGEM VIRTUAL

Esta seção tem por objetivo inicial apresentar a ferramenta ACAvA de Locatelli (2007). Entretanto, o foco do capítulo é apresentar a modularidade da ferramenta com vistas a proposta de implementação do método de aprendizagem por competências como um módulo adicional na arquitetura do ACAvA 2.0.

O ACAvA de Locatelli (2007) é uma aplicação web baseada no paradigma instrucionalista de aprendizagem. Ou seja, trata-se de um conhecimento concreto que é transferido do professor ao aluno com um plano prévio de ensino, organizado pelos professores, que têm mais experiência em definir as necessidades de seus alunos.

Segundo Locatelli (2007) o ACAvA agrega à teoria tradicional a prática modernizada, via avaliação continuada e é base para desenvolvimento de um sistema que auxilie o professor à mudança do paradigma da sala de aula. O sistema desenvolvido permite a anotação em tempo real do desenvolvimento do aprendizado do aluno.

Além disso, o ACAvA apoia-se no conceito de Mapas Conceituais, que são representações gráficas que indicam relações entre conceitos e representam uma estrutura que vai desde os conceitos mais gerais até os menos abrangentes.

A Figura 9 apresenta a tela inicial do ACAvA:

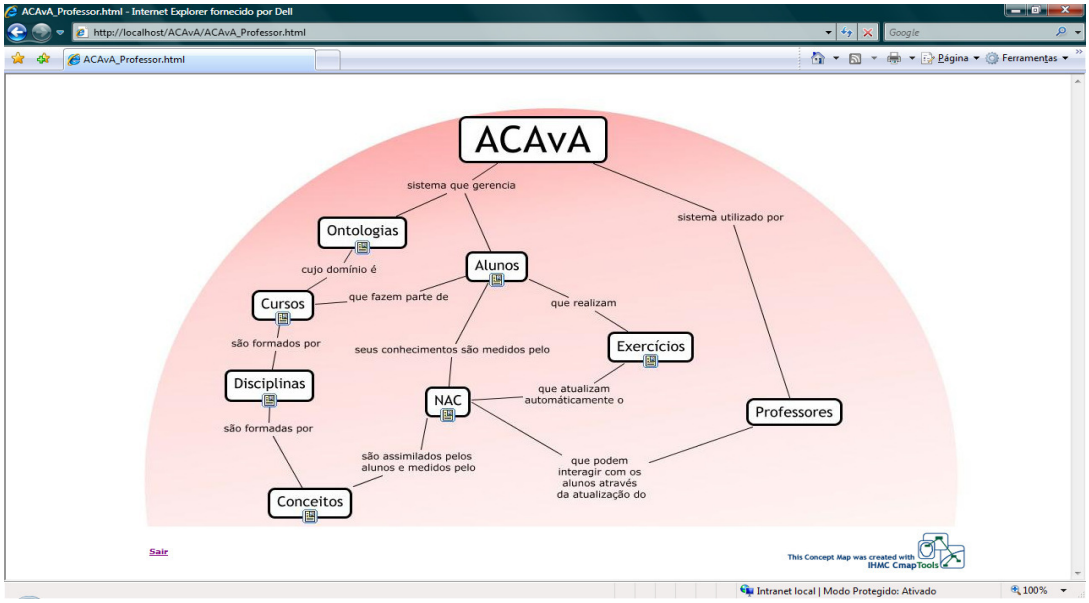

Figura 9. Imagem da tela inicial do software ACAvA e a disposição dos módulos.

O ACAvA também apoia-se no conceito de NAC, ou seja, nível de aquisição de conhecimento, em que cada unidade de avaliação deve, de certa forma, refletir os objetivos de aprendizagem num domínio específico de conhecimentos ou em um conjunto deles (multidisciplinaridade) (PIMENTEL, 2006).

Dentro do ACAvA proposto por Locatelli (2007), o NAC de um aluno varia de 0 a 10. Ao longo de um período, o NAC pode ser atualizado, conforme o desempenho do aluno. Ao final deste período, o NAC não poderá sofrer alterações de modo que se construa um histórico a fim de acompanhar o processo de aprendizagem do aluno.

Conforme explica Locatelli (2007) no ACAvA existem três maneiras de se obter o *NAC*:

1. O professor, mediante o conhecimento que ele possui a respeito do aluno, pode atualizar o seu NAC;
2. O aluno atualiza seu próprio NAC, proporcionando assim exercitar o conhecimento que ele possui dele mesmo, bem como avaliar o que ele tem aprendido e o que já trouxe consigo como conhecimento prévio;
3. O aluno realiza exercícios objetivos, que possuem a diversos conceitos associados. Perante sua resposta, certa ou errada, seu NAC é atualizado automaticamente pelo sistema.

A nova versão do ACAvA irá implementar as características presentes na arquitetura de um ambiente computacional baseado em avaliação formativa proposta por Pimentel (2006).

3.2 ARQUITETURA DA FERRAMENTA

Conforme Pimentel (2006) a arquitetura do sistema é modular e cada módulo possui as seguintes atribuições:

- Módulo de comunicação: é a camada de visualização. Intermedia a comunicação entre o sistema e o usuário. Visa apresentar o sistema para uma "linguagem" que seja agradável aos usuários;
- Módulo de acompanhamento: armazena *feedbacks* e analisa o desempenho tanto do aprendiz quanto do professor. Fornecer orientação que conduza ao aprendiz revisões progressivas que melhorem sua atividade e raciocínio, é um fator que torna um ambiente de aprendizagem eficaz.
- Módulo do aprendiz: armazena o histórico de desempenho e o NAC atual do aluno;
- Módulo de avaliação: apresenta exercícios, considerados unidades de avaliação. Os exercícios se relacionam com os conceitos armazenados no módulo de conhecimento;
- Módulo de administração: administradores e professores têm acesso ao gerenciamento da ontologia, alunos e exercícios;
- Módulo de conhecimento: armazena o conteúdo que será abordado no curso sobre o qual será feito o acompanhamento da aprendizagem.

Uma visão dessa arquitetura é apresentada pela Figura 10.

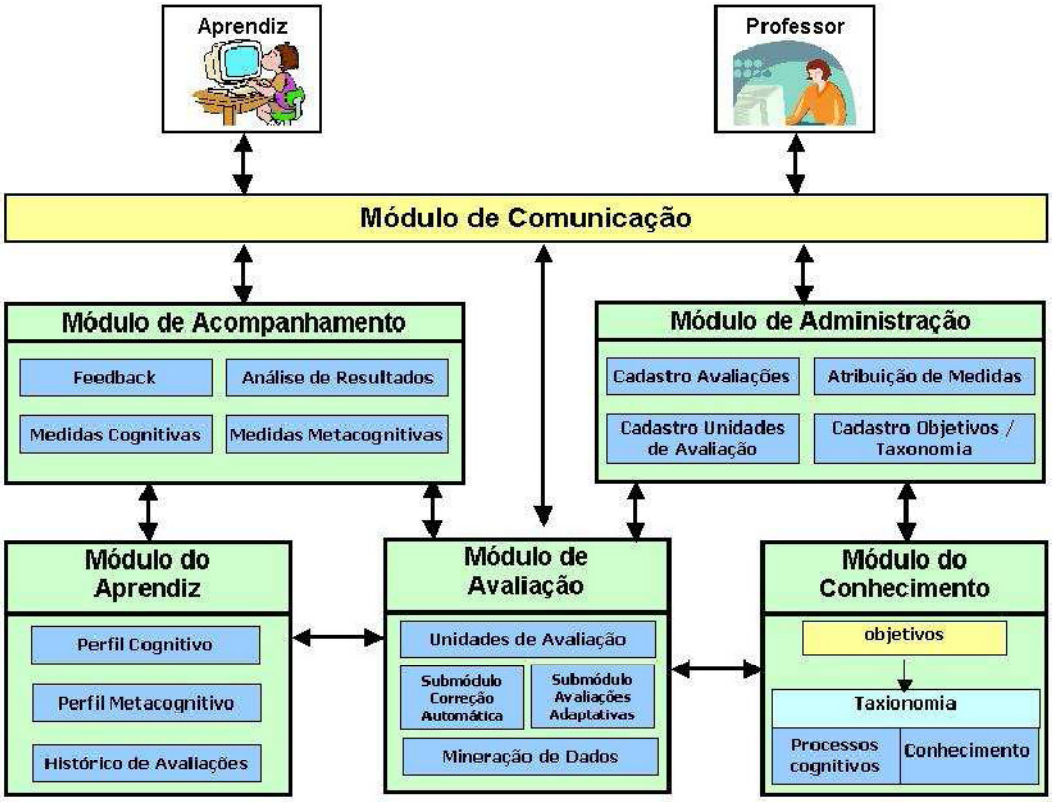

Figura 10. Arquitetura geral do Ambiente Computacional de Avaliação e Acompanhamento (PIMENTEL, 2006).

A implementação desses módulos significa poder diagnosticar, monitorar a aprendizagem, motivação, melhorias da aprendizagem que envolve os aprendizes.

Nesta nova versão do ACAvA, o ACAvA 2.0, este trabalho contribui com a inclusão de, no módulo de conhecimento, conteúdos das disciplinas que envolvem programação, com ênfase na linguagem de programação Java, abordando os principais conceitos que envolvem a sua aprendizagem e criando uma ontologia para os mesmos, aproveitando os recursos oferecidos pela ferramenta em sua versão original.

Já no módulo de avaliação será realizada a inclusão de exercícios de caráter objetivo (testes) que se relacionam com os conceitos e a peculiaridade de fornecer comentários às respostas direcionando ou corrigindo os aprendizes perante suas respostas. Como parte do desenvolvimento do módulo de competências, esta pesquisa realizou o levantamento de um conjunto de exercícios para cada disciplina aqui utilizada.

O módulo de competências é apresentado na próxima seção deste capítulo.

3.3 APLICANDO O MÉTODO DE APRENDIZAGEM POR COMPETÊNCIAS NO ACAvA 2.0

Para os alunos, de uma maneira geral, além de aprender, outro objetivo fundamental é ser aprovado nas disciplinas. Cada disciplina, por sua vez, exige uma série de requisitos, os quais os estudantes devem se mostrar capazes de preencher. Estes requisitos são traduzidos para competências, ou seja, um conjunto de atributos e padrões necessários para desempenhar determinada atividade.

O Método de Aprendizagem por Competências assume que o indivíduo, para desempenhar determinada tarefa com eficácia, necessita possuir um conjunto de Conhecimentos, Habilidades e Atitudes (PARRY, 1996). Neste cenário pode-se exemplificar que, para que o aluno consiga entender plenamente o algoritmo de ordenação "Bubble Sort", ele tem que conhecer, além dos conceitos de ordenação, os conceitos de programação como: declaração de variáveis, estruturas de dados (arrays), operadores lógicos e controle de fluxo (if ou while). Ou seja, para que o aluno tenha a Competência para entender/elaborar um algoritmo de ordenação do tipo Bubble Sort ele precisa conhecer e compreender bem esses conceitos. Conforme Pimentel, Feitosa e Omar (2008) um requerido conjunto de *skills* e habilidades para desempenhar determinada atividade também define um conjunto de conceitos necessários.

3.3.1 Módulo de competências

No âmbito acadêmico, considerando que a disciplina é o ponto de partida para se levantar competências, conceitos e exercícios, este trabalho considera que o processo de aprendizagem de uma disciplina, normalmente, apresenta a seguinte estrutura, conforme a figura 11:

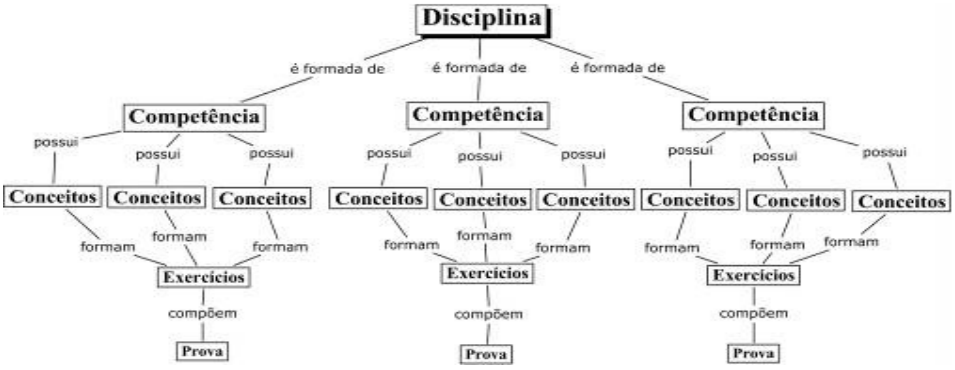

Figura 11. Processo de Aprendizagem de uma Disciplina.

O módulo de competências do ACAvA 2.0, por sua vez, permitirá as seguintes ações, conforme indicam os Casos de Uso da Figura 12:

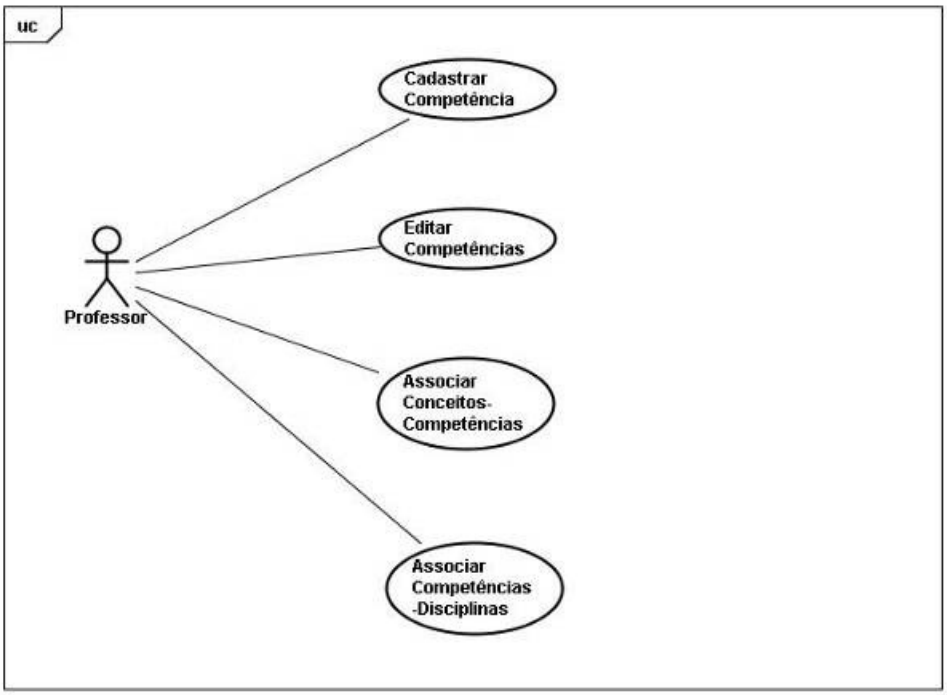

Figura 12. Caso de Uso para o Módulo de Competências do ACAvA.

O módulo de competências foi incluído na pagina inicial junto com os outros módulos, conforme a Figura 13 exibe:

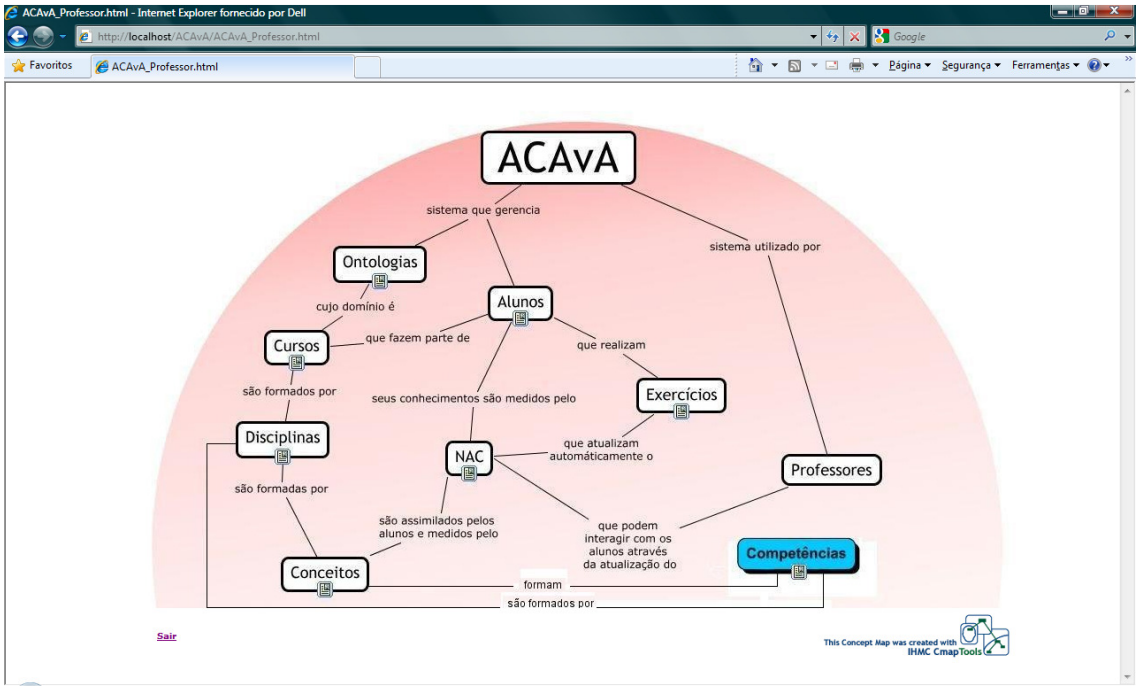

Figura 13. Módulo de Competências destacado na página inicial do ACAvA 2.0.

3.3.2 Levantamento de disciplinas

Tomaremos como suporte para a implementação do módulo de Competências as disciplinas de programação, apoiando-se no currículo da Universidade Presbiteriana Mackenzie (MACKENZIE, 2009), do seu curso de Ciência da Computação.

As disciplinas utilizadas são as seguintes:

• Desenvolvimento Orientado a Objetos;

• Estruturas de Dados;

• Linguagem de Programação I;

• Introdução à Engenharia de Software;

• Linguagem de Programação II;

• Paradigmas de Linguagens;

• Introdução a Técnicas de Desenvolvimento de Algoritmos.

Estas disciplinas foram escolhidas após um estudo sobre o currículo do curso e analisar cada objetivo, ementa, metodologia e conteúdo programático de cada disciplina.

A inclusão de disciplinas no ACAvA 2.0 é feita mediante o cadastramento das mesmas, conforme a Figura 14 apresenta:

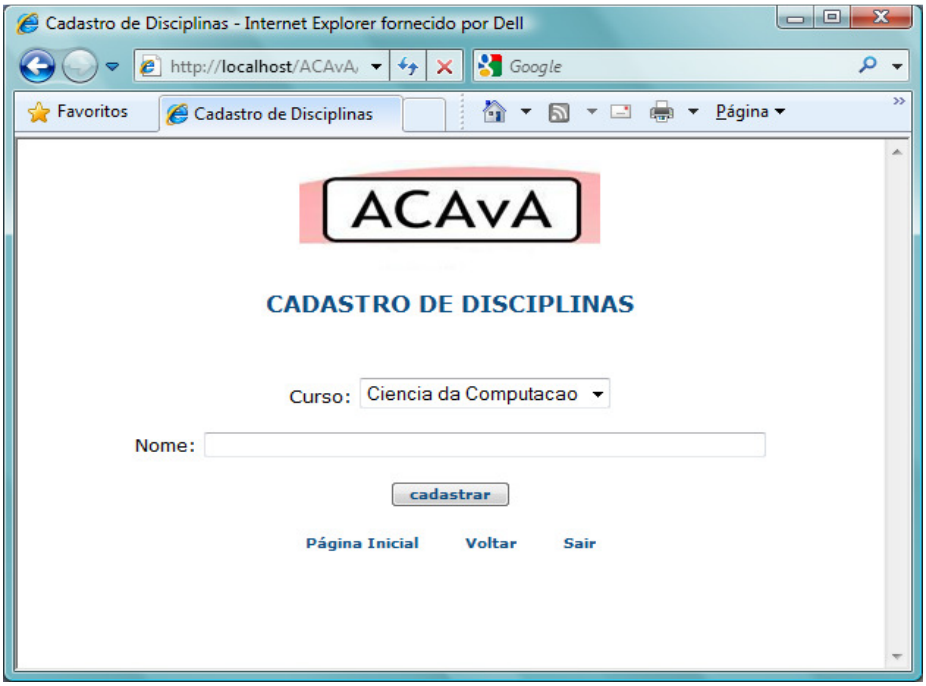

Figura 14. Cadastro de Disciplinas no ACAvA 2.0.

É possível ainda editar as disciplinas e associar os conceitos as disciplinas, conforme as Figuras 15 e 16:

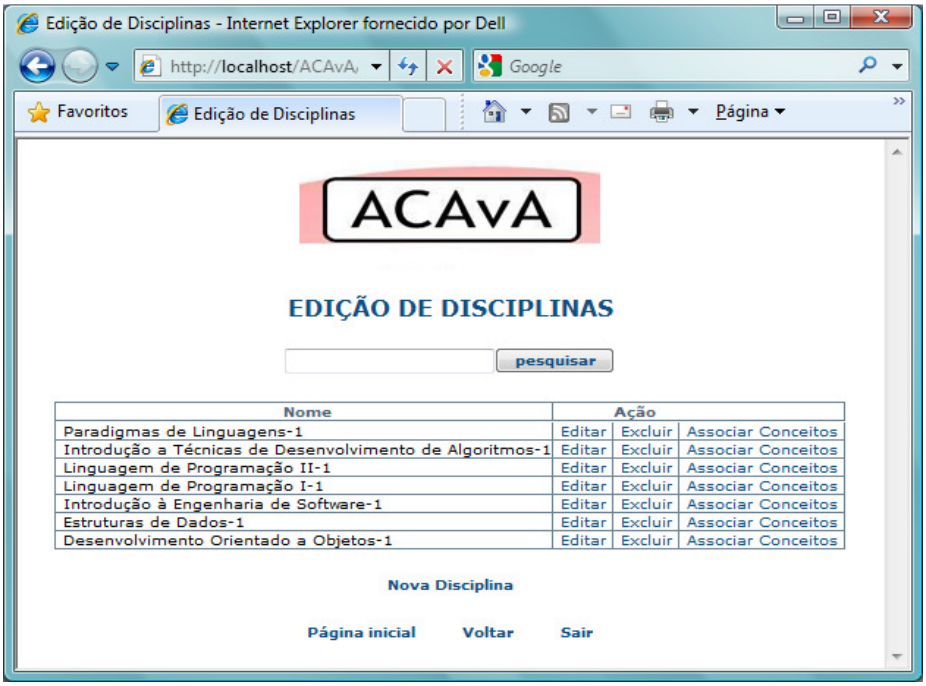

Figura 15. Associando Conceitos à Disciplinas.

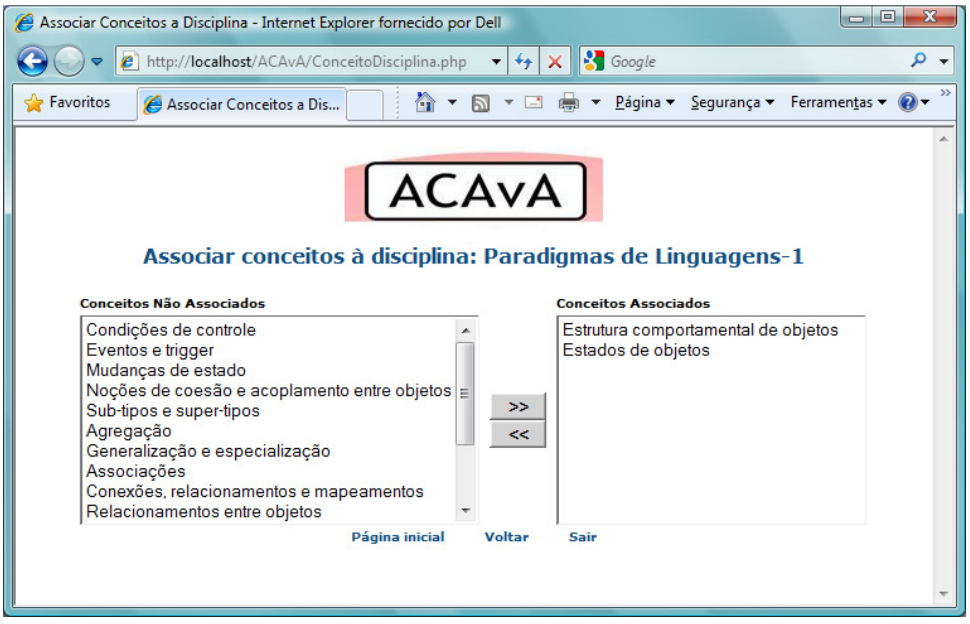

Figura 16. Associando conceitos e disciplinas no ACAvA 2.0.

Após levantar as disciplinas, é necessário definir um conjunto de competências para cada uma, como apresenta destacado em azul a Figura 17:

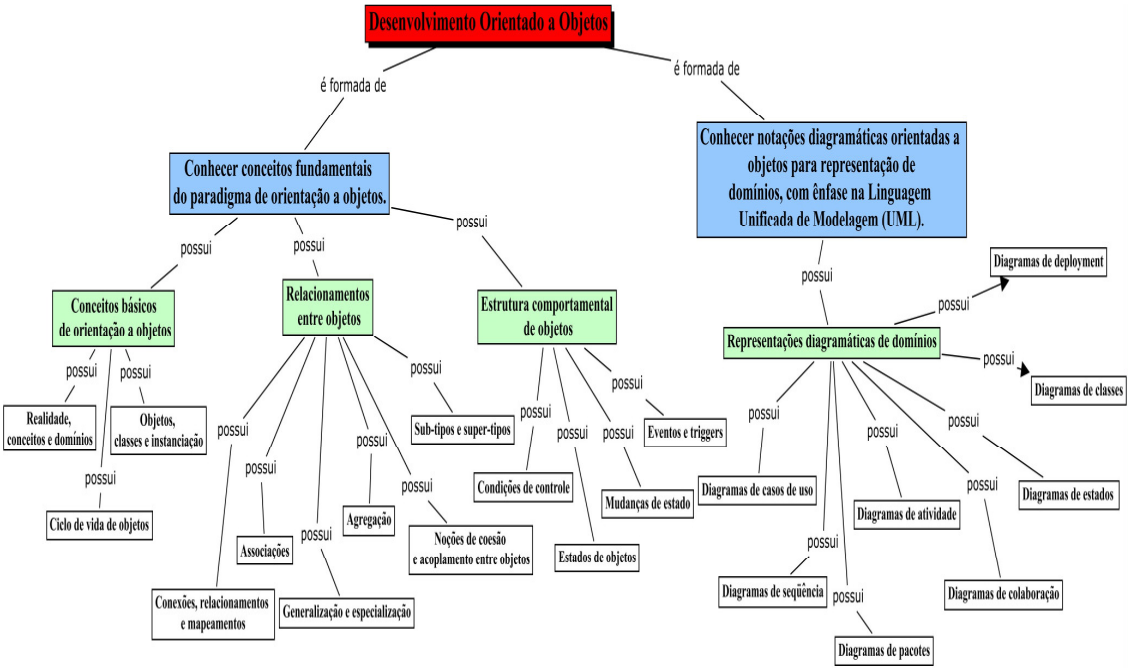

Figura 17. Conjunto de Competências e Conceitos da Disciplina "Desenvolvimento Orientado a Objetos".

Para cada disciplina foi levantado um conjunto de competências consideradas essenciais para a aprovação de um aluno:

- Desenvolvimento Orientado a Objetos
  - Conhecer conceitos fundamentais do paradigma de orientação a objetos;
  - Conhecer notações diagramáticas orientadas a objetos para representação de domínios, com ênfase na Linguagem de Modelagem Universal (UML).
  - Desenvolver programas em Java

- Estruturas de Dados
  - Conhecer conceito de tipo abstrato de dados e sua realização em estruturas de dados clássicas e Analisar estratégias de implementação dessas estruturas e avaliação de eficiência.
  - Desenvolver programas utilizando estruturas de dados

- Linguagem de Programação I
  - Conhecer conceitos fundamentais de linguagens orientadas a objetos, tendo como suporte a linguagem Java e estar apto a construções de projeto para a implementação.
  - Desenvolver programas

- Introdução à Engenharia de Software
  - Aprender conceitos básicos de processos, projeto, verificação, validação, qualidade e evolução de software;
  - Desenvolver um projeto de software desde a especificação de requisitos até a proposta de um plano de testes.

- Linguagem de Programação II

- Ter noções de linguagem de programação Java, reforçando os conceitos de orientação a objetos;
- Possuir noções de programação concorrente e distribuída.

- Paradigmas de Linguagens
  - Conhecer conceitos fundamentais para descrição de linguagens de programação;
  - Conhecer os principais paradigmas de programação e comparar as funcionalidades das linguagens mais comuns em cada paradigma.

- Introdução a Técnicas de Desenvolvimento de Algoritmos
  - Análise de algoritmos clássicos em computação;
  - Utilizar estruturas de dados fundamentais;
  - Provar a corretude de algoritmos simples, bem como avaliar a sua eficiência de forma experimental e analítica.

O módulo de competências do ACAvA 2.0 permite ainda o cadastramento das competências conforme a Figura 18 indica:

Figura 18. Cadastro de Competências no ACAvA 2.0.

A versão 2.0 do ACAvA implementa a funcionalidade que permite associar as competências a disciplina adequada conforme exibe a Figura 19:

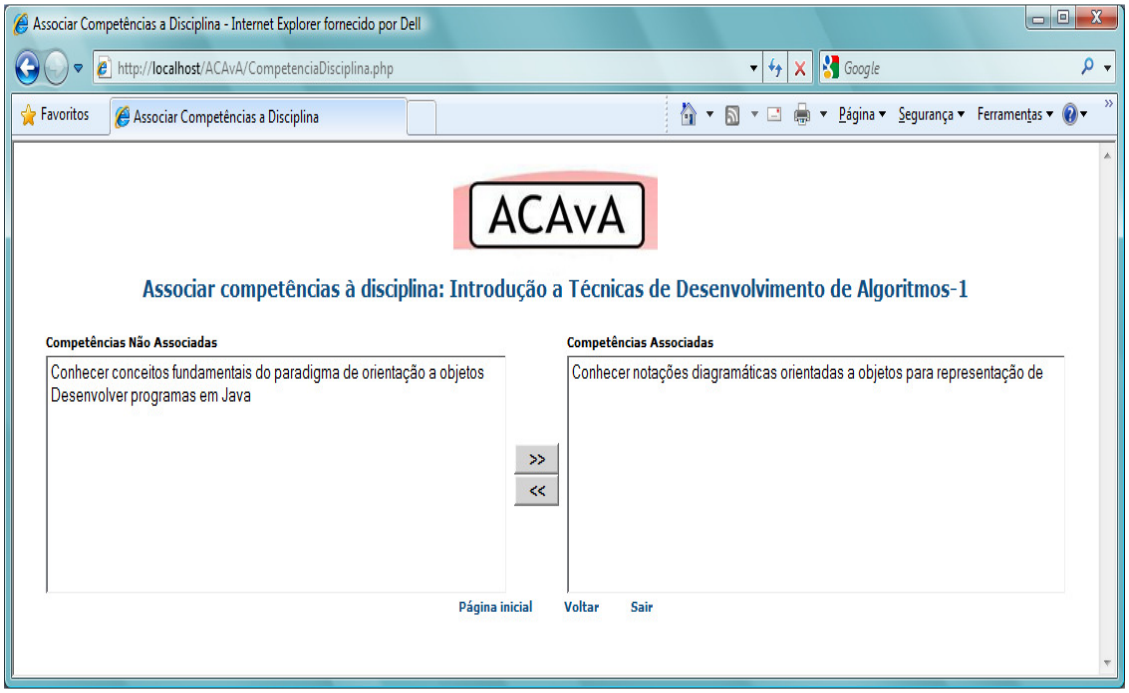

Figura 19. Associando Competências às Disciplinas no ACAvA 2.0.

Para o cadastramento e relacionamento tanto de disciplinas como competências e conceitos, foram criadas no banco de dados Mysql (MYSQL, 2009) as tabelas: Competencia, CompetenciaDisciplina e ConceitoCompetencia, conforme destacado na Figura 20:

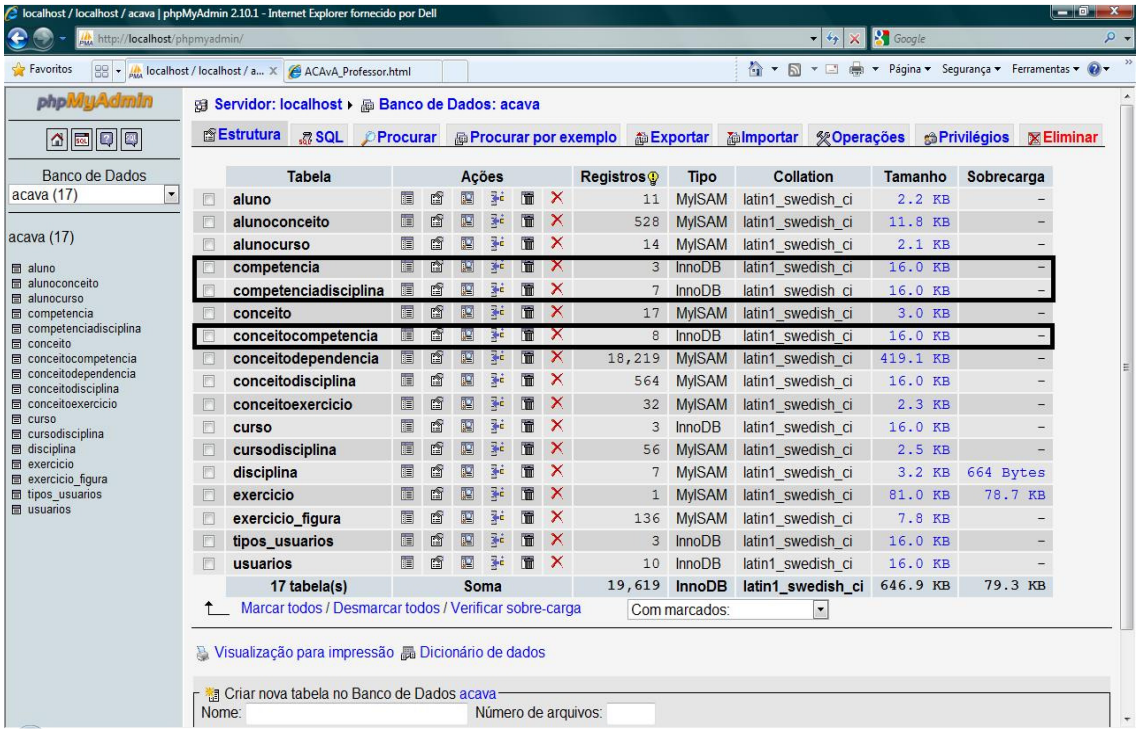

Figura 20. Novas tabelas criadas no Banco de Dados.

Foi utilizado por este trabalho a ferramenta Wamp (WAMP, 2009), um pacote de programas que gerencia automaticamente o Servidor Apache (APACHE, 2009), Php (PHP, 2009), Mysql (MySQL, 2009). Através da funcionalidade PHPMyAdmin que o Wamp disponibiliza é possível gerenciar o banco de dados, criando, alterando e excluindo tabelas.

O Diagrama do Modelo de Dados do módulo de Competências é apresentado pela Figura 21:

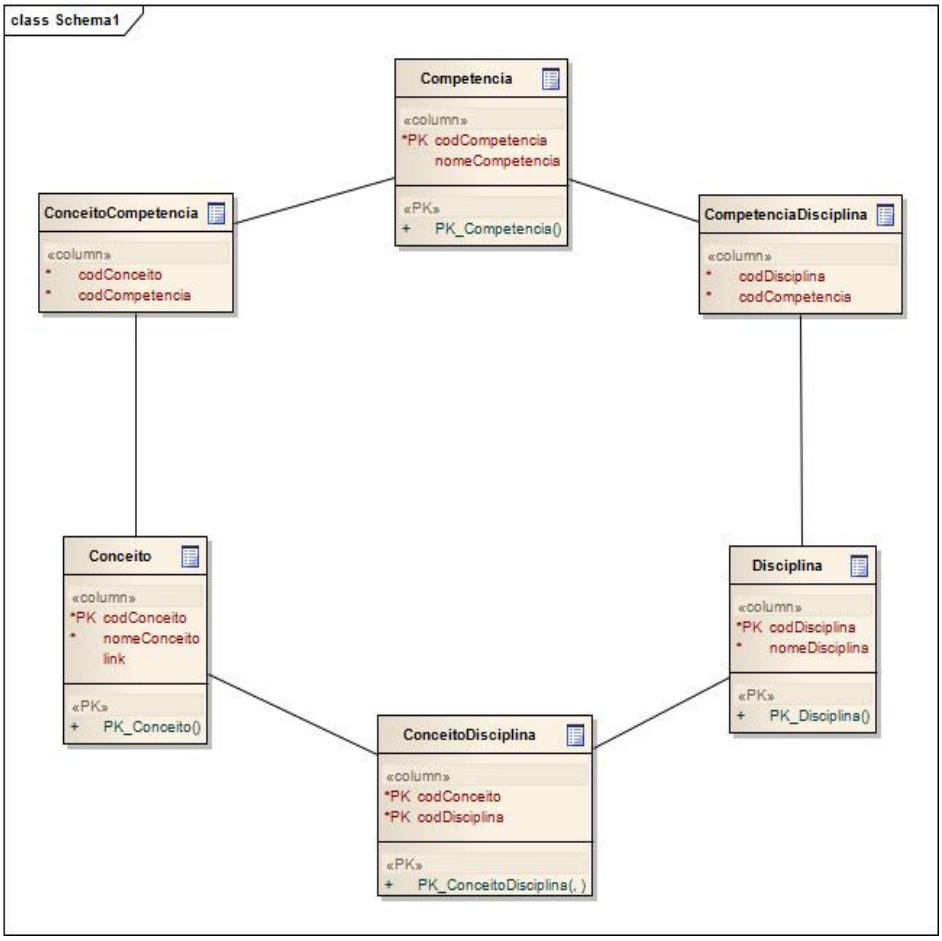

Figura 21. Modelo de Dados das Tabelas utilizadas pelo Módulo de Competências.

3.3.3 Associando conceitos às competências

Após a definição de competências e disciplinas é necessário identificar o conjunto de conceitos cada competência envolverá. A Tabela 5 exibe a associação para a Disciplina Linguagem de Programação II, e sua competência "Possuir Noções de Programação Concorrente e Distribuída".

Tabela 5. Conceitos da Competência "Possuir Noções de Programação Concorrente e Distribuída" e seus respectivos *NACs* fictícios.

Conceitos	NAC
Fundamentos de programação concorrente e distribuída	7
Programação com threads	6
Métodos de sinconização e a declaração synchronized	8
Noções do modelo cliente-servidor e sockets e Servlets e *scripts* JSP	5
Acesso a bancos de dados via JDBC	9
Interfaces gráficas	6
Pacote java.awt e javax.swing	4
Modelo MVC em componentes Swing	8
Applets	8
Modelo de execução de applets e políticas de segurança	9
Construção de applets e páginas HTML	7

A versão 2.0 do ACAvA implementa essa funcionalidade de relacionar conceitos as competências como mostra a Figura 22 e 23:

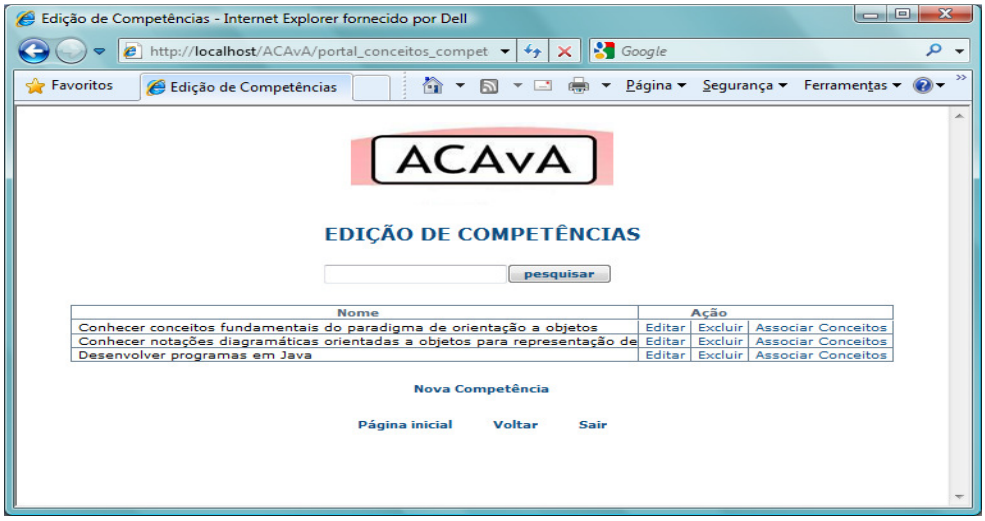

Figura 22. Edição de Competências.

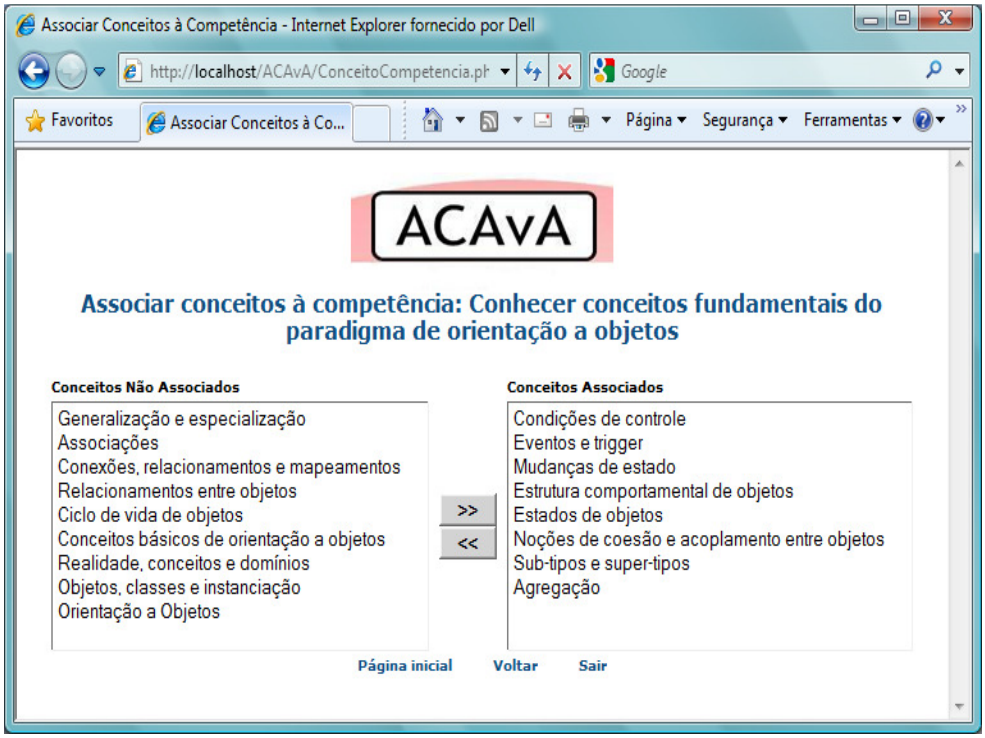

Figura 23. Associação entre conceitos e competênicas.

Portanto, para cada disciplina foi feito um levantamento das competências e conceitos necessários para que os alunos sejam aprovados (Apêndice B).

As competências aqui levantadas são cadastradas no módulo de competências do ACAvA 2.0, e possuem relação com os conceitos, previamente cadastrados no módulo de conhecimento.

3.3.4 Níveis de conhecimento para as competências

No processo de aprendizagem de uma disciplina, é importante associar níveis de conhecimento as competências, ou seja rotular o status dos alunos perante a uma competência. Segundo Pimentel, Feitosa e Omar (2009) os valores das colunas Básico, Intermediário e Avançado correspondem ao nível mínimo exigido para os alunos serem classificados naquele nível.

A disciplina de Estrutura de Dados é um exemplo, conforme Tabela 6:

Tabela 6. Competências da Disciplina Estrutura de Dados.

Competências	Básico	Intemediário	Avançado
Conhecer conceito de tipo abstrato de dado e sua realização em estruturas de dados clássicas.	4	6	9
Implementar estruturas de dados, via ferramentas de orientação a objetos, para aplicações com uma boa estruturação de dados para alcançar requisitos de eficiência.	4	6	9
Analisar estratégias de implementação dessas estruturas e avaliação de eficiência.	4	6	9

3.3.5 Exercícios na composição do módulo de avaliação

Os exercícios estabelecem um cenário no qual o aprendiz se depara com um desafio que ele pode estar apto a fazer ou não, muitas vezes contrariando o seu momento de aprendizagem. Exercícios não são apenas práticos medidores de conhecimento, eles possuem um significado mais profundo, dependendo da maneira como se relacionam. É uma ferramenta que pode se tornar um conector de conceitos, ora estabelecendo relações entre conjuntos de conceitos que envolvem um exercício, ora identificando lacunas conceituais que conectaria um conceito a outro.

É fundamental que nos períodos de aprendizagem de determinados conceitos, os aprendizes se exercitem com atividades que, por trás, tenham uma forte ligação conceitual. Isso, implicitamente, estará forçando o aprendiz a conviver com aqueles conceitos, de modo que ele não poderá encarar outro exercício, sem ter aprendido o conceito anterior.

A ferramenta aqui apresentada se baseia na abordagem de Avaliação Formativa, que se fundamenta em aprendizagens significativas aplicáveis em vários contextos e atualizam-se constantemente, caracterizando uma aprendizagem continuada.

No ACAvA 2.0, os exercícios ficam no módulo de avaliação no formato de provas e na medida em que os exercícios são realizados, ainda quando em caráter objetivo, a ferramenta fornece links previamente cadastrados e atribuídos aos exercícios (relacionados aos conceitos também), redirecionando o aprendiz para um site explicativo caso o aprendiz esteja com alguma dificuldade na resolução da tarefa, mediante o seu erro ou acerto da questão.

A parti daí levantou-se um conjunto de exercícios referente a cada disciplina referente a programação de computadores. O apêndice A é um exemplo de uma prova da disciplina de Linguagem de Programação I incluída no módulo de avaliação do ACAvA 2.0.

O objetivo de realizar avaliações é justamente identificar que conceitos provocam lacunas de aprendizagem nos alunos em determinado conteúdo, permitindo que se estabeleça o *Conceptual Gap*, abordado na próxima seção.

3.3.6 Conceptual Gap

A Competência de uma determinada pessoa poderá ser aferida verificando se a mesma é detentora dos conhecimentos requeridos por determinada Competência. Desta comparação tira-se o *Conceptual Gap*, ou seja, o quão distante está a Competência da pessoa em relação à Competência-modelo. A Figura 24 mostra um exemplo de Competência que envolve quatro conceitos, cada um deles com a mesma relevância, ou seja, peso de 25% cada para se atingir a Competência, de forma que a falta de um deles resultará num *Conceptual Score* de 75% e num *Conceptual Gap* de -25%. Ou seja, há falta de 25% para que a Competência-modelo seja satisfeita.

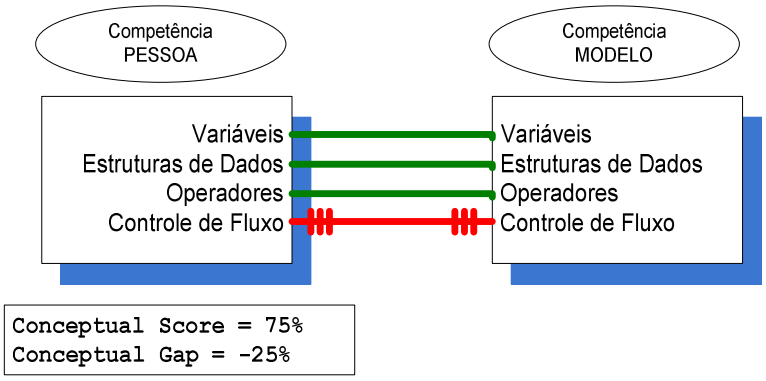

Figura 24. Modelo de Competências e seu respectivo Conceptual Score (FEITOSA, 2009).

Esse modelo é análogo ao modelo do ACAvA, mediante o uso do NAC, incluído no módulo do aprendiz citado anteriormente. Pode-se determinar um NAC ótimo com valor 10, equivalente à nota da Competência-Modelo. Cada competência trata de um conjunto de conceitos que permitirá o aprendiz ter o seu NAC atribuído a eles. Com isso, determina-se que, mediante exercícios que comprovem e ajudem a constituir o NAC do aprendiz, poderemos determinar se ele atingiu a Competência-Modelo para a atividade ou disciplina que deseja desempenhar.

4 MECANISMO DE PIRÂMIDE CONCEITUAL COMO ALTERNATIVA AO PLANEJAMENTO E IDENTIFICADOR DE LACUNAS

Um conteúdo deve ser ensinado separadamente. Esse conteúdo assimilado inicialmente deve ser base de sustentação de um próximo conteúdo.

Seguindo essa linha, este capítulo apresenta o mecanismo denominado Pirâmide Conceitual, que trata de uma proposta para o planejamento da aprendizagem e identificador de lacunas conceituais de um aluno a partir de um conjunto de conceitos, competências e NACs definidos. Uma espécie de norteador, indicando por onde começar e onde se deseja chegar e os possíveis percalços no meio do caminho. Trata-se de uma proposta, estando ainda em fase de desenvolvimento.

Neste cenário, após estabelecer competências e conceitos que envolvem cada competência, é possível traçar planos e definir como os conceitos estão posicionados fisicamente e sua relevância, relacionado a sua importância perante a disciplina que deseja ser aprovado.

Inicialmente, a pirâmide conceitual principal é composta pelos elementos: Curso de Ciência da Computação, Paradigmas de Linguagens, Linguagem de Programação II, Introdução à Engenharia de Software, Introdução à Técnicas de Desenvolvimento de Algoritmos, Linguagem de Programação I, Desenvolvimento Orientado a Objetos e Estrutura de Dados, conforme a Figura 25:

Figura 25. Pirâmide Conceitual principal do curso de Ciência da Computação.

Para cada disciplina, existe uma pirâmide conceitual também, composta inicialmente pelas competências. Ou seja, temos pequenas pirâmides dentro de uma pirâmide principal. É um processo recursivo, como indica a Figura 26:

Figura 26. Pirâmide relacionada as competências da disciplina Desenvolvimento Orientado a Objetos.

O objetivo é de que se faça entender que sempre existe um alvo para se alcançar e que exista uma base de sustentação consistente como alicerce. Nesse caso, o alvo é o topo da pirâmide. Etapa por etapa, tijolo por tijolo, dessa forma construindo uma grande pirâmide conceitual final da disciplina que o aluno está cursando. A ausência de um tijolo poderá indicar uma lacuna conceitual.

Portanto, temos agora uma pirâmide conceitual mais completa, porém não trata-se da pirâmide final ainda, com cada disciplina tendo como suporte suas competências, conforme exibe a Figura 27:

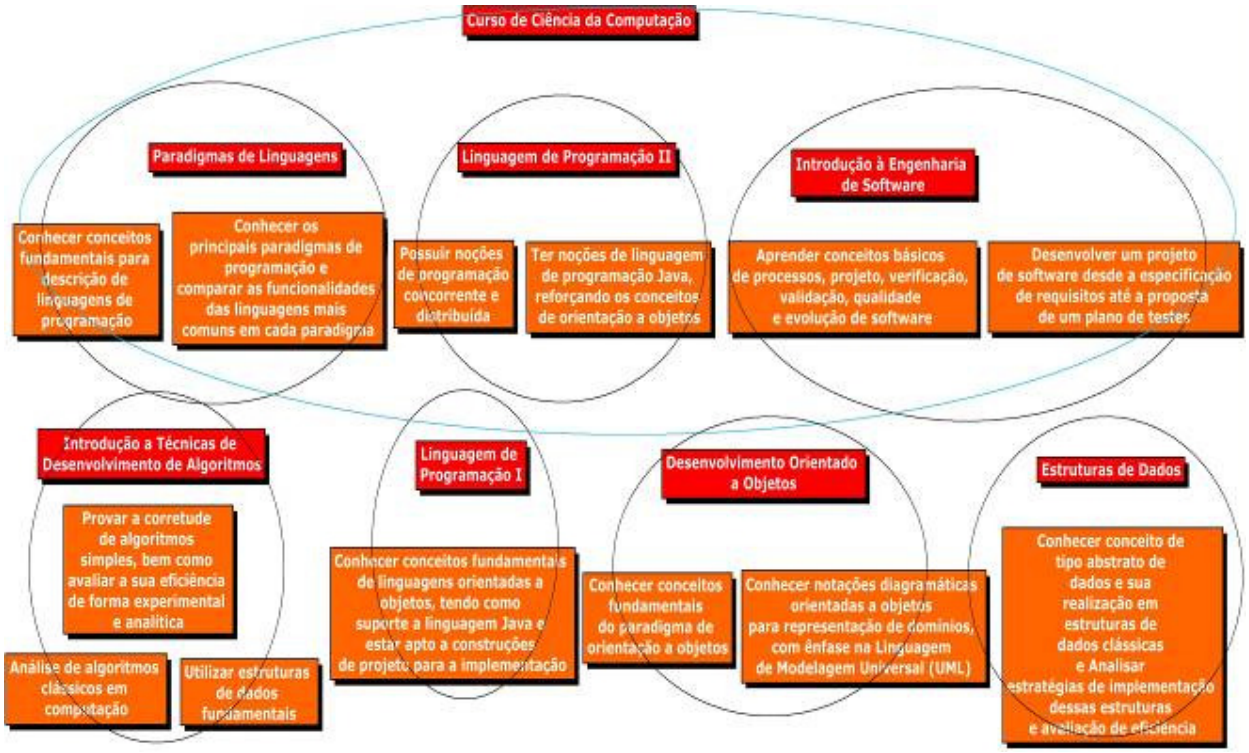

Figura 27. Pirâmide das disciplinas abordadas por esta pesquisa e suas respectivas competências.

A disposição das disciplinas na Pirâmide Conceitual respeita a ordem em que elas são estudadas no Curso de Ciência da Computação do Mackenzie, conforme indica a Tabela 7:

Tabela 7. Semestre em que se estuda determinada disciplina.

Disciplinas	Semestre
Introdução a Técnicas de Desenvolvimento de Algoritmos	1º
Estrutura de Dados	3º
Desenvolvimento Orientado a Objetos	3º
Linguagem de Programação I	3º
Introdução a Engenharia de Software	4º
Linguagem de Programação II	4º
Paradigmas de Linguagens	5º

Após a inclusão das competências na formação da pirâmide conceitual, insere-se nela agora os conceitos que compõem suas competências, como segue, para a disciplina de Desenvolvimento Orientado a Objetos, como segue a Figura 28:

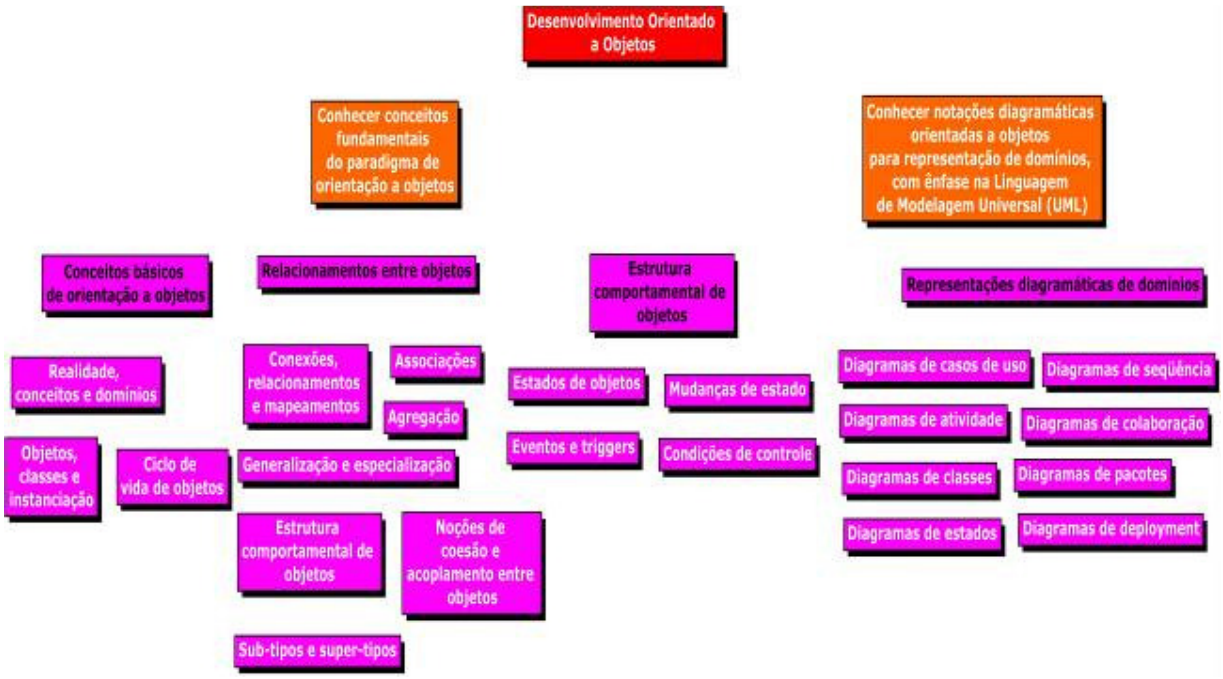

Figura 28. Inclusão dos conceitos na pirâmide relacionada a cada competência.

Nota-se que para que um aluno seja aprovado na disciplina de Desenvolvimento Orientado a Objetos, um conjunto de conceitos deve ser estudado. É fundamental que os exercícios contemplem cada conceito pertencente a cada competência.

Toma-se como exemplo a Competência “Conhecer notações diagramáticas orientadas a objetos para representação de domínios, com ênfase na Linguagem de Modelagem

Unificada (UML)". Essa Competência é formada por nove conceitos, todos com a mesma prioridade. Ou seja, cada conceito (ou cada tijolo desta pirâmide) tem um peso de 11,11%. Já para a Competência "Conhecer conceitos fundamentais do paradigma de orientação a objetos", cada conceito possui um peso de 5, 88%. Se o aluno possui 100% de *Conceptual Score* na Competência "Conhecer conceitos fundamentais do paradigma de orientação a objetos" e um *Conceptual Gap* de 11,11% ou um *Conceptual Score* de 88,89% na Competência "Conhecer notações diagramáticas orientadas a objetos para representação de domínios, com ênfase na Linguagem de Modelagem Unificada (UML)", ou seja, a ausência de um dos conceitos está provocando um *Conceptual Gap* de 11,11%, conclui-se que, a partir da média das duas Competências dessa Disciplina, o aluno possui um *Conceptual Score* referente a Disciplina de 94,44%. Se considerarmos que 75% de aproveitamento em uma disciplina é o suficiente para a aprovação em uma disciplina, neste cenário, mesmo com a ausência de um conceito considerado de mesma relevância dos outros, o aluno seria aprovado.

Porém, a partir daí, esse mecanismo nos permite atacar diretamente onde o aluno está impactado, além de poder certificar o nível do aluno em Básico, Intermediário ou Avançado.

5 ESTUDO DE CASO

Este capítulo apresenta o estudo de caso realizado durante o segundo semestre de 2009, baseado nas atividades de estágio docente obrigatório do programa de Mestrado em Engenharia Elétrica da Universidade Presbiteriana Mackenzie. O estágio docente foi realizado na disciplina de Matemática para Sistemas de Informação II do curso de Sistemas de Informação do Mackenzie.

O objetivo deste trabalho é aplicar o ACAvA 2.0 dentro de uma sala de aula, de modo a contribuir para a organização de conceitos, exercícios e competências e com base em exercícios poder identificar lacunas conceituais pertencentes aos alunos e permitir traçar um plano de aprendizagem a esses alunos.

Embora o ACAvA 2.0 tenha sido apresentado ao longo do trabalho como uma ferramenta para auxílio na aprendizagem de computação, a utilização dele em um contexto diferente foi de extrema valia, na medida em que sua interdisciplinaridade e capacidade de generalização puderam ser testadas.

5.1 DEFINIÇÃO DE CONCEITOS E COMPETÊNCIAS

Segundo a ementa do curso que diz: "*Estudo e compreensão dos conceitos fundamentais do Cálculo Diferencial e Integral que compõem o processo de desenvolvimento de um vocabulário próprio, recursos para notação formal, abstrações úteis e raciocínio lógico, propiciando ao aluno uma visão geral das principais técnicas do cálculo infinitesimal e de sua ligação com a área da computação e informática*" e baseado na estratégia de ensino da disciplina, estabeleceu-se uma etapa fundamental, que se trata da definição dos conceitos e competências necessárias para que os alunos possam obter êxito na disciplina de Matemática para Sistemas de Informação II. As competências que foram previamente cadastradas no módulo de competências do ACAvA 2.0 são as seguintes:

- Compreender o conceito de limites e funções contínuas;
- Aprender o conceito de derivadas e as regras de derivação;
- Reconhecer e resolver problemas que envolvam derivadas;
- Aprender conhecimentos e técnicas que lhe sejam úteis posteriormente para resolução de problemas relativos ao conteúdo da ementa;
- Ser capaz de calcular limites de funções e identificar qual a técnica a ser utilizada;

- Ser capaz de conceituar, calcular e aplicar o conceito de derivada de uma função contínua;
- Ser capaz de resolver problemas de máximos e mínimos através da derivada de funções;
- Ser capaz de construir gráfico de funções definidas por várias sentenças e de outras que requerem os conceitos do Cálculo Diferencial e Integral.

Já no módulo de conhecimento do ACAvA 2.0, foram cadastrados os conceitos que formam o Cálculo Diferencial e Integral:

Tabela 8. Tabela de conceitos da disciplina de Matemática par Sistemas de Informação II.

CONCEITOS
Funções e Limites
Revisão dos conjuntos numéricos com ênfase no conjunto dos números reais
Definição de função e gráfico de funções
Domínio e imagem de uma função
Noção intuitiva de limites; o limite de uma função
Cálculos de limites usando suas leis
Continuidade
Limites no infinito: Assíntotas horizontais e verticais
Definição de derivada
Interpretação geométrica das derivadas
Derivadas das funções elementares a partir da definição
Regras de diferenciação: regras operatórias, derivada do produto e do quociente de funções, derivada da função composta
Derivadas das funções exponenciais e logarítmicas
Derivadas das funções trigonométricas e suas inversas
Aplicações da diferenciação

5.2 ASSOCIAÇÃO ENTRE CONCEITOS E COMPETÊNCIAS

Cada competência estabelecida inicialmente será formada por um conjunto de conceitos também pré-estabelecidos. A seguir este trabalho apresenta essa associação:

- **Competência: Compreender o conceito de limites e funções contínuas.**

- Revisão dos conjuntos numéricos com ênfase no conjunto dos números reais;
- Definição de função e gráfico de funções;
- Domínio e imagem de uma função.

- **Competência: Aprender o conceito de derivadas e as regras de derivação.**
  - Definição de derivada;
  - Interpretação geométrica das derivadas;
  - Derivadas das funções elementares a partir da definição.
- **Competência: Reconhecer e resolver problemas que envolvam derivadas.**
  - Regras de diferenciação: regras operatórias, derivada do produto e do quociente de funções, derivada da função composta;
  - Derivadas das funções exponenciais e logarítmicas;
  - Derivadas das funções trigonométricas e suas inversas.
- **Competência: Aprender conhecimentos e técnicas que lhe sejam úteis posteriormente para resolução de problemas relativos ao conteúdo da ementa.**
  - Revisão dos conjuntos numéricos com ênfase no conjunto dos números reais;
  - Definição de função e gráfico de funções;
  - Domínio e imagem de uma função;
  - Noção intuitiva de limites; o limite de uma função;
  - Cálculos de limites usando suas leis;
  - Continuidade;
  - Limites no infinito; assíntotas horizontais e verticais;
  - Definição de derivada;
  - Interpretação geométrica das derivadas;
  - Derivadas das funções elementares a partir da definição;
  - Regras de diferenciação: regras operatórias, derivada do produto e do quociente de funções, derivada da função composta;
  - Derivadas das funções exponenciais e logarítmicas;
  - Derivadas das funções trigonométricas e suas inversas;
  - Aplicações da diferenciação;
- **Competência: Ser capaz de calcular limites de funções e identificar qual a técnica a ser utilizada.**

- Revisão dos conjuntos numéricos com ênfase no conjunto dos números reais;
- Definição de função e gráfico de funções;
- Domínio e imagem de uma função;
- Noção intuitiva de limites; o limite de uma função.
- Cálculos de limites usando suas leis;
- Continuidade;
- Limites no infinito; assíntotas horizontais e verticais.

- **Competência: Ser capaz de conceituar, calcular e aplicar o conceito de derivada de uma função contínua.**
  - Definição de derivada;
  - Interpretação geométrica das derivadas;
  - Derivadas das funções elementares a partir da definição;
  - Regras de diferenciação: regras operatórias, derivada do produto e do quociente de funções, derivada da função composta;
  - Derivadas das funções exponenciais e logarítmicas;
  - Derivadas das funções trigonométricas e suas inversas;
  - Aplicações da diferenciação.
- **Competência: Ser capaz de resolver problemas de máximos e mínimos através da derivada de funções.**
  - Revisão dos conjuntos numéricos com ênfase no conjunto dos números reais;
  - Definição de função e gráfico de funções;
  - Domínio e imagem de uma função;
  - Noção intuitiva de limites; o limite de uma função;
  - Cálculos de limites usando suas leis;
  - Continuidade;
  - Limites no infinito; assíntotas horizontais e verticais;
  - Definição de derivada;
  - Interpretação geométrica das derivadas;
  - Derivadas das funções elementares a partir da definição;
  - Regras de diferenciação: regras operatórias, derivada do produto e do quociente de funções, derivada da função composta.
  - Derivadas das funções exponenciais e logarítmicas;

- Derivadas das funções trigonométricas e suas inversas;
- Aplicações da diferenciação;

- **Competência: Ser capaz de construir gráfico de funções definidas por várias sentenças e de outras que requerem os conceitos do Cálculo Diferencial e Integral.**
  - Revisão dos conjuntos numéricos com ênfase no conjunto dos números reais;
  - Definição de função e gráfico de funções;
  - Domínio e imagem de uma função;
  - Noção intuitiva de limites; o limite de uma função;
  - Cálculos de limites usando suas leis;
  - Continuidade;
  - Limites no infinito; assíntotas horizontais e verticais;
  - Definição de derivada;
  - Interpretação geométrica das derivadas;
  - Derivadas das funções elementares a partir da definição;
  - Regras de diferenciação: regras operatórias, derivada do produto e do quociente de funções, derivada da função composta;
  - Derivadas das funções exponenciais e logarítmicas;
  - Derivadas das funções trigonométricas e suas inversas;
  - Aplicações da diferenciação.

5.3 APLICAÇÃO DE EXERCÍCIOS E CONCLUSÃO

Ao submeter os alunos à avaliações constituídas por exercícios envolvendo os conceitos citados acima como o cálculo de limites e funções derivadas em sala de aula, podemos concluir que: A maioria dos alunos buscou conhecimentos anteriores como ferramenta, racionalização de denominadores, propriedades de potenciação, simplificação de frações, fatoração, propriedades logarítmas, porém constatamos que em muitas situações tais ferramentas não foram empregadas corretamente, os conceitos de derivação, e continuidade não estavam totalmente construídos, gerando um baixo *conceptual score* e um alto *conceptual gap*.

As análises mostraram que alguns dos obstáculos de alguns estudantes para compreender limites e derivadas estão relacionados com os conceitos de continuidade, e o uso das regras de derivação, pois mobilizaram apenas parcialmente os conhecimentos. Podemos concluir que, para alguns estudantes, que o uso das regras de derivação não traz dificuldades, mas percebemos variados erros nos procedimentos de cálculo e nas interpretações geométricas.

De uma forma geral, pudemos verificar que a maioria dos alunos apresentou dificuldade para solucionar questões em que a simples visualização de um gráfico resolveria, o que o impede de atingir a competência: ”*Ser capaz de construir gráfico de funções e interpretá-las*”.

A partir dessas informações e com a medida do Conceptual Gap do aprendiz é possível desenvolver um plano de aprendizagem para cada aluno. A pesquisa foi testada neste cenário também com vistas a generalizar a ferramenta em direção à interdisciplinaridade, que muitas vezes é responsável pelas dificuldades dos estudantes.

6 RESULTADOS E CONCLUSÕES

A construção deste ambiente, baseado nas pesquisas citadas nos capítulos anteriores resultou em uma ferramenta com algumas particularidades relevantes. Este trabalho contribui com a inclusão de links tutoriais na inclusão de novos conceitos no módulo de conhecimento. Esse item faz a ferramenta ter uma maior interação com o aluno, na medida em que um aluno, ao responder uma questão do módulo de avaliação, receberá esse link como um suporte a sua aprendizagem, já que os exercícios são compostos e relacionados a diversos conceitos.

Os exercícios incluídos no módulo de avaliação também são frutos desta pesquisa. Para enriquecer o módulo de avaliação, no cenário deste trabalho, levantou-se um conjunto de questões pertinentes a cada disciplina de programação abordada aqui. Foi-se buscar na literatura e na internet exercícios usados tanto em certificações, como em cursos superiores de computação.

Contudo, a principal contribuição deste trabalho trata-se da construção do novo módulo de competências, baseado no método de aprendizagem por competências. Definiram-se aqui como objetivo, as disciplinas do curso de Ciência da Computação da Universidade Presbiteriana Mackenzie. Para tanto, foi feito a análise da sua grade curricular e ementa do curso. A partir daí, estabeleceu-se um conjunto de competências necessárias para que um aluno seja aprovado em cada disciplina.

Cada competência traz consigo um conjunto de conceitos, citados anteriormente. Buscou-se competências e conceitos tanto na análise de grades curriculares de diversos cursos com em sites instrucionais. As competências definidas no capítulo 3 foram incluídas no módulo de competências construído nessa versão do ACAvA 2.0. Esse módulo traz consigo a implementação de uma funcionalidade para relacionar disciplinas e competências e competências e conceitos. Funcionalidade similar a de relacionamento entre conceitos do módulo de conhecimento. Além disso, o módulo permite a inclusão e edição de competências.

Já no capítulo 6, estudou-se o mecanismo de Pirâmide Conceitual, estabelecido por esta pesquisa. A Pirâmide Conceitual foi construída no intuito de definirmos um objetivo a se alcançar e que caminhos nos levam até ele, podendo identificar lacunas no meio do caminho. São lacunas conceituais, quais conceitos impactam em determinada aprendizagem, e que relevância esses conceitos possuem em determinado cenário.

Portanto, concluí-se que, considerando que a aprendizagem fundamenta-se na construção de novos conhecimentos e compreensões baseados no que o aprendiz já conhece, os professores precisam estar atentos às lacunas de conceitos que os estudantes adquirem ao

longo do tempo. Se os conceitos primários dos estudantes são ignorados, as compreensões que eles desenvolvem podem ser muito diferentes das que os professores pretendem. Determinar um objetivo e saber identificar quais caminhos podem ajudar nessa trajetória é de extrema valia.

Para resolver essa questão foi adotado o método de aprendizagem baseado em competências em que os aprendizes são classificados por seus conhecimentos prévios com base em seus Níveis de Aquisição de Conhecimentos, *NAC*, de cada um dos conceitos. A partir dessas informações e com a medida do *Conceptual Gap* do aprendiz com relação à pirâmide conceitual necessária para a aprendizagem de um novo conceito é possível desenvolver um plano de aprendizagem para esse aprendiz.

O ACAvA 2.0 pôde ser testado no período de Estágio Docente realizado na Disciplina Matemática para Sistemas de Informação II do curso de Sistemas de Informação da Universidade Presbiteriana Mackenzie. O software mostrou-se uma ferramenta adequada para auxiliar no processo de ensino-aprendizagem dentro de uma sala de aula, na medida em que ele permite a organização e a associação de disciplinas, competências, conceitos, exercícios, alunos e seus respectivos NACs, de uma maneira interdisciplinar, não sendo específica para determinado conteúdo.

6.1 TRABALHOS FUTUROS

Como proposta de trabalhos futuros, considera-se o estudo e aplicação de técnicas de Inteligência Artificial, como por exemplo, a utilização da linguagem AIML (2009) no contexto de avaliação. Desenvolver um módulo de avaliação de modo que permita incluir exercícios dissertativos, com a possibilidade de o sistema emitir *feedbacks* mediante as respostas dos alunos. O AIML possui essa particularidade, é uma linguagem baseado em XML que proporciona diálogos como em linguagem natural, através da sua base de conhecimento. Isso permitiria aumentar a interação entre software e usuário e ampliar o universo de exercícios que compõem o módulo de avaliação. O sistema, por sua vez, abandonaria aos poucos o uso de banco de dados, e usaria apenas XML.

REFERÊNCIAS

AIML Specification (2009) - *Artificial Intelligence Markup Language* - Disponível em http://www.alicebot.org/aiml.html - Acesso em 22/08/2009.

APACHE Specification (2009) - *The Apache Software Foundation* - Disponível em http://www.apache.org/ - Acesso em 05/07/2009.

AUSUBEL, David Paul - *Educational Psycology: A cognitive view* - Holt, Rinehart and Winston Inc., NewYork – 1968.

BARROS, Leidiana Nunes de; DELGADO, Karina Valdivia - *Aprendizado de Programação* - WEI l XIV Workshop sobre Educação em Computação, 2006, Campo Grande, MS. Anais do XXVI Congresso da SBC, p. 31-40 – 2006.

CARVALHO, Aparecido Fabiano Pinatti de et al. - *Planejamento de Ações de Aprendizagem Apoiado por Conhecimento de Senso Comum – Um Exemplo Prático* - Laboratório de Interação Avançada (LIA) – Departamento de Computação – Universidade Federal de São Carlos, São Carlos, SP, Brasil – 2007.

CASTRO, Thais Helena Chaves de et al - *Utilizando Programação Funcional em Disciplinas Introdutórias de Computação* - XXII Congresso da Sociedade Brasileira de Computação / X Workshop sobre Educação em Computação - Florianópolis - 2002.

CMAPTOOLS Specification (2010). Disponível em http://cmap.ihmc.us/conceptmap.html - Acesso em 10/01/2009.

ECLIPSE Documentation (2010). Disponível em http://www.eclipse.org/ - Acesso em 10/11/2009.

FEITOSA, Marcio. Porto. - *Um Modelo Computacional de Apoio à Tomada de Decisão em Processos de Avaliação de Pessoas por Competências* – Dissertação de Mestrado – São Paulo: Instituto Presbiteriano Mackenzie – 2009.

FUJII, Gilson - *Análise de Estruturas Tridimensionais: Desenvolvimento de uma Ferramenta Computacional Orientada a Objetos* – Dissertação de Mestrado – São Paulo: USP – 1997.

GAVIDIA, Jorge Juan Zavaleta; ANDRADE, Leila Cristina Vasconcelos de - *Sistemas tutores inteligentes* - Trabalho de conclusão da disciplina Inteligência Artificial do Programa de Pós-Graduação da COPPE Sistemas da Universidade Federal do Rio de Janeiro - Rio de Janeiro - 2003.

GCC Compiler (2009). *The GNU Compiler Collection*. Disponível em http://gcc.gnu.org/ - Acesso em 10/11/2009.

GCJ Compiler (2009). *The GNU Compiler for the Java Programming Language*. Disponível em http://gcc.gnu.org/java/ - Acesso em 10/11/2009.

GOMES, Anabela Jesus; MENDES, Antonio Nunes José - *Suporte à aprendizagem da programação com o ambiente SICAS* - Viña del Mar: Actas do V Congresso Ibero-Americano de Informática Educativa – 2000.

GOSLING, James (2009) - Java examples (example source code) Organized by topic – Disponível em http://www.java2s.com/Tutorial/Java/CatalogJava.htm - Acesso em 23/04/2009.

GRUBER Thomas R. - *A Translational Approach to Portable Ontology specifications. Knowledge Acquisition* – 1993.

JAVA PROGRAMMER CERTIFICATION (2003). *Mock Exam Number 3 based on the Objectives for the Sun Java Programmers for the Sun Java Programmers Exam*. Disponível em http://www.jchq.net/mockexams/exam3.htm - Acesso em 20/08/2009.

LOCATELLI, Mariana Brólio - *Ambiente de Apoio à Avaliação Continuada Baseada em Aprendizagem Significativa* - Dissertação de Mestrado – São Paulo: Instituto Presbiteriano Mackenzie – 2007.

MANSINI, Elcie F. Salzano - *Psicopedagogia na escola: buscando condições para aprendizagem significativa* - São Paulo: Unimarco - 1993.

MACKENZIE (2004). *Universidade Presbiteriana Mackenzie, Faculdade de Computação e Informática, curso de Ciência da Computação, currículo do ano de 2004*. Disponível em http://www.mackenzie.br/fileadmin/Graduacao/FCI/cc/curric_cc_2004.pdf - Acesso em 15/10/2009.

MEDEIROS, Alex da Rosa; MARCHIORO, Gilberto Fernandes - *AvaliaNet: Geração Automática de Provas e Exercícios* - Universidade Luterana do Brasil (Ulbra) – Fortaleza: Simpósio Brasileiro de Informática na Educação - 2008.

MOREIRA, Marco Antonio - *Aprendizagem significativa* - Brasília: Editora Universidade de Brasília – 1999.

MOREIRA, Marco Antonio; MASINI, Elcie F. Salzano - *Aprendizagem Significativa: a teoria de David Ausubel* - São Paulo: Editora Centauro - 2001

MYSQL Specification (2010). Disponível em http://www.mysql.com/ - Acesso em 05/07/2009.

NETBEANS Documentation (2009). Disponível em http://netbeans.org/ - Acesso em 10/11/2009.

NETO, Francisco Araújo de Almeida; CASTRO, Thais Helena Chaves de; CASTRO, Alberto Nogueira - *Utilizando o Método Clínico Piagetiano para Acompanhar a Aprendizagem de Programação* - XVII Simpósio Brasileiro de Informática na Educação, 2006, Brasília. Simpósio Brasileiro de Informática na Educação - Brasília: Gráfica e Editora Positiva Ltda - 2006.

NOVAK, Joseph D. - *Learning, Creating and Using Knowleadge: Concept Maps as*

Facilitative Tools in Schools and Coorporations - Cornell University - 1998

NOVELLO, Taisa Carla - *Ontologia, sistemas baseados em conhecimentos e modelos de Banco de Dados* – Rio Grande do Sul: Universidade Federal do Rio Grande do Sul - 2002.

PARRY, Scotty B. (1996). *The quest for competencies: Competency studies can help you make HR decisions, but the results are only as good as the study* – Disponível em http://www.trainingmag.com/msg/search/article_display.jsp?vnu_content_id=1505256&imw=Y – Acesso em 29/03/2009.

PARVEZ, Shahida M.; BLANK, Glenn D. - *A Pedagogical Framework to Integrate Learning Style Into Intelligent Tutoring Systems* - Computer Science and Engineering Department, Lehigh University – 2007.

PERRENOUD, Philippe - *Novas Competências para Ensinar* – Porto Alegre: Editora Artmed – 2000.

PEZZA, André Bernardes - *Um sistema para integração disciplinar e gestão continuada da Aprendizagem* - Dissertação de Mestrado – São Paulo: Universidade Presbiteriana Mackenzie – 2004.

PHP Language (2009). Disponível em http://php.net/index.php. Acesso em 05/07/2009.

PIMENTEL, Edson Pinheiro - *Um modelo para avaliação e acompanhamento contínuo do nível de aquisição de conhecimento do aprendiz* - Tese de Doutorado – São José dos Campos: ITA – 2006.

PIMENTEL, Edson Pinheiro; FEITOSA, Marcio Porto; OMAR, Nizam - *Formative Assessment and Certification in Lifelong Learning with Cognitive and Metacognitive Measurements* - São Paulo, SP, Brasil – 2008.

PIMENTEL, Edson Pinheiro; OMAR, Nizam - *Métricas para o Mapeamento do Conhecimento do Aprendiz em Ambientes Computacionais de Aprendizagem* - XVIII SBIE - Simpósio Brasileiro de Informática na Educação, São Paulo, SP, Brasil – 2007.

PRETO, Tânia Martins; GUEDES, André Luiz Pires; CORDEIRO, Luis Guilherme - *A Utilização de Visualizações como Auxílio ao Ensino na Área de Programação* - Universidade de Tecnologia Federal do Paraná (UTFPR), Curitiba, Paraná, Brasil. SBIE - Simpósio Brasileiro de Informática na Educação - Workshop de Ambientes de Apoio à Aprendizagem de Algoritmos e Programação, 2007, São Paulo – 2007.

PROTEGE (2010). The Protege Project. Disponível em http://protege.stanford.edu/ - Acesso em 20/03/2009.

RATIONAL APPLICATION DEVELOPER (2009). Rational Application Developer for WebSphere Software. Disponível em http://www-01.ibm.com/software/awdtools/developer/application/ - Acesso em 20/11/2009.

RICARTE, Ivan Luiz Marques - *Programação Orientada a Objetos: Uma abordagem com java* - Departamento de Engenharia de Computação e Automação Industrial - Faculdade de

Engenharia Elétrica e de Computação - Universidade Estadual de Campinas – Brasil - 2001.

RUSSEL, Stuart J.; NORVIG, Peter - *Artificial Intelligence: A modern approach* - New Jersey. - P. Hall – 1995.

SANTANCHÈ, André; LAGO, Andréa; DOURADO, Patrícia; FERREIRA, Pedro - *Objetos Digitais Complexos na Educação e os Objetos de Aprendizagem* - XVIII Simpósio Brasileiro de Informática na Educação, São Paulo. Anais do XVIII Simpósio Brasileiro de Informática na Educação - Mini Cursos -2007.

SANTOS, Eduardo Toledo; BARROS, Leliane Nunes de; VALENTE, V. C. P. Nogueira - *Projetando uma Ontologia de Geometria Descritiva* - São Paulo: 15° Simpósio Nacional de Geometria Descritiva e Desenho Técnico - 2001.

SILVA, V. – *Ambiente Cooperativo para aprendizagem significativa com mapas conceituais* - Dissertação de Mestrado - São Paulo: Universidade Presbiteriana Mackenzie - 2004.

SONG, J. S. et al. - *An Intelligent Tutoring System for Introductory C Language Course* - Computers Education - Vol 28 No. 2 – 1997.

SUN Preparation For Java Programmer Language Certification (1995). Disponível em http://java.sun.com/docs/books/tutorial/extra/certification/index.html - Acesso em 15/05/2009.

WAMP. Apache, MySQL, PHP on Windows (2009). Disponível em http://www.wampserver.com/en/ - Acesso em 05/07/2009.

XAVIER, Gláucia Maria Costa et al. - *Estudo dos Fatores que Influenciam a Aprendizagem Introdutória de Programação* - Faculdade de Ciência da Computação - FUOM. Formiga, MG, Brasil. IV ERBASE – IV Escola Regional de Computação Bahia-Sergipe - 2004.

APÊNDICE A – EXEMPLO DE PROVA DA DISCIPLINA DE LINGUAGEM DE PROGRAMAÇÃO I

A seguir encontra-se um conjunto de exercícios utilizados como base para testes do módulo de avaliação do ACAvA 2.0. Estes exercícios foram adaptados do site Java Programmer Language Certification (JAVA PROGRAMMER CERTIFICATION, 2009).

Pergunta 1) Qual das seguintes afirmações são verdadeiras?
1) float f=1/3; 2) int i=1/3; 3) float f=1.01; 4) double d=999d;
Pergunta 2) Qual das seguintes palavras são palavras-chave Java?
1) NULL 2) new 3) instanceOf 4) wend
Pergunta 3) Qual das seguintes afirmações são válidas?
1) System.out.println(1+1); 2) int i=2+'2'; 3) String s="on"+'one'; 4) byte b=255;
Pergunta 4) Qual das seguintes afirmações são verdadeiras?
1) O tamanho dos tipos primitivos é dependente de plataforma 2) O tipo padrão de um literal numérico com componente decimal é um float. 3) Você pode modificar o valor de uma instância da classe Integer com o método setValue
Pergunta 5) Qual das seguintes afirmações são verdadeiras?
1) I / O em Java só pode ser executada usando as classes Listener 2) A classe RandomAccessFile lhe permite mover-se diretamente a qualquer ponto de um arquivo. 3) A criação de uma instância com nome da classe do arquivo cria um arquivo correspondente no sistema operacional somente quando o método Close é chamado. 4) As características de uma instância da File Class, tais como o separador de diretório, depende do atual sistema operacional subjacente
Pergunta 6) Qual das seguintes afirmações são verdadeiras?
1) O operador instanceof pode ser usado para determinar se uma referência é uma instância de uma classe, mas não uma interface. 2) O operador instanceof pode ser usado para determinar se uma referência é uma

instância de uma classe de invólucro especial primitivo
3) O operador instanceof só irá determinar se uma referência é uma instância de uma classe imediatamente superior na hierarquia, mas não mais para cima na cadeia de herança.

Pergunta 7)
Qual das seguintes afirmações são verdadeiras?

1) Uma interface pode conter apenas o método e não variáveis
2) Interfaces não podem ter construtores
3) Uma classe pode estender apenas uma outra classe e implementar uma única interface
4) As interfaces são a abordagem Java para enfrentar a falta de herança múltipla, mas exigem a implementação de classes para criar a funcionalidade das interfaces.

Questão 8)
Qual das seguintes afirmações são válidas?

1) public class MyCalc extends Math
2) Math.max(s);
3) Math.round(9.99,1);
4)Math.mod(4,10);

Pergunta 9)
Qual das seguintes são os métodos da interface Runnable?

1) run
2) start
3) yield
4) stop

Pergunta 10)
Qual das seguintes afirmações são verdadeiras?

1) Um byte pode representar entre -128 a 127
2) Um byte pode representar entre -127 a 128
3) Um byte pode representar entre -256 a 256
4) Um char pode representar entre 2x2-pow 16 2 x2 pow 16-1

Pergunta 11)
O que vai acontecer quando você tenta para compilar e executar o seguinte código:

```
class Base{
   public void Base(){
         System.out.println("Base");
   }
}
public class In extends Base{
   public static void main(String argv[]){
         In i=new In();
   }
}
```

1) Compile time error Base é uma palavra-chave
2) Compilação e sem saída em tempo de execução
3) Saída da Base
4) Runtime error Base não tem nenhum construtor válido

Pergunta 12)
Você tem uma classe pública chamada MyClass com o principal método definido da seguinte forma:

```
public static void main(String parm[]){
   System.out.println(parm[0]);
}
```

Se você tentar compilar a classe e executar o programa como segue
java MyClass Hello
O que vai acontecer?

1) Compile time error, main não é corretamente definido
2) Run time error, main não é corretamente definido
3) Compilação e saída de Java
4) Compilação e saída de hello

Pergunta 13)
Qual das seguintes afirmações são verdadeiras?

1) Se uma classe possui os métodos abstratos deve ser declarada abstrata em si.
2) Todos os métodos em uma classe abstrata deve ser declarada como abstrata
3) Quando aplicado a uma classe, o modificador final significa que não pode ser sub-classificado
4) transient e volatile são modificadores Java

Pergunta 14)
Qual dos seguintes métodos são válidos?

1) public static native void amethod(){}
2) public static void amethod(){}
3) private protected void amethod(){}
4) static native void amethod();

Pergunta 15)
Qual das seguintes afirmações são verdadeiras?

1) Construtores não podem ter um modificador de visibilidade
2) Os construtores podem ser marcadas públicos e protegidos, mas não privados
3) Os construtores só pode ter um tipo primitivo de retorno
4) Os construtores não são herdados

Pergunta 16)
O que vai acontecer quando você tenta para compilar e executar a seguinte classe?

```
class Base{
Base(int i){
   System.out.println("Base");
   }
}
class Severn extends Base{
public static void main(String argv[]){
       Severn s = new Severn();
   }
void Severn(){
        System.out.println("Severn");
   }
}
```

1) compilação e produção da seqüência "Severn" em tempo de execução
2) Compile time error
3) Compilação e sem saída em tempo de execução
4) Compilação e saída da String "Base"

Pergunta 17)
Qual das seguintes afirmações são verdadeiras?

1) Os métodos estáticos não têm acesso à variável implícita chamada presente
2) Um método estático pode ser chamado sem criar uma instância de sua classe
3) Um método static não pode ser sobrecarregado para ser não-estático
4) Um método static não pode ser sobrecarregado

Pergunta 18)
Qual das seguintes irá compilar sem erro?

1)
```
char c='1';
System.out.println(c>>1);
```
2)
```
Integer i=Integer("1");
System.out.println(i>>1);
```
3)
```
int i=1;
System.out.println(i<<<1);
```
4)
```
int i=1;
System.out.println(i<<1);
```

Pergunta 19)
Qual das seguintes são verdadeiras?

1) Um componente pode ter apenas um event listener associado ao mesmo tempo 2) Um event listener pode ser removido de um componente 3) A interface ActionListener tem nenhuma classe correspondente Adapter 4) A transformação de um ouvinte de evento requer um bloco try / catch
Pergunta 20) Qual das seguintes palavras são palavras-chave Java?
1) sizeof 2) main 3) transient 4) volatile

APÊNDICE B – COMPETÊNCIAS E CONCEITOS PARA CADA DISCIPLINA

A seguir, encontra-se a listagem das competências e conceitos para cada disciplina utilizada neste trabalho. Cada disciplina, competência e conceito foi utilizado para cadastramento no ACAvA 2.0.

Desenvolvimento Orientado a Objetos

Competência: Conhecer conceitos fundamentais do paradigma de orientação a objetos.

Conceitos:

- **Conceitos básicos de orientação a objetos;**
  - Realidade, conceitos e domínios;
  - Objetos, classes e instanciação;
  - Ciclo de vida de objetos;
- **Relacionamentos entre objetos;**
  - Conexões, relacionamentos e mapeamentos;
  - Associações;
  - Generalização e especialização;
  - Agregação;
  - Sub-tipos e super-tipos;
  - Noções de coesão e acoplamento entre objetos;
- **Estrutura comportamental de objetos;**
  - Estados de objetos;
  - Mudanças de estado;
  - Eventos e *trigger*;
  - Condições de controle.

Competência: Conhecer notações diagramáticas orientadas a objetos para representação de domínios, com ênfase na Linguagem de Modelagem Universal (UML).

Conceitos:

- **Representações diagramáticas de domínios;**
  - Diagramas de casos de uso;
  - Diagramas de atividade;
  - Diagramas de classes;
  - Diagramas de estados (ou máquina de estados);
  - Diagramas de seqüência;
  - Diagramas de colaboração;
  - Diagramas de pacotes;
  - Diagramas de *deployment*.

Estruturas de Dados

Competência: Conhecer conceito de tipo abstrato de dados e sua realização em estruturas de dados clássicas e Analisar estratégias de implementação dessas estruturas e avaliação de eficiência.

Conceitos:

- **Tipos abstratos de dados;**
  - Conceito de tipo abstrato de dados;
  - Especificação e refinamento;
  - Pré e pós-condições;
- **Estruturas Lineares;**
  - Listas;
    - Especificação do tipo de dado abstrato Lista;
    - Tipos de listas;
    - Padrão de projeto *Iterator* e percurso em listas;
    - Implementação e análise de eficiência;
    - Implementação de listas via STL (Standard Template Library);
    - Aplicações;
  - Pilhas;
    - Especificação do tipo de dado abstrato Pilha;
    - Implementação (vetor/lista) e análise de eficiência;
    - Implementação de pilhas via STL;
    - Aplicações;
  - Filas;
    - Especificação do tipo de dado abstrato Fila;
    - Implementação (vetor/lista) e análise de eficiência;
    - Implementação de filas via STL;
    - Aplicações;
  - Deques;
    - Especificação do tipo de dado abstrato Deque;
    - Implementação (vetor/lista) e análise de eficiência;
    - Implementação de deques via STL;
    - Aplicações;
- **Tabelas de *Hashing*;**
  - Especificação do tipo de dado abstrato Tabela de *Hashing*;
  - Funções de *hashing*;
  - Resolução de colisão via endereçamento aberto;
  - Resolução de colisão via estruturas ligadas;
  - *Bucket hashing* e *hashing* extensível;
  - Implementação e análise de eficiência;
  - Implementação de hashing via STL;
  - Aplicações;
- **Árvores binárias;**
  - Especificação do tipo de dado abstrato Árvore Binária;
  - Percursos em árvores;
  - Padrão de projeto *Visitor* e percurso em árvores;
  - Árvores binárias de busca;
  - Heaps e o algoritmo de ordenação Heapsort;
  - Estratégias de balanceamento (AVL, Splay);
  - Implementação e análise de eficiência;

    - Aplicações;
- **Árvores n-árias;**
    - Especificação do tipo de dado abstrato Árvore n-ária;
    - Pomares(*Orchads)*, florestas e florestas ordenadas;
    - Árvores de busca lexicográfica (*tries*);
    - B-árvores;
    - Implementação e análise de eficiência;
    - Aplicações.

Linguagem de Programação I

Competência: Conhecer conceitos fundamentais de linguagens orientadas a objetos, tendo como suporte a linguagem Java e estar apto a construções de projeto para a implementação.

Conceitos:

- **Teoria elementar de tipos;**
    - Classificação de tipos: fundamentais e compostos;
    - Sub-tipos;
    - Equivalência de tipos;
    - Tipos monomórficos e polimórficos;
    - Promoção e conversão de tipos;
    - Lvalues;
    - *Name spaces*;
- **Estruturação elementar de classes e objetos;**
    - Classes como tipos derivados;
    - Atributos e funções (funções isoladas e funções-membro);
    - Construtores;
    - Objetos e o processo de instanciação;
    - Ponteiros para objetos;
    - Destruidores;
    - Herança Simples;
- **Encapsulamento e acesso;**
    - Controle de acesso: estruturas x classes;
    - Especificadores de acesso: private, protected e public;
    - Declarações friend;
- ***Overloading* de funções e operadores;**
    - Conceito de acúmulo;
    - Acúmulo de funções;
    - Declaração de operadores;
    - Acúmulo de operadores;
    - Mecanismo de acesso na presença de acúmulo;
- **Classes derivadas;**
    - Herança múltipla;
    - Polimorfismo universal por inclusão;
    - Classes-base virtuais;
    - Métodos virtuais e a Tabela de Métodos Virtuais;
    - Métodos virtuais e herança simples;
    - Métodos virtuais e herança múltipla;
    - Classes-base virtuais e métodos virtuais;

    - *Renaming*;
    - *Dispatching*: simples, duplo e múltiplo, RTTI;
- **Tratamento de exceções;**
    - Mecanismos de exceções em Java;
    - Regiões críticas e blocos try-catch;
    - Tratamentos múltiplos e ordem de manipulação;
    - Lançamento de exceções e declarações throw;
    - Tratamento hierárquico de exceções;
- **Streams;**
    - Conceito de stream;
    - Streams de entrada;
    - Streams de saída;
- **Templates;**
    - Tipos parametrizados;
    - Polimorfismo universal paramétrico;
    - Templates de função;
    - Templates de classe;
    - STL (*Standard Template Library*).

Introdução à Engenharia de Software

Competência: Aprender conceitos básicos de processos, projeto, verificação, validação, qualidade e evolução de software.

Conceitos:

- **Projeto de software;**
    - Projeto de Arquitetura;
    - Projeto Estruturado;
    - Projeto orientado a objetos;
    - Projeto de software de tempo real;
    - Projeto com reuso;
    - Projeto de interfaces com usuário;
    - Projeto de sistemas críticos;
- **Verificação e validação;**
    - Planejamento de verificação e validação;
    - Testes de detecção de defeitos;
    - Testes de integração;
    - Testes orientados a objetos;
    - Validação de sistemas críticos;
- **Qualidade de software;**
    - Noções de medição e métricas de software;
    - Qualidade de processo e de produto;
    - Medições de processo;
    - Noções de modelos de maturidade;
- **Evolução de software;**
    - Sistemas legados;
    - Evolução de arquitetura;
    - Reengenharia de software;
    - Reengenharia de dados;

- Gerenciamento de versões;

Competência: Desenvolver um projeto de software desde a especificação de requisitos até a proposta de um plano de testes.

Conceitos:

- **Processo de desenvolvimento de software;**
  - Processo de engenharia de sistemas;
  - Modelos de processo de software;
  - Especificação de software;
  - Projeto e implementação;
  - Validação de software;
  - Evolução de software;
  - Qualidade de software;
- **Gerenciamento de projetos;**
  - Atividades de gerenciamento;
  - Planejamento de projeto;
  - Programação de projeto;
  - Gerenciamento de riscos;
  - Noções de estimativa de custos;
- **Engenharia de requisitos;**
  - Requisitos de software;
  - Documento de requisitos;
  - Estudos de viabilidade;
  - Obtenção e análise de requisitos;
  - Validação e gerenciamento de requisitos;
  - Modelos de sistemas;
    - Modelos de contexto;
    - Modelos de comportamento;
    - Modelos de dados;
    - Modelos de objetos;
  - Noções de prototipação de software.

Linguagem de Programação II

Competência: Ter noções de linguagem de programação Java, reforçando os conceitos de orientação a objetos.

Conceitos:

- **Características de linguagem;**
  - Histórico da linguagem;
  - Vantagens e deficiências da linguagem;
  - Comparação com outras linguagens;
  - Geração de bytecodes e a interpretação via JVM (Java Virtual Machine);
  - Integração com outras linguagens;
- **Sintaxe básica;**
  - Tipos e operadores;
  - Declaração de variáveis;

    - Comandos condicionais(if, swtich-case) e iteração(for,while, do-while);
    - Arrays;
    - Organização básica de classes e programas Java;
- **Tratamento de exceções;**
    - Principais exceções;
    - Detecção e recuperação de exceções via try-catch;
    - Tratamentos com finally;
    - Lançamento de exceções via throw;
    - Sinalização de exceções com throws;
    - Construção de classes de exceção;
- **Estruturação de classes;**
    - Atributos, métodos, construtores e instanciação;
    - Superclasses, sub-classes e o processo de herança em Java;
    - Polimorfismo;
    - Declarações static e final;
    - Modificadores de acesso: public, protected, default e private;
    - Classes abstratas;
- **Interfaces;**
    - Declaração e implementação de interfaces;
    - Extensão de interfaces;
    - Comparação com classes abstratas;
- **Pacotes;**
    - Organização e implementação de pacotes;
    - Acesso a pacotes;
    - Organização dos pacotes básicos da linguagem Java: java.lang, java.util; java.io, java.net, java.awt e javax.swing;
- **Classes básicas e utilitárias;**
    - Pacote java.lang (classes básicas);
    - Pacote java.util (classes utilitárias);
- **Tratamento de E/S;**
    - Modelo de streams;
    - Pacote java.io;
    - Tratamento de arquivos.

Competência: Possuir noções de programação concorrente e distribuída.

Conceitos:

- **Fundamentos de programação concorrente e distribuída;**
    - Programação com threads;
    - Métodos de sinconização e a declaração synchronized;
    - Noções do modelo cliente-servidor e sockets;
    - Pacote java.net;
    - Servlets e *scripts* JSP;
    - Acesso a bancos de dados via JDBC;
- **Interfaces gráficas;**
    - Pacote java.awt;
    - Pacotes javax.swing;
    - Modelo MVC em componentes Swing;
- **Applets;**

- Modelo de execução de applets e políticas de segurança;
- Construção de applets;
- Integração com páginas HTML.

Paradigmas de Linguagens

Competência: Conhecer conceitos fundamentais para descrição de linguagens de programação.

Conceitos:

- **Conceitos Fundamentais em Linguagens de Programação;**
  - Distância cognitiva entre linguagens;
  - Classificação das linguagens de programação em função da distância cognitiva;
  - Conceito de paradigma de programação e a classificação de Ambler (paradigmas operacional, declarativo e demonstrativo);
  - Linguagens multi-paradigma;
  - Descrição de linguagens de programação;
  - Nível sintático: gramáticas e as notações BNF/EBNF;
  - Nível semântico e noções de semânticas operacional, denotacional e axiomática;
  - Noções de implementação de linguagens de programação (análise léxica, análise sintática, análise semântica e geração de código/interpretação).

Competência: Conhecer os principais paradigmas de programação e comparar as funcionalidades das linguagens mais comuns em cada paradigma.

Conceitos:

- **Paradigma Imperativo;**
  - Fundamentos da programação imperativa;
  - Conceito de *statement* e suas variantes;
  - Controle de fluxo em *statements*;
  - Programação com invariantes: pré-condições, pós-condições, invariantes de laços, correções parcial e total;
  - Tipos: representação, estáticos e dinâmicos;
  - Conceito de *binding* e suas variantes;
  - Ativação de procedimentos e funções;
  - Estudo comparativo entre as linguagens Pascal e C;
- **Paradigma Orientado a Objetos;**
  - Fundamentos de programação orientada a objetos: objetos, classes, instanciação, métodos e mensagens;
  - Classes-base, classes derivadas e os mecanismos de inicialização, herança e controle de acesso;
  - Invariância, covariância e contravariância de tipos;
  - Polimorfismo e *late binding*;
  - Estudo comparativo entre as linguagens Smalltalk, C++ e Java;
- **Paradigma Orientado a Aspectos;**

    - Problema de *entanglement* em código;
    - Conceitos fundamentais de programação orientada a aspectos: interesses ortogonais, *pointcuts*, *join points*, *advices* e aspectos;
    - Levantamento de interesses ortogonais, instrumentação de código e o processo de *weaving*;
    - Aspectos estáticos e dinâmicos;
    - Aspectos hierárquicos e orientados a fluxos de execução;
    - Estudo comparativo entre as linguagens AspectC++ e AspectJ;
- **Paradigma Funcional;**
    - Conceitos fundamentais de programação funcional: expressões, funções e tipos;
    - Noções das bases matemáticas da programação funcional (λ-cálculo);
    - Expressões simbólicas e listas;
    - Manipulação de listas e o mecanismo de *lazy evaluation*;
    - Passagem de ambiente e de continuação;
    - Estudo comparativo entre as linguagens ML, Lisp e Scheme;
- **Paradigma de Programação em Lógica;**
    - Conceitos fundamentais de programação em lógica: fatos, termos, regras e consultas;
    - Noções dos mecanismos de resolução, substituição e unificação;
    - Árvores de Busca e mecanismos de cortes/falhas;
    - Estudo comparativo entre as linguagens Gödel e PROLOG;
- **Paradigma Concorrente;**
    - Conceitos fundamentais de programação concorrente: processos, *threads*, regiões críticas, acesso concorrente;
    - Mecanismos de sincronização em linguagens para acesso concorrente: semáforos, monitores e *rendezvous*;
    - Problemas de *starvation* e *deadlock*;
    - Estudo comparativo dos mecanismos de programação concorrente em ADA, Java e Concurrent C++;
- **Outros Paradigmas de Linguagens;**
    - Linguagens transformacionais (JESS,CLIPS);
    - Linguagens relacionais (SQL);
    - Linguagens baseadas em *scripts* (linguagens para *shells,* linguagens para interfaces, linguagens para WEB, linguagens para processamento de texto);
    - Linguagens baseadas em marcação (SGML, HTML, XML);
    - Linguagens baseadas em propagação de restrições (Oz).

Introdução a Técnicas de Desenvolvimento de Algoritmos

Competência: Análise de algoritmos clássicos em computação.

Conceitos:

- **Algoritmos de busca;**
    - Busca sequencial;
    - Vetores;
    - Listas ligadas;
    - Análise;

    - Busca binária e sua análise;
- **Algoritmos de ordenação;**
    - Cota inferior para ordenação baseada em comparação Ω(nlogn);
    - Algoritmos clássicos (em vetores);
    - Seleção;
    - Inserção;
    - Bubblesort;
    - Mergesort;
    - Heapsot;
    - Quicksort;
    - Análise de pior caso para os algoritmos clássicos de ordenação baseados em comparações;
    - Algoritmo Radixsort.

Competência: Utilizar estruturas de dados fundamentais.

Conceitos:

- **Recursividade e divisão-e-conquista;**
    - Algoritmos recursivos x algoritmos iterativos;
    - Exemplos elementares de algoritmos recursivos;
    - Análise de algoritmos recursivos (recorrências básicas);
    - Conversão de algoritmos recursivos para iterativos;
    - Paradigma de divisão-e-conquista;
    - Relação entre divisão-e-conquista e recursividade;
    - Exemplos de aplicação de divisão-e-conquista, com análise;
- ***Backtracking*;**
    - Algoritmo geral de *backtracking*;
    - Aplicação em problemas clássicos: labirinto, jogos de tabuleiro, palavras-cruzadas;
    - Eficiência de algoritmos baseados em *backtracking*;
- **Estruturas de dados básicas;**
    - Pilhas (em vetor);
    - Operações básicas;
    - Aplicações;
    - Filas (em vetor);
    - Operações básicas;
    - Aplicações;
    - Listas ligadas simples;
    - Alocação dinâmica de estruturas;
    - Operações básicas;
    - Aplicações;
- **Busca de padrões em *strings*;**
    - Problema da busca de padrões em strings e aplicações;
    - Algoritmo de força bruta;
    - Algoritmo de Karp-Rabin;
    - Algoritmo de Knuth-Morris-Pratt (KMP).

Competência: Provar a corretude de algoritmos simples, bem como avaliar a sua eficiência de forma experimental e analítica.

Conceitos:

- **Medidas de complexidade de algoritmos;**
  - Medidas experimentais de eficiência de algoritmos;
  - Cotas superiores e a notação O(.);
  - Exemplos elementares de análise O(.);
  - Cotas inferiores e a notação Ω (.);
  - Otimalidade de algoritmos;
- **Noções de corretude de algoritmos;**
  - Pré e pós-condições;
  - Invariantes de *loops*;
  - Exemplos elementares de provas de corretude.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo