Igor Lemos Rodarte Arouca
Thiago Alves Silva

SISTEMA DE APOIO À REALIZAÇÃO DE DIAGNÓSTICOS DA APRENDIZAGEM

Projeto de Formatura apresentado à disciplina PCS 2502 – Laboratório de Projeto de Formatura II, da Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Eletricista.

São Paulo
2005

Igor Lemos Rodarte Arouca
Thiago Alves Silva

SISTEMA DE APOIO À REALIZAÇÃO DE DIAGNÓSTICOS DA APRENDIZAGEM

Projeto de Formatura apresentado à disciplina PCS 2502 – Laboratório de Projeto de Formatura II, da Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Eletricista.

Área de Concentração:
Engenharia de Computação

Orientador:
Profa. Dra. Selma Shin Shimizu Melnikoff
Co-Orientador:
Prof. MSc. Rodrigo Filev Maia

São Paulo
2005

Aos nossos pais e familiares, por terem acreditado em nossa capacidade e por nos apoiar durante todos esses anos.

AGRADECIMENTOS

À Professora Dra. Selma Shin Shimizu Melnikoff por sua paciência em nos ensinar e guiar durante toda a trajetória.

Ao Professor MSc. Rodrigo Filev Maia pela confiança depositada no grupo e apoio nos momentos mais difíceis.

Ao Prof. Dr. Paulo Sérgio Cugnasca pelo apoio irrestrito dado em todos os momentos de nossa trajetória acadêmica.

Ao mestrando Eduardo Bertassi, por sua eterna disposição em nos ajudar e aconselhar.

EPÍGRAFE

"A auto-satisfação é inimiga do estudo. Querem-se realmente aprender alguma coisa, devemos começar por libertar-nos disso. Em relação a nós próprios devemos ser 'insaciáveis na aprendizagem' e em relação aos outros, 'insaciáveis no ensino'".

Mao Tse-Tung

RESUMO

O presente trabalho aborda uma maneira de como é possível aprimorar sistemas de ensino à distância (EAD) através da adoção de mecanismos automáticos para acompanhamento e diagnostico da aprendizagem de estudantes.

De maneira a exemplificar a adoção de tais mecanismos, o trabalho apresenta a construção de um sistema de EAD simplificado e o posterior acoplamento de um módulo de apoio ao diagnóstico de estudantes (MADE) a esse sistema.

O mecanismo de diagnóstico desenvolvido nesse trabalho baseia-se na avaliação do desempenho do estudante através de testes de múltipla escolha, e fornece orientações ao mesmo que visam a maximizar o seu aproveitamento em um dado curso. O foco do trabalho não é apresentar mecanismos complexos de diagnósticos e avaliação, mas sim servir de guia para elaboração de arquiteturas que venham a fazer uso de mecanismos de suporte a sistemas de EAD.

Um resultado importante que pode ser tirado desse trabalho é a percepção de que uma vez definida uma interface padronizada entre um sistema de EAD e um módulo de suporte, tal como um MADE é possível trocar a funcionalidade de suporte (técnica de avaliação, por exemplo) a qualquer momento pela simples substituição desse módulo por outro que siga a mesma interface.

O sistema desenvolvido (EAD + MADE) foi denominado SARDA (Sistema de Apoio à Realização de Diagnósticos da Aprendizagem), e também tem o intuito de servir como modelo para a confecção de sistemas *Web* organizados em três camadas que seguem os conceitos de orientação a objetos através da linguagem Java e têm o foco em manutenção e boas práticas de programação.

ABSTRACT

The current work presents a proposal to enhance Distance Learning Environments by adopting automatic mechanisms for tracking and diagnose students' learning.

In order to exemplify the adding of such mechanisms, the work presents the building of a simple DLE system and the forward combination of a support module to diagnose difficulties in the student learning process.

The diagnostic mechanism presented in this work is based on student performance evaluation done by multiple choice tests, aiming to lead students during the learning process, in order to maximise his/her probability of success. This work doesn't focus on present complex evaluation and diagnosis mechanisms, but to serve as a guide for the elaboration of DLE support system architectures.

An important result that can be acquired from this work is the perception that once the standard interface is established (set up) between a DLE system and the support module it is possible to change the support functionality at any moment by the substitution of this module for another with the same interface.

The system which is developed in this work, named SARDA, is a Learning Diagnostics Aid System, and is divided in two parts: DLE and Learning Diagnostic Evaluation Modules. Such system aims to provide a model for the development of web-based three-tier systems, and follows object oriented concepts through Java programming language. Additionally, SARDA development was focused on diminishing maintenance issues and following best programming practices.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
ASF	Apache Software Foundation
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CRC	Class Responsibility Collaborator
EAD	Ensino à Distância
EAR	Enterprise Archive
EIS	Enterprise Information System
EJB	Enterprise JavaBeans
ERS	Especificação de Requisitos de Software
GOF	Gang of Four
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
IHC	Interface Homem-Computador
J2EE	Java 2 Enterprise Edition
J2SE	Java 2 Standard Edition
JSDK	Java Software Development Kit
JAR	Java Archive
JCP	Java Community Process
JSR	Java Specification Request
JSP	Java Server Pages
JVM	Java Virtual Machine

LMS	Learning Management System
MADE	Módulo de Apoio ao Diagnóstico de Estudantes
MVC	Model View Controller
OOA	Object-Oriented Analysis
OOD	Object-Oriented Design
OOP	Object-Oriented Programming
OSI	Open System Interconnection
RPC	Remote Procedure Call
RMI	Remote Method Invocation
SARDA	Sistema de Apoio à Realização de Diagnósticos da Aprendizagem
SMTP	Simple Mail Transfer Protocol
TLD	Tag Library Descriptor
UML	Unified Modeling Language
URL	Universal Resource Locator
WAR	Web Archive

1 INTRODUÇÃO

Conforme mencionado por Maia (2004 p. 22):

> A Educação à Distância (EAD) é um tema bastante discutido atualmente, tanto no âmbito acadêmico como em empresas e indústrias. Ainda existem diversos pontos de controvérsia, como a validade de uma avaliação ou a eficácia do aprendizado num ambiente onde o aluno não possui contato direto com o professor. Embora estas questões sejam relevantes, este também é um momento para reflexão e melhoria tanto da educação à distância (*on-line*) quanto do presencial, pois se está frente a uma nova realidade, onde se podem quebrar vícios presentes nos métodos tradicionais e aplicar conceitos pedagógicos mais adequados aos nossos dias.
> Uma das discussões que se faz presente no contexto de EAD é como um professor pode acompanhar os estudantes em um curso à distância, dada a pouca ou nenhuma interação presencial. Deve-se considerar inclusive a quantidade de alunos substancialmente maior, comparada aos cursos presenciais, uma vez que as limitações físicas de espaço e cadeiras, dentre outras, existente no ambiente escolar tradicional desaparecem na educação à distância.

A proposta desenvolvida no presente trabalho vem justamente de encontro à discussão levantada no excerto anterior e trata da possibilidade de se criar mecanismos simples que ajudem a suprir, ao menos em parte, a ausência do professor em cursos não presenciais. A idéia é adicionar módulos de suporte a um sistema de EAD de maneira a promover certo grau de automação voltado ao auxílio ao aluno durante seu o aprendizado e interação com o sistema. Com isso pretende-se identificar as dificuldades enfrentadas pelos alunos e aumentar a probabilidade de aprovação dos mesmos através de auxílio adequado.

1.1 Objetivo

O objetivo deste projeto é fornecer um modelo para o aprimoramento de um sistema de EAD através da inclusão de mecanismos que permitam a realização de diagnósticos do desempenho de alunos cadastrados nesse sistema.

De maneira a prover este modelo, o trabalho constitui-se da elaboração de uma plataforma *Web* contendo a infra-estrutura básica tipicamente fornecida em um

ambiente de EAD e na construção e posterior adição de um MADE (Módulo de Apoio ao Diagnóstico de Estudantes) a esse ambiente simplificado.

Além das funcionalidades necessárias para a criação e gerência de cursos on-line, o ambiente de EAD desenvolvido fornece aos professores cadastrados no sistema uma ferramenta para a criação de testes de múltipla escolha a serem disponibilizados aos estudantes. O modelo de diagnóstico de alunos proposto neste trabalho é baseado no desempenho obtido pelos mesmos nestes testes de múltipla escolha elaborados pelo professor do curso no qual estão cadastrados. Sendo assim, não serão consideradas outras formas de análise senão a baseada no acompanhamento do resultado das avaliações. Segundo citado por Maia (2004 p.23):

> [...] há outras variáveis que poderiam ser utilizadas para o acompanhamento do estudante, como o tempo de estudo associado com a nota obtida em um determinado tópico de um curso. Ainda se pode acrescentar a relação entre o desempenho nos pré-requisitos e o tópico de estudo.
> Estima-se que estas formas de acompanhamento indicam que o aluno pode não obter bons resultados em um tópico devido a problemas não solucionados adequadamente durante o estudo dos pré-requisitos. Ou então porque o tempo de estudo está abaixo do considerado adequado pelo professor.

O importante de se enfatizar é que a forma de análise, incluindo os critérios para avaliar o aproveitamento dos estudantes, está encapsulada no MADE na forma de um algoritmo que pode ser trocado sem causar impacto ao sistema de EAD.

Um mecanismo interessante que poderia ser acoplado à plataforma desenvolvida e que não foi desenvolvido no presente trabalho seria a criação de uma interface na qual um professor poderia dinamicamente definir os critérios a serem utilizados para avaliação do desempenho dos estudantes e assim tornar flexível o desenvolvimento dos MADEs.

Um último ponto importante para se ressaltar com relação ao método de avaliação adotado diz respeito ao fato desse método ter por finalidade apontar as deficiências de cada aluno, exigindo para tanto, que os professores tomem certos cuidados na elaboração dos testes de múltipla escolha. Esses cuidados serão detalhados no decorrer do texto.

1.2 Motivação

Nos dias atuais, os sistemas de EAD vêm ganhando cada vez mais espaço, e esse avanço pode ser percebido tanto no ambiente escolar e acadêmico, quanto no meio empresarial.

Há inúmeras razões que sustentam e impulsionam essa tendência, dentre as quais podemos citar o crescente volume de informações, ao qual somos submetidos, aliado a contínua necessidade de aprendizado acelerado de novos conceitos, impostos pela sociedade e vinculados às atividades das pessoas. (ROSEMBERG, 2001 apud Maia, 2004).

Isto ocorre porque as empresas, impulsionadas pela concorrência de mercado, querem cada vez mais manter um staff altamente qualificado e bem treinado, gastando o mínimo possível de recursos para isso. As empresas procuram formas de atualização que permitam que seus funcionários estudem e melhorem seu desempenho, porém gastando poucos recursos e mantendo-os presentes no ambiente de trabalho pelo maior tempo possível. De acordo com estimativas levantadas em Robbins (2000), "calcula-se que mais de 90% de todas as empresas dispõem de algum tipo de programa de treinamento sistemático e que mais de 44 bilhões de dólares por ano sejam gastos com iniciativas de treinamento".

Se de um lado existem empresas buscando formas eficazes e baratas para treinar seus funcionários, do outro, existem escolas buscando formas de melhorar o aprendizado e de atingir um maior número de estudantes. As escolas e universidades, além de tentar reduzir seus custos e superar suas limitações de recursos, procuram por maneiras de melhorar o acompanhamento dos alunos e oferecer uma avaliação mais individualizada.

Tais melhorias no processo de aprendizagem seriam impensáveis no passado. Porém hoje, com o avanço das tecnologias computacionais, já é possível se pensar maneiras de criar ambientes de EAD, que tirem proveito dos recursos tecnológicos disponíveis para suprir dificuldades oriundas da falta de contato direto com professor e para prover um ensino personalizado a cada estudante.

Acredita-se que com as ferramentas de interação existentes atualmente nos ambientes de EAD, tais como salas de bate-papo, fórum de discussões, listas de e-mails, videoconferência, dentre outras, e o acréscimo de técnicas de acompanhamento e avaliação dos alunos baseadas em conceitos pedagógicos, é possível desenvolver sistemas que suportem e propiciem um aprendizado eficiente e acessível a um maior número de pessoas.

1.3 Organização

Este texto tem por finalidade documentar o trabalho desenvolvido para a construção de um sistema com orientação educacional. São abordados conceitos de pedagogia relacionados à aprendizagem e avaliação, para então detalhar-se o projeto do sistema proposto, que tem por base estes conceitos.

O texto está organizado segundo a seguinte estrutura de capítulos:

O capítulo 2 apresenta os conceitos relacionados à área de educação e pedagogia, envolvendo considerações sobre aprendizagem, métodos de avaliação, objetivos educacionais e interpretação de resultados. Esses conceitos são abordados devido à natureza e à proposta do trabalho, que visa o aprimoramento de sistemas de EAD. Porém essa é uma área muito ampla e complexa, e, devido ao escopo deste projeto, as idéias e discussões traçadas nesse capítulo têm como fonte primária de informações os estudos realizados em Maia (2004), cuja dissertação como um todo também serviu de inspiração e motivação para a realização do presente trabalho.

O capítulo 3 apresenta uma discussão geral sobre aplicações de software distribuídas, e de aplicações *Web* em particular. O objetivo desse capítulo é oferecer um panorama sobre as tecnologias e metodologias em voga hoje no mercado para o desenvolvimento de aplicações *Web* robustas e escaláveis e com o foco em manutenção e flexibilidade. Vale mencionar que a discussão realizada, apesar de fornecer alguns conceitos genéricos e abstratos, gira em torno dos recursos disponibilizados pela plataforma J2EE (*Java 2 Enterprise Edition*), a qual foi adotada na elaboração desse trabalho.

O capítulo 4 traz detalhes da metodologia empregada para elaboração desse trabalho como um todo, apresentando uma discussão sobre as fases que constituíram a elaboração do projeto.

O capítulo 5 aborda a especificação de requisitos, a análise e o projeto do sistema proposto, detalhando a arquitetura e apresentando os modelos de software deste sistema.

O capítulo 6 detalha a implementação do projeto, argumenta sobre decisões tomadas, explica aspectos técnicos relevantes e relata algumas dificuldades enfrentadas.

O capítulo 7 aborda a metodologia utilizada na estruturação e execução de testes. É apresentada a descrição dos testes mais relevantes realizados e os objetivos dos mesmos. Por fim, é feita uma discussão e avaliação dos resultados obtidos.

O capítulo 8 apresenta um balanço sobre o projeto, levantando discussões sobre a validade do mesmo, tanto no sentido do aprendizado proporcionado aos seus elaboradores, quanto nos benefícios que podem ser obtidos com a adoção da solução proposta. Também são feitas algumas discussões sobre melhorias que poderiam ser realizadas na implementação atual e indicações sobre possíveis linhas de desenvolvimento e pesquisa que podem emergir dessa proposta inicial.

2 EDUCAÇÃO E AVALIAÇÃO

Como já mencionado, o tema principal abordado no presente trabalho é a utilização de técnicas para acompanhamento e avaliação do desempenho de estudantes em ambientes de EAD, as quais têm por objetivo propiciar uma melhoria na aprendizagem destes estudantes e aumentar a probabilidade de aprovação dos mesmos. Essas técnicas, apesar de implementadas através de mecanismos computacionais (software), têm seu alicerce teórico em conceitos pedagógicos. Sendo assim, faz-se necessária a introdução de alguns conceitos, que permitem entender melhor o que é aprendizagem e como ela ocorre, e quais são os princípios envolvidos no processo de avaliação.

2.1 Conceitos

As discussões e conceitos que são apresentados foram elaborados tendo como base principal a pesquisa desenvolvida por Maia (2004), onde se encontra um breve estudo sobre os trabalhos de autores como Piaget, Vygotsky e Bloom.

2.1.1 Aprendizagem

A discussão apresentada a seguir aborda considerações a cerca do conceito de aprendizagem e os meios a partir dos quais ela se processa, segundo a visão de dois pesquisadores:

- **Jean Piaget (1896-1980)** – pesquisador suíço, licenciado em biologia, que realizou uma série de estudos, principalmente com crianças, sobre as formas com que estas aprendiam os assuntos que lhes eram apresentados (veja um resumo da biografia desse autor no apêndice I).
- **Lev Semynovitch Vygotsky (1896-1934)** – pesquisador e psicólogo russo, Vygotsky foi contemporâneo de Piaget e trabalhou em diversas áreas da literatura e psicologia (veja um resumo da biografia da biografia desse autor no apêndice I).

Ambos os autores fazem parte das correntes interacionista e construtivista[1], porém suas teorias divergem de várias maneiras: Piaget centra todo o desenvolvimento no indivíduo, enquanto Vygotsky leva em consideração o ambiente como um fator preponderante na aprendizagem, ou seja, considera o efeito causado pelo ambiente sócio-histórico no desenvolvimento cognitivo (Palangana, 1998 apud Maia, 2004).

2.1.1.1 Aprendizagem Segundo Piaget

Segundo Piaget, o conhecimento evolui progressivamente por meio de estruturas de raciocínio que substituem umas às outras através de estágios. Isto pode ser evidenciado através da diferença na lógica e forma de pensar de uma criança quando comparada com a lógica de um adulto. Em seu trabalho, Piaget identifica os quatro estágios de evolução mental de uma criança. Cada estágio é um período onde o pensamento e o comportamento infantil é caracterizado por uma forma específica de conhecimento e raciocínio e, portanto, deve haver tipos adequados de avaliação para cada estágio. Os quatro estágios são: sensório-motor, pré-operatório, operatório concreto e operatório formal.

Pode-se dizer que Piaget modificou a teoria pedagógica tradicional que, até então, afirmava que a mente de uma criança é vazia, esperando ser preenchida por conhecimento. Na visão de Piaget, as crianças são as próprias construtoras ativas do conhecimento, constantemente criando e testando suas teorias sobre o mundo.

2.1.1.2 Aprendizagem Segundo Vygotsky

Já segundo Vygotsky, a interação social exerce um papel fundamental no desenvolvimento da cognição. De acordo com Vygotsky (1978)

---

[1] Correntes teóricas empenhadas em explicar como a inteligência humana se desenvolve partindo do princípio de que o desenvolvimento da inteligência é determinado pelas ações mútuas entre o indivíduo e o meio. A idéia é que o homem não nasce inteligente, mas também não é passivo sob a influência do meio, isto é, ele responde aos estímulos externos agindo sobre eles para construir e organizar o seu próprio conhecimento, de forma cada vez mais elaborada.

> Cada função no desenvolvimento cultural da criança aparece duas vezes: primeiro, no nível social, e depois, no nível individual; primeiro, entre pessoas (interpsicológico) e depois, dentro da criança (intrapsicológico). Isto se aplica igualmente para a atenção voluntária, para a memória lógica e para a formação de conceitos. Todas as funções superiores se originam como relações reais entre indivíduos.

Um outro aspecto da teoria de Vygotsky é a idéia de que o potencial para o desenvolvimento cognitivo é limitado para certa duração de tempo a qual ele chama de zona de desenvolvimento próximo (ZDP). Além disso, o pleno desenvolvimento durante a ZPD depende da interação social por inteiro. A gama de habilidades que podem ser desenvolvidas com a orientação de adultos, ou por colaboração do grupo, excede o que pode ser obtido sozinho. Vygotsky levanta a necessidade de um trabalho organizado e coletivo em busca de um objetivo, o que também é defendido por Chickering e Gamson (1987), segundo os quais:

> Learning is enhanced when it is more like a team effort than a solo race. Good learning, like good work, is collaborative and social, not competitive and isolated. Working with others often increases involvement in learning. Sharing one's own ideas and responding to others' reactions improves thinking and deepens understanding

2.1.2 Avaliação

A avaliação é um mecanismo chave no processo de aprendizagem e permite que os professores possam averiguar se os estudantes construíram seu conhecimento. Os professores buscam na avaliação respostas às seguintes perguntas (SANT'ANNA, 1995 apud Maia, 2004):

- Se os objetivos propostos foram alcançados;
- Se o programa proposto foi cumprido;
- Se o tempo foi suficiente;
- Se outros objetivos indiretos foram alcançados.

Começou-se a falar em avaliação aplicada à educação com Tyler (1949), considerado como o pai da avaliação educacional. Tyler encara a avaliação como uma comparação constante entre os resultados (desempenho) dos alunos e objetivos

previamente definidos. Segundo ele, "*A avaliação é, assim, o processo de determinação da extensão com que os objetivos educacionais se realizam*".

A seguir, são apresentadas definições de avaliação segundo outros autores (SANT'ANNA, 1995 apud Maia, 2004):

"A avaliação educativa é um processo complexo, que começa com a formulação de objetivos e requer a elaboração de meios para obter evidências de resultados, interpretação dos resultados para saber em que medida foram os objetivos alcançados, formulação de um juizo de valor".

(Sarabbi, 1971 apud Maia, 2004).

"O crescimento profissional do professor depende de sua habilidade em garantir evidências de avaliação, informações e materiais, a fim de constantemente melhorar seu ensino e a aprendizagem do aluno. Ainda, a avaliação pode servir como meio de controle de qualidade, para assegurar que cada ciclo novo de ensino e aprendizagem alcance resultados tão bons ou melhores que os anteriores".

(Bloom, Hasting, Madaus apud Maia, 2004)

"... avaliação é uma coleta sistemática de dados, por meio da qual se determinam as mudanças de comportamento do aluno e em que medida estas mudanças ocorrem".

(Bloom, 1972 apud Maia, 2004).

"Avaliação significa atribuir um valor a uma dimensão mensurável do comportamento em relação a um padrão de natureza social ou científica".

(Bradfield e Moredock, 1963 apud Maia, 2004).

"É um processo contínuo, sistemático, compreensivo, comparativo, cumulativo, informativo e global que permite avaliar o conhecimento do aluno".

(Juracy C. Marques, 1976 apud Maia, 2004).

As definições acima suscitam muitos pontos importantes a respeito do processo de avaliação. A partir da definição dada por Sarabbi (1971), por exemplo, pode-se perceber que a avaliação envolve muito mais do que a formulação de testes e a posterior comparação entre os resultados obtidos pelos alunos.

É enfatizado que o processo de avaliação deve começar pela identificação e formulação dos objetivos educacionais a serem averiguados no processo de avaliação. Uma vez definidos os objetivos, deve proceder-se a elaboração de mecanismos eficazes através dos quais se possam obter evidências de que os alunos alcançaram esses objetivos. Tais mecanismos podem envolver testes com questões dissertativas ou objetivas, trabalhos escritos, apresentações ou qualquer outra técnica. O importante é que eles possibilitem a avaliação dos objetivos educacionais traçados previamente.

A fase, onde se encontram as maiores deficiências nos processos de avaliação tradicionais, é a de interpretação e análise de resultados. O que normalmente ocorre é que a maioria dos professores utiliza testes, erroneamente, como mecanismos de mensuração e não de investigação. Segundo Maia 2004,

> Para o professor, o aluno ideal é "o nota 10", e os demais são avaliados conforme o desempenho deste.
> [...] A medida, que deveria ser uma forma de se medir erros e acertos baseados em um modelo, a partir do qual fossem extraídas conclusões, tornou-se a medida absoluta, não levando em consideração critérios de análise.

Outra crítica feita no mesmo sentido, é que muitas vezes os professores costumam verificar apenas se o aluno errou ou acertou a questão, quando na verdade deveriam avaliar quais são as deficiências do aluno. Um resultado que não esteja de acordo com a escala comparativa, não deve ser considerado um fracasso, e sim uma oportunidade de aprendizado. Porém, para que tais interpretações e análises sejam possíveis, é necessário que se tenha definido, a priori, objetivos claros, simples e precisos, nos quais a avaliação possa se basear. Dessa maneira, espera-se que o avaliador tenha uma visão clara do que será avaliado e que a atividade avaliadora sirva para analisar o rendimento e mudanças ocorridas no aluno, na metodologia de ensino e no professor. (HOFFMAN, 2000 apud Maia, 2004), (ROMÃO, 1998 apud Maia, 2004), (SANT'ANNA, 1995 apud Maia, 2004).

Há muitos outros pontos interessantes que podem ser levantados a partir das definições acima, porém essa sessão é encerrada apenas destacando alguns termos utilizados por Juracy (1976) em sua definição, que diz, dentre outras coisas, que a avaliação é um processo contínuo, comparativo, cumulativo e global.

2.2 Formas e Modalidades de Avaliação

Conforme idealizado por Bloom (1983) e mencionado em Maia (2004), há três categorias ou modalidades de avaliação que diferem segundo sua finalidade:

- **Diagnóstica:** é uma preparação inicial para a aprendizagem, permitindo a verificação da existência ou não de conhecimentos e habilidades que sejam pré-requisitos para o aprendizado em questão;
- **Formativa:** é uma verificação da existência de dificuldades por parte do aluno durante a aprendizagem, averiguando se o mesmo absorveu determinado conhecimento (deve ser aplicado regularmente).
- **Somativa:** trata-se de um controle para saber se os alunos atingiram os objetivos fixados previamente, classificando os mesmos ao final de um período determinado.

Há uma descrição em Chickering e Gamson (1987) que resume a necessidade dos três tipos de avaliação:

> Knowing what you know and don't know focuses your learning. In getting started, students need help in assessing their existing knowledge and competence. Then, in classes, students need frequent opportunities to perform and receive feedback on their performance. At various points during college, and at its end, students need chances to reflect on what they have learned, what they still need to know, and how they might assess themselves.

2.2.1 Normas e Critérios de Avaliação

Há dois métodos para a análise dos resultados obtidos:

- **Por normalização** – os desempenhos dos alunos são comparados entre si utilizando-se uma norma, sendo a avaliação orientada por um conjunto de regras comuns. Considera-se a existência de um aluno médio e de outros que aprendem mais ou menos, em relação ao primeiro. As atividades de

avaliação devem refletir as diferenças entre os alunos, sendo o grupo a referência. A comparação de resultados pode ser processada ao longo do tempo, tendo como finalidade fundamental classificar, posto que informa da posição (relativa) do indivíduo em relação a um grupo. Nesse tipo de avaliação o critério é externo em relação ao indivíduo que aprende e às condições de aprendizagem, sendo a classificação feita por referência a padrões exteriores a essas condições. Esse tipo de avaliação é útil quando se deseja realizar uma classificação do estudante, independentemente de erros e acertos obtidos, sem o parâmetro do aprendizado ideal. (Maia, 2004).

- **Por critério** – o padrão de referência é um critério e não uma norma, ou seja, é avaliado o conhecimento do aluno em relação a critérios pré-estabelecidos constituídos pelos objetivos educacionais, sem que seja feita, necessariamente, comparação entre alunos. O resultado desse tipo de avaliação deve servir de referência para o aluno para este saber se o seu desempenho está de acordo com o esperado e indicar pontos positivos e aqueles que devem ser aprimorados para atingir os objetivos educacionais. (Zaina, 2002 apud Maia, 2004).

2.2.2 Tipos de Questões

As questões que compõem uma avaliação podem ser divididas basicamente em dois tipos:

- **Dissertativas** – estas questões são constituídas por textos livres, onde os alunos respondem a determinada pergunta com suas palavras, fazendo considerações e comentários a respeito de um tema. Como salientado em Maia (2004),

> Este tipo de questão apresenta uma difícil análise das respostas, uma vez que o aluno possui total liberdade para exprimir suas opiniões, evidenciando a individualidade, com respostas subjetivas. São consideradas boas questões para verificar a compreensão, a análise e habilidades de síntese, mas pouco adequadas para medir o domínio de determinado tópico. Portanto, dado o grau de precisão que se tem neste tipo de resposta, a avaliação geralmente é feita por um ser humano.

- **Objetivas** – este tipo de questão restringe as opções de resposta do aluno em valores pré-definidos, evitando que as respostas possuam conotação subjetiva. Questões são consideradas objetivas quando a opinião ou interpretação do avaliador não influenciam a correção da questão. As questões objetivas podem ser divididas em dois tipos:

    - **Questões de recordação** – neste tipo de questão o aluno é levado a preencher determinados campos ou lacunas com palavras ou símbolos. São questões de fácil concepção e que oferecem dificuldade para adivinhação de respostas. Este tipo de questão é adequado quando se pretende medir capacidades de memorização, mas não avalia o quanto o aprendiz sabe sobre o tópico. As perguntas podem ser feitas diretamente ("Quem inventou a lâmpada elétrica?") ou em forma de preenchimento de lacunas ("A capital do Japão é: _____").

    - **Questões de reconhecimento** – as questões de reconhecimento podem ser divididas em:

        - **Verdadeiro-falso** – são questões em que o aluno lê uma proposição e assinala se esta é verdadeira ou falsa. Estas questões devem ser cuidadosamente elaboradas, de modo a evitarem-se ambigüidades ou dúvidas de interpretação. Sua correção é muito fácil, porém tem a desvantagem de aumentar a possibilidade de respostas ao acaso.

            Exemplo:

            Lisboa é a capital de Portugal.

            ( ) Verdadeiro ( ) Falso

        - **Múltipla escolha** – neste tipo de questão o aluno deve escolher uma ou mais alternativas dentro de uma série. Apresentam grande versatilidade, por permitir que o aluno marque as alternativas corretas, erradas, ou a mais certa ou

mais errada. Além disto, consegue-se verificar o raciocínio e julgamento dos alunos. Em compensação são questões de difícil elaboração, pois se precisa garantir que não haja ambigüidades.

- **Ordenação ou associação** – são questões formadas por listas de palavras, frases ou símbolos que devem ser combinados. Este tipo de questão dificulta a adivinhação de respostas, porém não avalia a compreensão dos alunos, apenas sua capacidade de associação.

  Exemplo: Relacione os estados do Brasil e suas respectivas capitais:

(a) Maranhão	( ) Palmas
(b) Sergipe	( ) São Luís
(c) Tocantins	( ) Porto Alegre
(d) Rio Grande do Sul	( ) Aracaju

- **Interpretação** – são questões nas quais se exige do aluno a capacidade de avaliar qual é a alternativa mais adequada e coerente, dentro das alternativas fornecidas, baseando-se em idéias textuais, mapas, gráficos, tabelas e outros insumos. Tais questões estimulam a inferência, a identificação de explicações e o raciocínio[2].

  Exemplo (SANT'ANNA, 1995 apud Maia, 2004):

  O futuro pode não ser tão incerto como se pensa. Ele pode ser visto, sentido e pensado no presente. Mas exige que a pessoa

[2] No presente trabalho optou-se pelo uso de questões de interpretação, pela sua facilidade de correção (o que facilita a implementação de algoritmos de avaliação e realização de diagnósticos e aprendizagem) e pela sua ampla gama de aptidões medidas. Tal escolha será detalhada no capítulo 5.2 – Módulos pedagógicos.

aprenda a vê-lo como futuro, a senti-lo e percebê-lo como futuro que, inevitavelmente, se tornará presente.

A partir da idéia expressa no texto, concluímos que o processo educacional está exigindo:

( ) uma ação dinâmica;

( ) uma ação planejada;

( ) uma ação de mutações múltiplas;

( ) um processo de adaptação e readaptação;

( ) uma nova filosofia no agir.

2.3 Objetivos Educacionais e Taxionomia Educacional de Bloom

Segundo Bloom (1972), cada avaliação deve possuir um objetivo educacional, o qual se traduz em metas que identifiquem eficazmente o que é esperado do aprendizado de algo, ou seja, no momento que o professor define o conteúdo a ser ensinado, deve ser claro para o aluno identificar quando ele chegou à meta. Portanto, tem-se claro qual é a mudança de comportamento esperada por parte do aluno, e quais são as habilidades que ele possuirá ao final do estudo. Com este propósito, Bloom estabelece a taxionomia dos objetivos educacionais, sendo aqui considerado o domínio cognitivo, o qual apresenta uma hierarquia de comportamentos observáveis, que podem ser utilizadas para nortear o que se espera de um estudante. As categorias do comportamento são:

- **Conhecimento** – refere-se a estruturas específicas e universais, de métodos e processos, padrões ou composições, onde o que é privilegiado é a memória.
- **Compreensão** – trata as capacidades e habilidades de organizar operações e as técnicas para generalizar e tratar com materiais e problemas.
- **Aplicação** – faz uso das abstrações e as utiliza em situações reais, onde princípios e operações são utilizados na compreensão de um fenômeno.

- **Análise** – quarta categoria, é onde ocorrem desdobramentos e hierarquização das idéias.
- **Síntese** – é a combinação dos elementos anteriores, o momento onde o estudante consegue combinar um conjunto de conhecimentos, análise e os demais passos anteriores e forma uma nova estrutura de conhecimento, anteriormente desconhecida ou não trivial.
- **Avaliação** – que são os métodos e materiais utilizados para aferir o aprendizado, segundo critérios pré-estabelecidos. Pode-se dividir a avaliação em duas categorias, como é enfatizado na teoria dos objetivos educacionais em evidências externas, como generalizações de conceitos ou comparação de trabalhos; e as evidências internas, as quais são a clareza, a consistência, e a estrutura do raciocínio.

3 APLICAÇÕES DISTRIBUÍDAS PARA SISTEMAS DE SOFTWARE EMPRESARIAIS

Esse capítulo aborda aspectos conceituais relacionados à tecnologia e às características do sistema proposto. A finalidade é introduzir termos e conceitos que são utilizados nos capítulos subseqüentes, contendo detalhes da especificação, do projeto e da implementação do sistema proposto.

3.1 Visão Geral e Histórica

O surgimento e a evolução do conceito de computação distribuída devem ser considerados e, segundo Jim Farley (1998, p. 40) "*The history of truly distributed computing begins with the first day that someone tapped a mainframe operator on the shoulder and asked, 'Hey, is there any way we can both use that?* ". Da evolução dos sistemas de *timesharing* utilizados para compartilhar os recursos de um mainframe entre diversos terminais burros nas chamadas centrais de computação, para a formação de redes de computadores interconectando diversas estações de trabalho, concebeu-se e expandiu-se o conceito de computação distribuída, o qual está conectado a muitos outros conceitos, como o de aplicações cliente-servidor.

O desenvolvimento de aplicações de software distribuídas e modulares é um tema recorrente em computação e que há bastante tempo desperta o interesse de cientistas da computação, engenheiros de softwares e entusiastas do assunto. Essa área da computação, apesar de não ser recente, sofreu um grande avanço nas últimas décadas, resultando na criação de uma enorme variedade de metodologias, arquiteturas, padrões e soluções de mercado baseadas em seus conceitos.

O conhecimento necessário para o desenvolvimento de aplicações distribuídas alia conceitos de redes de computadores e de desenvolvimento de software. A idéia central desse ramo da computação é segmentar uma aplicação em pedaços de software que se responsabilizem por grupos bem definidos de funcionalidades, trabalhando de maneira colaborativa e comunicando para prover os serviços finais desejados pelos usuários da aplicação. Essa segmentação permite que os módulos, segmentos ou camadas (como são normalmente designados) resultantes possam ser

separados fisicamente, abrindo um enorme leque de possibilidades e arquiteturas que podem ser adotadas.

Uma vantagem que emerge da adoção desse modelo é a possibilidade de utilização de uma infra-estrutura de hardware dedicada para camadas específicas da aplicação, o que pode trazer ganhos de escalabilidade e flexibilidade, dentre outros.

Vale notar, porém, que essa abordagem também introduz inúmeros complicadores a serem superados, como questões de comunicação, sincronismo e segurança, por exemplo.

Um fato decorrente da abordagem de construção em várias camadas inerente às aplicações distribuídas[3] é que estas podem ser vistas de diferentes níveis de visão.

O mais baixo nível de programação de aplicações distribuídas envolve a programação utilizando *sockets* que permitem acessos aos serviços de rede, ao mesmo tempo em que abstraem detalhes da comunicação e fornecem à aplicação uma visão da infra-estrutura de rede baseada em *streams* de dados.

No próximo nível do modelo de camadas, aplicações distribuídas definem e implementam protocolos de aplicação de maneira a padronizar a comunicação entre aplicações clientes e servidoras através da infra-estrutura fornecida pelos *sockets*. Há vários protocolos de aplicações padronizados, tais como HTTP (*Hypertext Transfer Protocol*), SMTP (*Simple Mail Transfer Protocol*), TELNET, dentre outros, e diversos aplicativos que implementam esses protocolos.

---

[3] O conceito de aplicação distribuída adotado no presente texto refere-se a aplicações que rodam usufruindo-se de serviços e protocolos de rede estruturados em camadas segundo o modelo de referência OSI (*Open System Interconnection*) e que permitem que computadores troquem dados via uma rede. Aplicações distribuídas trabalham no topo de tais camadas e também fazem uso dos serviços disponibilizados por sistemas operacionais de maneira a executar tarefas coordenadas através de redes.

A seguir é apresentada uma visão geral da plataforma para desenvolvimento de aplicativos empresariais distribuídos utilizando Java como linguagem de programação, a qual foi adotada na elaboração da solução proposta projeto.

3.2 Java e *Networking*

Java, como plataforma de desenvolvimento, oferece muitas facilidades para programação de aplicativos distribuídos. Os recursos oferecidos pela plataforma encontram-se agrupados em vários pacotes, que descritos nas documentações das APIs (*Application Programming Interfaces*) fornecidas juntamente com os *kits* de desenvolvimento oferecidos pela indústria. A seguir são listados alguns dos principais pacotes e recursos (Deitel, 2003):

- *java.net package* → contém as classes e interfaces que definem os recursos fundamentais para *networking*. Oferece recursos para comunicação baseada em *sockets* ou para comunicação baseada em pacotes;
- *java.rmi* e *org.omg packages* → um nível mais elevado de visão para *networking* é oferecido pelas interfaces e classes contidas nesses pacotes (5 subpacotes para RMI e 7 para CORBA). Os pacotes de RMI habilitam objetos Java rodando em diferentes JVMs (*Java Virtual Machines*), normalmente em diferentes computadores, a se comunicarem através de chamadas remotas de métodos. Essas chamadas remotas fornecem um nível de abstração dando a impressão de que foram invocados métodos de um objeto no mesmo programa, quando na verdade são usados recursos embutidos de *networking* (baseados nos recursos do pacote *java.net*) para invocar métodos de objetos remotos. Os pacotes de CORBA oferecem funcionalidades semelhantes aos de RMI, com a diferença de que RMI só pode ser usado entre objetos Java, enquanto que CORBA pode ser utilizado entre quaisquer duas aplicações que entendam CORBA, incluindo aplicações escritas em outras linguagens de programação.
- *javax.ejb* → fornece as classes e interfaces necessárias para a criação de EJBs, que são objetos distribuídos e gerenciados baseados no padrão RMI.

- *javax.servlet* → esse pacote juntamente com seus subpacotes fornecem os recursos necessários para programação *Web* em Java. São fornecidas classes e interfaces para criação de *Servlets* e JSPs (*JavaServer Pages*), que são componentes *server-side* gerenciados e baseados no padrão CGI (*Common Gateway Interface*) para geração dinâmica de páginas *Web*.

Os pacotes citados nos primeiros dois itens fazem parte do *core* da plataforma Java, denominado J2SE (*Java 2 Standard Edition*). Já os pacotes mencionados nos dois últimos itens, fazem parte do J2EE, que é a edição de java voltada para construção de aplicativos empresariais[4] e que foi utilizada para o desenvolvimento da solução proposta. Note que as APIs da edição J2EE baseiam-se nos recursos oferecidos na J2SE.

3.3 Vantagens de uma arquitetura *web-based*[5]

O desenvolvimento de ambientes de EAD, como o elaborado no presente projeto, deve levar em consideração uma série de requisitos que são tidos como inerentes a esse tipo de ambiente. A seguir são apresentados alguns desses requisitos:

- **Acesso remoto** – a própria natureza e proposta do ensino à distância implicam na necessidade de interação remota entre seus participantes, assim como o acesso remoto a conteúdos educacionais.
- **Interface com o usuário simples** – esse requisito está relacionado à ampla diversidade de indivíduos que interagem em um ambiente de EAD, o que implica na necessidade de uma interface simples e acessível a todos,

---

[4] Aplicativos empresariais, segundo nomenclatura adotada no desenvolvimento com a plataforma Java, referem-se a aplicativos compostos por módulos *Web* (WAR files) e módulos de objetos distribuídos ou EJBs (EJB-JAR files). Os múltiplos módulos EJBs e *Web* são empacotados em um único arquivo que representa a aplicação empresarial em questão (EAR file).

[5] *Web-based* é o termo utilizado quando o meio de acesso principal ao sistema é um navegador (*browser*) e o protocolo de comunicação principal é o HTTP.

mantendo o foco na aprendizagem do conteúdo que se pretende ensinar e não em habilidades computacionais necessárias à interação com o ambiente.

- **Aplicativos clientes de fácil obtenção e instalação** – Os aplicativos clientes para acesso ao ambiente de EAD, devem ser fáceis para se obter e instalar. Esses aplicativos também devem ser flexíveis o suficiente para atender ao amplo espectro de computadores que poderão ser utilizados pelos usuários do sistema de EAD.

Com a finalidade de atender os requisitos citados anteriormente, assim como outros quesitos que serão detalhados nos itens relativos à especificação do projeto, optou-se pela adoção de uma arquitetura *Web* implementada em Java para construção do sistema de EAD, que é parte integrante da solução proposta por esse trabalho.

Nesse tipo de arquitetura, o acesso ao sistema é feito via navegadores *Web* (*browsers*), os quais permitem o acesso remoto, são fáceis de usar e amplamente disponíveis, dispensando a instalação e manutenção de aplicativos adicionais nas máquinas clientes.

A figura 1 ilustra de maneira simplificada uma arquitetura *Web* genérica:

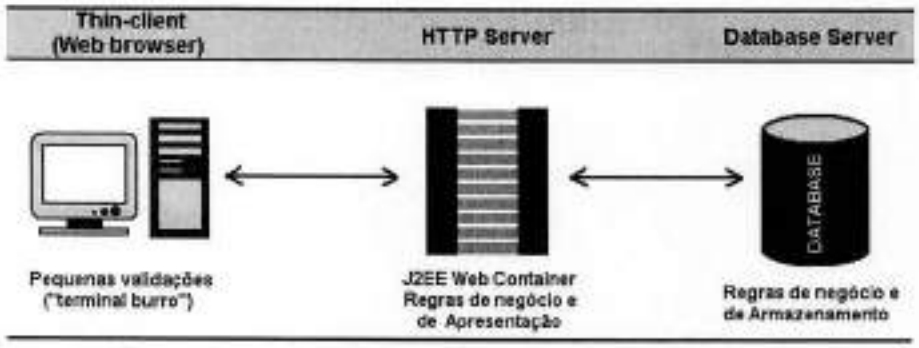


Figura 1 - Modelo *web-centric* com *thin-clients*

A idéia nesse tipo de arquitetura, como se pode observar da figura 1, é ter clientes magros (navegadores web) acessando um servidor que além de conter a lógica de negócio e determinar o fluxo da aplicação, também é o responsável pelo acesso ao banco de dados. A intenção é deixar todo o processamento do lado do servidor.

Apesar da simplicidade e popularidade das aplicações *Web* dispensarem maiores explicações sobre o seu funcionamento, na próxima seção é apresentada uma visão geral dos recursos e boas práticas fornecidos pela plataforma J2EE para construção de sistemas *Web* robustos.

3.4 A Tecnologia *Web* da Plataforma J2EE

Como mencionado anteriormente, o desenvolvimento de aplicações *Web* com a plataforma Java faz uso das classes e interfaces contidas no pacote (e respectivos subpacotes) javax.servlet que é parte integrante da plataforma J2EE.

A construção de aplicações *Web* utilizando Java tem em sua base a tecnologia de *Servlets*, os quais são componentes gerenciados e baseados na tecnologia CGI para geração de conteúdo dinâmico servido via HTTP[6], como arquivos HTML (*Hypertext Markup Language*) e afins. Para o gerenciamento dos *servlets* de uma aplicação *Web* desenvolvida, é necessário utilizar-se um container[7] *Web*, que é o *runtime* de suporte que viabiliza a API de Servlets. O container *Web* é o responsável pelo gerenciamento do ciclo de vida de servlets, ou seja, é ele quem determina quando criar, executar e destruir servlets, além de ser o responsável pela aplicação correta dos mapeamentos de requisições de usuários, restrições de acesso e segurança, dentre outras funcionalidades verticais. Vale ressaltar que o ambiente de *runtime* necessário para execução de uma aplicação *Web* desenvolvida em Java, deve incluir tanto um container *Web* quanto um servidor *Web*, os quais podem estar implementados em um único aplicativo (prática comum), ou podem ser integrados através de conectores desenvolvidos especialmente.

---

[6] A API de servlets fornece uma interface (*Servlet*) e uma classe abstrata (*GenericServlet*) que podem ser implementadas de modo a viabilizar a criação e utilização de servlets que façam uso de outro protocolo que não o HTTP. A implementação de tal interface e classe abstrata para uso do protocolo HTTP é fornecida pela própria API (classe HttpServlet).

[7] O *container Web Jakarta Tomcat* da *Apache Software Foundation* é a implementação oficial de referência da API de Servlets e JSP, cujas especificações atuais, desenvolvidas pelo JCP (Java Community Process), encontram-se nas versões 2.4 (JSR - 154) e 2.0 (JSR - 152), respectivamente, e são implementadas no Jakarta Tomcat versão 5.

A figura 2 ilustra a integração entre um servidor HTTP (Apache) e um container de servlets (*Tomcat*), ambos desenvolvidos pela *Apache Software Foundation* e extremamente populares no mercado[8]. Note a presença dos componentes mod_x, que são conectores que fazem a interface com o servidor Apache.

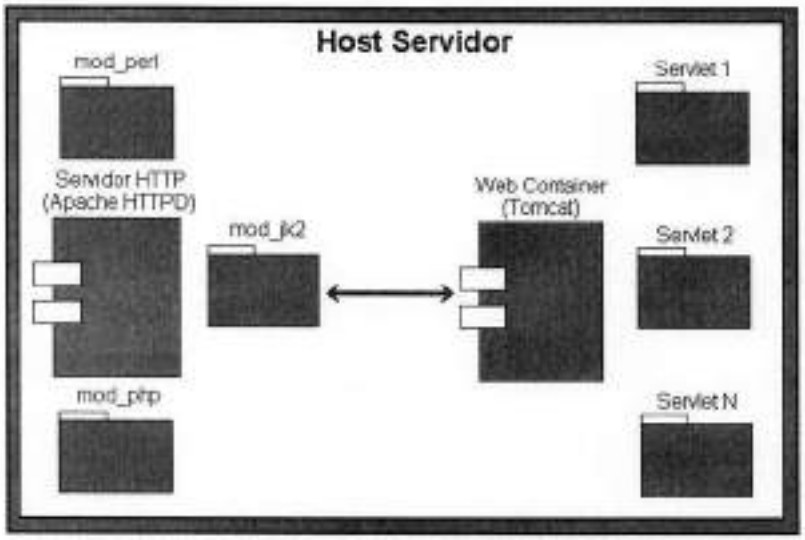


Figura 2 - Integração do Jakarta Tomcat com o Apache HTTPD

Outro componente central da API de Servlets é a tecnologia de JSPs que são recursos para páginas dinâmicas que mesclam código HTML com código Java. JSP, na verdade, é uma opção de mais alto nível para construção de *Servlets*, dada a dificuldade da escrita manual de servlets para gerar conteúdo HTML (páginas JSPs são transformadas em servlets equivalentes que são depois compilados pelo container *Web*).

A API de Servlets também fornece os requisitos para a criação de outros componentes como: *Servlet Filters*, *Custom Tags*, etc. Porém, o intuito dessa e das próximas sessões é apresentar alguns conceitos básicos relativos a essa vasta API e também mostrar como seus componentes podem ser combinados e utilizados de

[8] Refira-se ao apêndice II para maiores informações sobre a estrutura do Tomcat e a integração deste com um servidor HTTP.

maneira colaborativa para formar uma solução robusta, flexível e manutenível. Não é objetivo entrar em detalhes do funcionamento de um *container web* e nem apresentar os aspectos técnicos e de implementação de cada componente da API de Servlets.

3.5 Arquitetura em Camadas e *Design Patterns*

Segundo Inscore e Kassem (2002), a plataforma J2EE é extremamente flexível, de modo que com seus recursos é possível montar soluções para atender desde pequenos projetos com arquiteturas *Web-based* bem simples, até projetos de missão crítica com mais de 4 milhões de transações por dia. Sendo assim, um dos grandes desafios em um projeto J2EE está exatamente em decidir qual a melhor arquitetura, além dos melhores componentes e padrões de projeto adequados ao cenário em questão.

Conforme nota de abertura apresentada no documento de especificação da plataforma J2EE 1.4, sob controle do JCP,

> Enterprises today need to extend their reach, reduce their costs, and lower the response times of their services to customers, employees, and suppliers.
> Typically, applications that provide these services must combine existing enterprise information systems (EISs) with new business functions that deliver services to a broad range of users. The services need to be:
> • Highly available, to meet the needs of today's global business environment.
> • Secure, to protect the privacy of users and the integrity of the enterprise.
> • Reliable and scalable, to ensure that business transactions are accurately and promptly processed.

No anseio de desenvolver sistemas empresariais que alcançassem as metas listadas anteriormente, satisfazendo as expectativas dos clientes e ao mesmo tempo adequando-se as limitações de tempo e orçamento, foram desenvolvidos muitos modelos, técnicas e ferramentas de mercado para auxílio e suporte à criação de tais sistemas. A seguir, serão apresentadas algumas dessas práticas e metodologias com foco nas que foram utilizadas no presente projeto.

Uma das técnicas mais disseminadas para elaboração de projetos em geral é a decomposição do problema em pequenas partes, o que também é feito na construção de sistemas computacionais, ainda mais se tratando de aplicações distribuídas como

mencionado anteriormente. Atualmente, há muitos *frameworks*[9], principalmente na área de sistemas *Web*, que forçam a divisão do sistema em várias camadas e dão estrutura e organização ao mesmo.

Essa divisão em camadas aliada a boas práticas de modelagem que serão detalhadas adiante e a outros recursos disponibilizados por *frameworks*, traz diversas vantagens, incluindo o aprimoramento das capacidades de uma arquitetura, utilizadas para designar as qualidades não-funcionais de um sistema, tais como:

- **Capacidade** – quantas requisições simultâneas o sistema consegue processar por uma determinada unidade de tempo;
- **Confiabilidade** – medida da capacidade do software de manter seu nível de desempenho dentro de condições estabelecidas por um dado período de tempo (ISO/IEC 9126);
- **Desempenho** – capacidade de executar tarefas dentro do período de tempo especificado;
- **Disponibilidade** – grau de operabilidade do sistema, associado ao seu desempenho;
- **Escalabilidade** – capacidade de suportar a disponibilidade e o desempenho, à medida que a carga transacional aumenta;
- **Extensibilidade** – facilidade de estender as funcionalidades programadas;
- **Flexibilidade** – maleabilidade para alterações na arquitetura do projeto. Por exemplo, mudar de duas para três camadas, de três para quatro, etc.;

---

[9] Aqui framework se refere a uma estrutura de suporte definida a partir da qual um outro projeto de software pode ser organizado e desenvolvido. Tipicamente, um framework pode incluir programas de apoio, bibliotecas de código, linguagens de script e outros softwares para ajudar a desenvolver e juntar diferentes componentes do projeto a ser implementado, além de obrigar que o projeto siga uma dada estrutura pré-definida (*Wikipedia*). Refira-se ao apêndice III, para maiores informações sobre *frameworks*.

- **Manutenibilidade** – esforço exigido para localizar e reparar erros num programa, bem como realizar alterações no produto de software.
- **Reusabilidade** – capacidade de aplicar o mesmo componente em mais de uma solução;
- **Segurança** – capacidade de assegurar que as informações estão sendo acessadas pelos usuários de acordo com as políticas estabelecidas;
- Na figura 3 é apresentado um exemplo que mostra a infra-estrutura, em camadas, abaixo de uma aplicação *Web* que utiliza um *framework* e a plataforma J2EE.

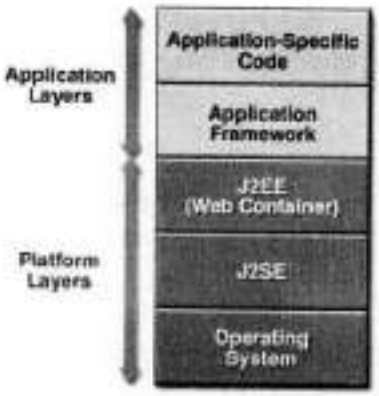


Figura 3 - Camadas da plataforma e da aplicação

No capítulo 5, que trata da implementação do projeto, há uma referência a um diagrama de seqüência genérico (Anexo K), elaborado durante a fase de definição da arquitetura, sendo uma boa ilustração das interações que ocorrem em uma arquitetura típica oferecida por *frameworks* que adotam o modelo MVC [10] (*Model-View-Controller*). De qualquer forma, na figura 4 é apresentado um exemplo que mostra esse fluxo de interações entre as camadas integrantes de uma arquitetura MVC.

---

[10] O MVC foi largamente difundido pelo livro "*Design Patterns*" de Erich Gamma et al, e é considerado um padrão de modelagem arquitetural que separa as responsabilidades de uma aplicação em 3 camadas: visualização (View), modelo (Model) e controle. Este modelo de arquitetura foi desenvolvido pela comunidade do *Smalltalk*.

Figura 4 – Interação entre camadas do *MVC*

Através da utilização deste modelo, o software é separado em 3 partes:

- **Modelo** – representa os dados e a lógica da aplicação através de entidades e classes de processos. Essa camada pode ser vista como uma aproximação, em software, das funcionalidades do mundo real. O modelo pode notificar a camada de visualização quando for modificado, ou pode permitir que a visualização verifique seu estado;
- **Controle** – controla o fluxo e define o comportamento da aplicação. É responsável por despachar as requisições dos usuários, interpretando os dados de entrada e mapeando-os em ações a serem executadas pelo modelo. Também é responsável por selecionar as visualizações a serem apresentadas ao usuário, além de controlar, autorizar e efetuar registros de acesso. Pode-se dizer que essa camada age como uma ponte entre o usuário e o modelo;
- **Visualização (ou Apresentação)** - todo componente de apresentação de resultados e geração de diálogo com o usuário faz parte desta camada, que recebe dados do modelo e especifica como eles devem ser apresentados. O modelo MVC possibilita muitas visualizações para um mesmo resultado.

No projeto utilizou-se o *framework* Struts, que é um dos mais populares do mercado. A idéia por trás do uso de *frameworks* MVC, como o Struts, é que eles provêem os seguintes benefícios:

- **Desacoplam apresentação e lógica de negócio em componentes separados** – encorajam a separação entre apresentação e lógica através do projeto de interfaces a serem estendidas que forçam essa separação;
- **Separam os papéis dos desenvolvedores** – fornecem diferentes visões e interfaces para diferentes desenvolvedores envolvidos no projeto. A divisão de tarefas entre desenvolvedores da lógica de negócio e desenvolvedores de componentes da camada de apresentação, por exemplo, permite que ambos trabalhem de maneira mais independente e promove a engenharia simultânea;
- **Provêem um ponto de controle centralizado** – a adoção de um servlet que intercepta todas as requisições (*frontController*) do usuário, facilita tarefas como verificação de acesso e segurança, *logging* e controle de fluxo, além de evitar a duplicação de código que seria gerada pela utilização de um controlador para cada tipo de visualização de dados;
- **Simplificam mecanismos de internacionalização**[11] – fornecem suporte a mecanismos flexíveis de internacionalização;
- **Simplificam as tarefas de validação** – a maioria dos *frameworks* possui mecanismos consistentes de validação, além de permitirem a escolha de se efetuar a validação do lado do cliente ou do servidor;
- **Encorajam o uso de componentes padronizados** – os *frameworks*, de modo geral, disponibilizam bibliotecas de *custom tags* para construção da camada de apresentação;

---

[11] Internacionalização é o processo de projetar uma aplicação de maneira que ela pode ser adaptada para vários idiomas e regiões sem mudanças na sua implementação.

- **Têm o suporte de comunidades** – *frameworks* populares atraem comunidades de usuários e entusiastas que reportam *bugs*, provêm consultorias e serviços de treinamento, publicam tutoriais e produzem itens adicionais ao *framework* bastante úteis.

Há muitos outros benefícios advindos da adoção de *frameworks* MVC que poderiam ser citados, no entanto o objetivo é passar uma idéia geral dos mesmos.

Além da adoção de *frameworks* de mercado, outra prática bastante comum na construção de sistemas *Web* e que traz muitos benefícios é a adoção de design *patterns*[12] (ou padrões de projeto e modelagem). Na verdade, a utilização de um *framework* já implica na utilização implícita de alguns *design patterns*, como visto no caso do MVC.

Segundo Gama (1992) "Um design *pattern* é uma descrição de comunicação de objetos e classes que são customizados para resolver um problema genérico de design em contexto particular", ou seja, pode-se entender um *pattern* como uma solução para problemas de modelagem que ocorrem com freqüência em situações específicas. Existem diversos *design patterns* para os mais variados problemas e situações enfrentados na implementação de um projeto orientado a objetos.

Os *design patterns* são normalmente separados em 3 categorias:

- Comportamentais;
- Estruturais e
- De criação;

Entre os mais difundidos estão os denominados *patterns* GOF (Gang Of Four), em uma alusão aos 4 autores[13] que definiram e difundiram a utilização de *design patterns* em projetos de software. Eles englobam um total de 23 *patterns*.

---

[12] O apêndice IV apresenta um artigo abordando o conceito de *design patterns*.

[13] Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides.

Para o desenvolvimento de soluções que utilizam a plataforma J2EE, há um conjunto de *patterns* derivados dos *patterns* GOF que são denominados *design patterns* J2EE e são descritos em Alur (2001).

Na solução proposta no presente trabalho, foram utilizados vários *patterns* GOF e J2EE, os quais serão detalhados na parte de especificação do projeto (capítulo 5).

Nesse capítulo não serão fornecidos detalhes sobre a utilização ou vantagens da adoção de cada *pattern* usado no projeto, mas sim uma visão geral dos benefícios da adoção de *patterns*. A seguir são listadas as vantagens mais relevantes:

- É uma maneira de compartilhar arquitetura entre projetos;
- É uma maneira de compartilhar conhecimento entre desenvolvedores;
- É uma padronização;
- É uma documentação implícita;
- Facilita o aprendizado;
- Facilita a manutenção;
- Impõe implicitamente melhores práticas em projetos orientados a objetos;
- Reduz o custo total de propriedade do software;
- Facilita a comunicação entre os desenvolvedores.

4 METODOLOGIA

4.1 Fases do projeto

O desenvolvimento do SARDA foi dividido em duas grandes etapas, as quais são subdivididas em fases que foram elaboradas ao longo das disciplinas PCS 2501 – Projeto de Formatura I e PCS 2502 – Projeto de Formatura II:

- **Primeira Etapa (PCS 2501) – abordou as seguintes fases:**
  - Definição do tema e escopo do projeto;
  - Pesquisas e estudos de bibliografia correlata;
  - Especificação dos requisitos do sistema;
  - Início da modelagem da arquitetura do sistema (divisão em módulos fundamentais ou unidades lógicas);
  - Análise do sistema;

- **Segunda Etapa (PCS 2502) – abordou as seguintes fases:**
  - Projeto do sistema;
  - Escolha das ferramentas e instalação do ambiente de desenvolvimento;
  - Implementação do sistema;
  - Testes e validação do sistema;
  - Análise dos resultados;
  - Documentação do projeto.

4.2 Processo de desenvolvimento de software

Para este projeto foi adotado um processo de desenvolvimento baseado no modelo iterativo-incremental. Para definir os elementos a serem desenvolvidos iterativamente, o sistema foi dividido em unidades lógicas. O desenvolvimento destas unidades lógicas foi priorizado e realizado seqüencialmente. Assim, a experiência adquirida em um módulo foi aplicada nos demais módulos. Outro aspecto importante é que a construção incremental do software facilitou o controle dos prazos do projeto, uma vez que a capacidade de estimativa de tempos de desenvolvimento foi melhorando durante o projeto. Estimar prazos fica mais fácil quando a equipe fica mais experiente e também possui dados históricos sobre o projeto.

O processo de desenvolvimento de software adotado é sumarizado através do seguinte procedimento:

1. Especificação dos requisitos do sistema;
2. Definição dos grandes módulos do sistema (divisão em unidades lógicas);
3. Definição dos testes de aceitação para cada unidade;
4. Para cada unidade lógica:
    a. Análise
        i. Modelo de casos de uso (baseado nas especificações levantadas);
        ii. Modelo de classes (nível 0 – definição de classes e relacionamentos);
        iii. Cenários de teste;
    b. Projeto (*Design*)
        i. Modelo de classes (nível 1 – detalhamento das classes através de seus métodos e atributos);
        ii. Projeto da arquitetura da unidade lógica (inclui descrição dos *design patterns* adotados);

iii. Modelo de interação (inclui diagrama de seqüência ilustrativo da arquitetura adotada);

iv. Modelo da interface com o usuário (ou com outra unidade lógica);

v. Verificação da consistência com documento de requisitos;

c. Implementação

i. Geração de código através de ferramenta CASE (incluindo interfaces, classes abstratas e classes concretas do modelo de negócio);

ii. Criação e implementação do modelo E-R (Entidade-Relacionamento), gerando tabelas do banco de dados;

iii. Criação das classes de manipulação[14] de dados através da implementação dos DAOs (*Data Access Objects*);

iv. Teste unitário dos DAOs;

v. Implementação das funcionalidades previstas;

vi. Verificação com documento de casos de uso;

d. Testes

i. Execução dos cenários de teste;

ii. Depuração;

5. Documentação da unidade;
6. Teste integrado;
7. Teste de aceitação;
8. Compilação e revisão da documentação final.

Nem sempre este procedimento foi seguido à risca, uma vez que em alguns momentos do projeto houve a necessidade de se adiantar algumas etapas em detrimento de outras, com a finalidade de cumprir entregas demandadas pela disciplina e também pra superar atrasos no cronograma traçado.

[14] Manipulação de dados inclui inserção, acesso, atualização e exclusão dos mesmos.

5 ESPECIFICAÇÃO, ANÁLISE E PROJETO DO SISTEMA

A arquitetura de software do SARDA é composta por dois grandes módulos, conforme apresentado na Figura 6:

- **Infra-estrutura (Sistema de EAD)** - ambiente educacional simplificado onde é possível efetuar o gerenciamento[15] de alunos, professores, cursos, turmas, matrículas, materiais didáticos e exames;
- **Módulo de Apoio ao Diagnóstico de Estudantes (MADE)** – módulo contento mecanismos de suporte e heurística para avaliação do desempenho de estudantes frente a objetivos educacionais traçados por professores.

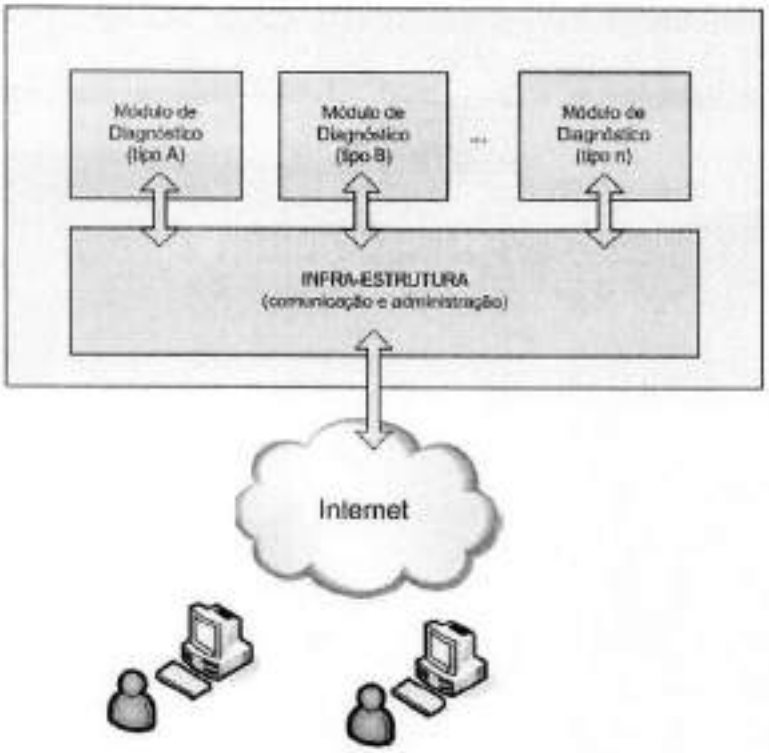


Figura 5 - Arquitetura do Sistema SARDA

[15] Por gerenciamento entende-se: inclusão, visualização, atualização e remoção de cadastros.

Note que na figura há diversos módulos de apoio ao diagnóstico de estudantes conectados ao módulo de infra-estrutura, mostrando que é possível ter vários módulos desse tipo acoplados ao SARDA. Além disso, as interfaces destes módulos com a infra-estrutura são padronizadas, para que eles possam ser intercambiados ou substituídos, de maneira a prover diferentes métodos de avaliação.

No contexto deste trabalho, desenvolveu-se um módulo de apoio ao diagnóstico de estudantes contendo uma heurística simples de avaliação, com o intuito de exercitar a interface com a infra-estrutura e também para servir de exemplo para a implementação de módulos que sigam a interface definida.

A seguir, são apresentadas as especificações dos requisitos e a modelagem de cada uma das unidade lógicas do SARDA. O desenvolvimento foi feito usando UML (*Unified Modeling Language*), com o apoio da ferramenta JUDE/Community (versão 1.5.3), que é uma ferramenta gratuita (no CD é fornecida a última versão dessa ferramenta para visualização dos arquivos gerados).

Todos os diagramas gerados no decorrer do desenvolvimento podem ser encontrados no CD em anexo, no arquivo denominado sarda_modeling-1.0.jude[16].

---

[16] O arquivo "sarda_modeling-1.0.jude" segue uma estrutura particular e contém os seguintes recursos: há um pacote denominado *Sarda2005* que contém os diagramas de casos de uso, de classes e sequência. Os diagramas de casos de uso estão agrupados segundo módulos que contém os casos de uso de um determinado ator (member, student, teacher ou administrator). Há também um pacote denominado *br* que contém a hierarquia de classes utilizadas nos diagramas e para geração automática de código.

5.1 Módulo de Infra-Estrutura

Nesta seção são apresentados os requisitos[17] funcionais e não-funcionais do módulo de infra-estrutura, assim como a arquitetura adotada para atender a esses requisitos.

A finalidade do módulo de infra-estrutura é fornecer as funcionalidades básicas de um ambiente educacional, onde se pode testar o módulo de apoio ao diagnósticos estudantes proposto. Esse módulo, apesar de não possuir todas as funcionalidades e recursos de um sistema de EAD comercial, foi desenvolvido de maneira a ser flexível e extensível para facilitar futuras expansões através de novas funcionalidades e recursos.

5.1.1 Especificação dos Requisitos Funcionais

O levantamento dos requisitos funcionais do módulo de infra-estrutura foi feito com base na observação de alguns sistemas de EAD, como COL, Panda, Moodle, dentre outros, e também com base na experiência do grupo no desenvolvimento de sistemas *Web*.

Tem-se, a seguir, a descrição das funcionalidades do módulo de infra-estrutura, agrupadas por atores:

- **Member** – é uma generalização (abstração) dos atores do sistema, de maneira que pode desempenhar as tarefas que são comuns a todos os demais atores, tais como: efetuar *log on* e *log out* no sistema e alterar dados pessoais.
- **Student** – representa um estudante registrado no sistema, o qual pode realizar as atividades de gerenciar seus dados, acessar material para estudos, realizar tarefas *on-line* disponíveis no curso do professor, verificar notas obtidas, entrar em contato com o professor, dentre outras funcionalidades;
- **Teacher** – representa um professor registrado no sistema, o qual pode criar e gerenciar cursos on-line. O gerenciamento de cursos envolve o controle dos

[17] Os requisitos do módulo de infra-estrutura foram levantados no primeiro semestre de 2005 durante a disciplina PCS 2501 – Projeto de Formatura I.

recursos associados a eles, tais como: materiais didáticos (incluindo exames e exercícios disponibilizados para os alunos), turmas e matrículas de alunos, etc. Note que professores não gerenciam alunos, apenas criam turmas e associam alunos previamente cadastrados no sistema a essas turmas;

- **Administrator** – representa um super-usuário cadastrado previamente no sistema e que é responsável por gerenciar os professores e alunos desse sistema. Ele também pode realizar tarefas consignadas aos professores, tal como gerenciar cursos e seus recursos (materiais didáticos, turmas, etc).

Maiores detalhes das funcionalidades do módulo de infra-estrutura podem ser encontrados nos anexos C, D, E, F, G e H, que contêm diagramas, tabelas e cartões de casos de uso. Note que no anexo H são apresentados os cartões de casos de uso apenas do professor (*teacher*), para fins de ilustração. A especificação dos demais casos de uso se encontra no arquivo sarda.sw.uc.doc, incluso no CD.

5.1.2 Modelo de Classes

Uma vez especificadas as funcionalidades do módulo de infra-estrutura, passou-se a fase de análise do mesmo. Nesta fase, as informações contidas nos casos de uso apresentados anteriormente, foram traduzidas na representação de classes e seus relacionamentos. Os anexos I e J apresentam, os diagramas de classes do módulo de infra-estrutura. O anexo I mostra as classes de negócio (concretas e abstratas) e os seus relacionamentos. O anexo J mostra o mesmo diagrama de classes, com os atributos e os métodos de cada classe. Para maiores informações, deve-se consultar o arquivo sarda_modeling-1.0.jude contido no CD.

5.1.3 Arquitetura

Como mencionado no capítulo 3, a arquitetura adotada para o módulo de infra-estrutura é do tipo *Web*-centric com *thin clients*, pois esse modelo é apropriado para simular um ambiente de *e-learning* típico, fornecendo as funcionalidades básicas de tais ambientes. Vale lembrar que a idéia deste módulo, como o próprio nome diz, é servir de infra-estrutura básica para o teste das técnicas de apoio à realização de diagnósticos do aprendizado propostas pelo projeto.

Como fator secundário na escolha dessa arquitetura, pode-se mencionar o fato de os integrantes do grupo possuirem know-how nas tecnologias voltadas ao desenvolvimento de aplicativos *Web*, o que acelerou o desenvolvimento dessa infra-estrutura básica, que não é o foco do projeto, e diminuiu os riscos do mesmo.

A especificação da arquitetura vai muito além da escolha de um modelo *Web-based*, sendo que esta arquitetura pode ainda ser organizada em camadas lógicas, separando as responsabilidades do sistema e promovendo reutilização, manutenibilidade e extensibilidade.

Para alcançar bons desempenhos nos requisitos não-funcionais citados no capítulo 3, optou-se por uma arquitetura modular, a qual é obtida através da adoção do padrão de projeto arquitetural MVC e da aplicação de outros padrões de projeto (*design patterns*) reconhecidos pelo mercado.

O padrão MVC, como já mencionado no capítulo 3, traz uma série de benefícios ao sistema a ser desenvolvido, e tem sido bastante utilizado tanto na construção de aplicações *desktop* que utilizam interfaces gráficas complexas, quanto no desenvolvimento de aplicações *Web* (como é o caso do sistema proposto nesse trabalho). A implementação desse padrão foi feita através da utilização de um *framework* que força a divisão do sistema nas camadas do MVC (*Model, View and Control*), como pode ser visto no capítulo 6.

Outros componentes e padrões de projetos, que também foram adotados nessa arquitetura, são descritos a seguir:

- *Singleton* GoF *design pattern* – garante que só haverá um objeto de uma determinada classe. É utilizado junto com outros *patterns* listados a seguir, tal como *Service Locator* e *Data Access Object*.
- *Data Access Object* J2EE *design pattern* – abstrai e encapsula a lógica de acessos a dados para recursos específicos. No caso desse projeto, esse *pattern* é utilizado para encapsular os detalhes de implementação do acesso ao banco de dados relacional (os DAOs contém todo o código SQL). Os diagramas ilustrativos desse *pattern* se encontram no apêndice I.

- *Service Locator* J2EE *design pattern* – simplifica acesso do cliente a serviços de negócio enterprise, encapsulando a complexidade de operações de *lookup*, que são usadas por aplicações clientes para localizar objetos gerenciados que provêm serviços. No caso desse projeto, o *Service Locator* será utilizado para acessar o Pool de conexões ao banco de dados. Os diagramas ilustrativos desse *pattern* se encontram no apêndice I.
- *Front Controller* J2EE *design pattern* – centraliza requisições de usuários. O controlador fornece um ponto centralizado para gerência e controle do tratamento de requisições *Web*. Essa centralização reduz a uso de lógica de negócio (processamentos) na camada de apresentação e promove a reutilização de código entre as requisições. O controlador utilizado nesse projeto é fornecido pelo *framework* adotado (maiores detalhes se encontram no próximo capítulo) e faz parte da camada de controle da arquitetura MVC. Os diagramas ilustrativos desse *pattern* se encontram no apêndice I.
- *Intercepting Filter* J2EE *design pattern* – utilizado para pré e pós-processamento de requisições. No caso desse projeto, o *Intercepting Filter* é utilizado para adicionar serviços de segurança (verificação da permissão de acesso a conteúdos) e *logging* ao processamento principal das requisições. Os diagramas ilustrativos desse *pattern* se encontram no apêndice I.
- *Command* GoF *design pattern* – Encapsula uma ação acionada por um cliente, sem que ele conheça os detalhes da ação. A implementação desse *pattern* também é realizada através das interfaces fornecidas pelo *framework* adotado no projeto. Esse *pattern*, junto com o *pattern Front Controller*, é parte integrante do modelo MVC. Os diagramas ilustrativos desse *pattern* se encontram no apêndice I.
- *View Helper* J2EE *design pattern* – encapsula a lógica de apresentação e acesso a dados da camada de visualização do modelo de MVC, mantendo a implementação desta camada mais simples e organizada. A lógica de apresentação trata de formatação de dados a serem exibidos em um página, enquanto a lógica de acesso a dados trata a recuperação do dados a serem

apresentados. *View Helpers* são implementados através de *custom tags*, para renderização e formatação de dados, e JavaBeans para recuperação de dados. Nesse projeto, as informações são disponibilizadas para os componentes da camada de visualização através de JavaBeans e também são utilizadas diversas *tags* customizadas fornecidas pelo *framework* adotado. Os diagramas ilustrativos desse *pattern* se encontram no apêndice I.

- *Composite View* J2EE *design pattern* – gerencia *layout* do conteúdo composto por múltiplas visualizações. Esse *pattern* foi utilizado no projeto através da criação de um *template* para interface com o usuário, que trata de maneira automatizada os componentes comuns (e estáticos) dessa interface. A interface foi dividida em: *top* (*header*), *menu*, *centre* e *bottom* (*footer*), conforme apresentado na próxima seção. Os diagramas ilustrativos desse *pattern* se encontram no apêndice I.

- *Transfer Object design pattern* – efetua transferência de dados entre camadas. Esse *pattern* reduz o custo da comunicação distribuída (entre camadas), através do encapsulamento de um conjunto de dados relacionados em um único *transfer object* (*value object*). Esse *pattern* é bastante utilizado no projeto, por exemplo, quando os dados de endereço de um dado usuário (cidade, bairro, rua, CEP, etc.) são encapsulados em um objeto da classe *Address* para troca de informações. Os diagramas ilustrativos desse *pattern* se encontram no apêndice I.

O anexo K apresenta um diagrama de seqüência genérico[18] ilustrativo da arquitetura adotada. Esse diagrama foi elaborado durante a fase de projeto e tem o intuito de mostrar como a utilização dos diversos *design patterns*, citados anteriormente, afeta o fluxo de chamadas e molda a interação entre os diversos componentes dessa arquitetura.

---

[18] Genérico é utilizado com o sentido de que esse diagrama não especifica a operação (inclusão, atualização, exclusão, etc) de fato sendo realizada no cenário em questão.

5.1.4 Interface com o Usuário

Conforme mencionado anteriormente, não se pretende que o módulo de infra-estrutura ofereça os recursos de um sistema de EAD comercial, sendo seu foco apenas de fornecer uma plataforma com os recursos básicos de um ambiente de EAD para teste do mecanismo de acompanhamento de estudantes proposto. Portanto, os mesmos critérios utilizados tanto na definição dos requisitos funcionais, quanto na modelagem da arquitetura desse módulo, foram também utilizados no projeto da interface com o usuário: manter a simplicidade (KIS – *Keep It Simple*), porém com flexibilidade.

Seguindo os preceitos mencionados, projetou-se uma IHC (Interface Homem-Computador) meramente funcional, porém modular e desacoplada da lógica de negócios. Dessa maneira, tem-se uma interface simples, que atende os requisitos do presente trabalho e, ao mesmo tempo, tem-se a flexibilidade necessária para substituição desta por uma nova interface.

Para atingir as características traçadas, o projeto da IHC baseou-se em três padrões de projeto listados na seção anterior: MVC, *Composite View* e *View Helper*.

O MVC permite o desacoplamento entre a IHC e a lógica de negócios, provendo assim, flexibilidade para troca dessa interface.

O *View Helper* introduz a utilização de componentes de suporte à interface, o que possibilita a engenharia simultânea, a reutilização de código (inclusive de terceiros) e o conseqüente aumento da produtividade na fase de implementação da interface.
Já o padrão *Composite View*, ajuda a modularizar a interface em fragmentos atômicos e bem definidos, facilitando o gerenciamento de mudanças no *layout* e promovendo a reutilização desses fragmentos. Através desse padrão, a interface foi estruturada da seguinte maneira:

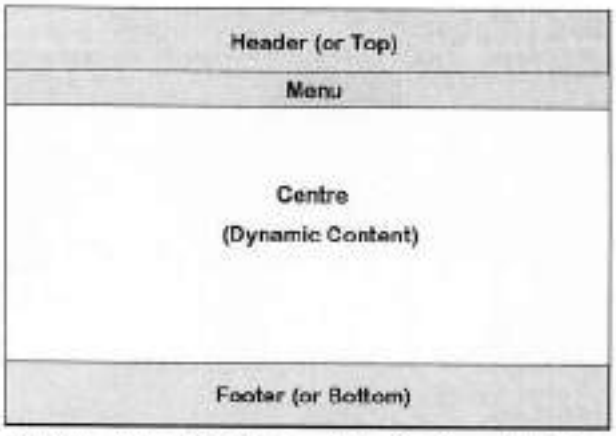


Figura 6 - *Layout* Utilizado para a Interface Homem-Máquina

Obs.: No diagrama utilizou-se os nomes adotados na implementação, por isso estão em inglês.

O posicionamento das regiões da interface é definido em uma página de *template* (*index*) e, sendo assim, pode-se alterar o *layout* da interface através dessa única[19] página. Poder-se-ia, por exemplo, alterar o *layout* anterior para o seguinte *layout*:

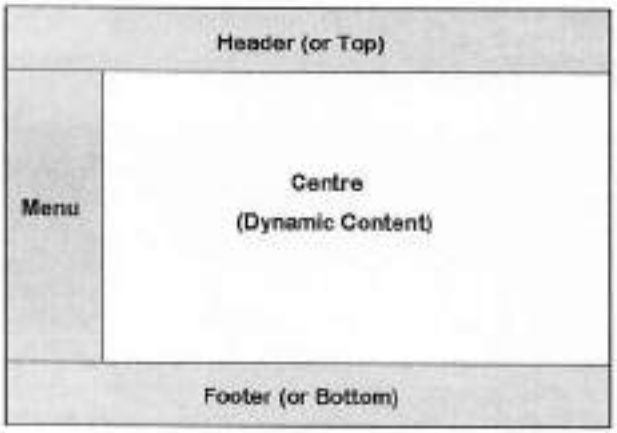


Figura 7 - *Layout* da Interface Homem-Computador com Menu Vertical

[19] Na implementação do projeto da IHC, optou-se pela utilização de uma página de template (*index*) para o ambiente de cada ator previsto, ou seja, há um *index.jsp* que define o *layout* do ambiente do administrador, outro para o ambiente do professor e um terceiro para o ambiente do aluno. Essa implementação permite configurar um *layout* diferente para usuários com papéis diferentes.

Os dois *layouts* mostrados anteriormente são freqüentemente encontrados em sistemas *Web*. Porém, há diversas outras combinações que poderiam ser feitas como, por exemplo, utilizar dois menus replicados em posições simétricas, passar o menu vertical da direita para esquerda, dentre outras. O importante de observar, é que essas alterações são feitas em uma única página e se propagam para todas as interfaces do sistema (ou a cada ambiente de usuário previsto).

Para acesso ao seu ambiente, o usuário passa por uma tela de *logon* no sistema que segue o *template* indicado a seguir:

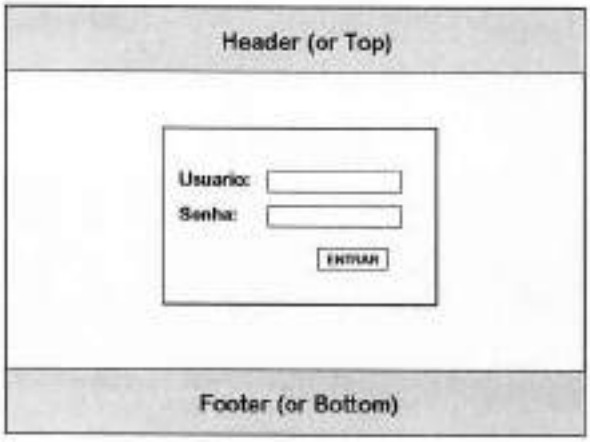


Figura 8 - Tela de Entrada do Sistema (Autenticação)

A autenticação do usuário é muito importante para a gerência e montagem do *layout* do sistema, uma vez que esse *layout* é definido de acordo com o papel do usuário que está acessando o sistema. Também é através da autenticação que se faz o controle do acesso aos conteúdos disponibilizados no sistema. Esse controle também é baseado em papéis; por exemplo, alunos não podem acessar o conteúdo destinado a professores ou administradores. No capítulo 6, destinado aos aspectos de implementação do sistema, são apresentados maiores detalhes sobre os mecanismos de controle de acesso a conteúdos e segurança de uma forma geral.

5.2 Módulo de Apoio ao Diagnóstico de Estudantes (MADE)

O MADE é a entidade do SARDA responsável pelo fornecimento de diagnósticos, relatórios e estatísticas que visem a melhoria do processo de ensino em plataformas de ensino à distância. O SARDA permite a adoção de múltiplos MADEs, uma vez que cada tipo de curso ministrado pode exigir diferentes tipos de diagnósticos de aprendizagem.

5.2.1 Especificação dos Requisitos Funcionais

Os módulos pedagógicos devem ser conectados facilmente ao sistema.

Deve existir um mecanismo de adaptação de sistemas já existentes com os módulos pedagógicos.

5.2.2 Modelo de Casos de Uso

O aluno interage com o MADE através do caso de uso *Take test* (Figura 9).

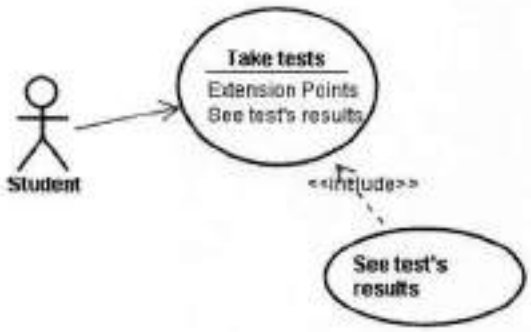


Figura 9 - Caso de uso *Take Test*

Quando o aluno opta por fazer um teste, este é exibido na tela. O aluno por sua vez, ao responder ao teste e acionar a opção "Ver resultados" executa o caso de uso *See test results*. Este caso de uso aciona o MADE para a obtenção do diagnóstico de aprendizagem correspondente ao teste respondido.

5.2.3 Modelo de Classes

A implementação do *framework* para módulos pedagógicos (MADEs) envolveu a criação de três pacotes básicos de classes: *adapter*, *evaluationsystem* e modules.

O pacote *adapter* contém uma classe que faz o papel de integração entre o sistema de Ensino à Distância (EaD) e o controlador de módulos pedagógicos, fornecendo todas as informações necessárias para a realização de diagnósticos de aprendizagem pelos MADEs.

O pacote *evaluationsystem* contém os algoritmos necessários para a realização de diagnósticos de aprendizagem. A realização de diagnósticos ocorre em três fases:

1. Extração de dados
2. Processamento
3. Formatação dos resultados

A interface do framework para MADEs com o mundo externo é realizada através de um *Façade* que faz toda a comunicação do controlador de módulos pedagógicos com o mundo externo.

5.2.4 Arquitetura

O modelo conceitual para a arquitetura do sistema prevê (Figura 10):

- A capacidade de adaptação de sistemas de ensino à distância (EaD) já existentes ao controlador de módulos pedagógicos.
- A capacidade de instalação, remoção e troca de módulos pedagógicos (MADEs), sem necessidade de desligamento do sistema.
- A possibilidade de criação de novos módulos que necessitem de informações específicas, tais como: histórico de navegação do aluno pelo sistema, quantidade de alunos matriculados em determinada turma e quantidade de tempo em que o aluno manteve aberto determinado tópico.

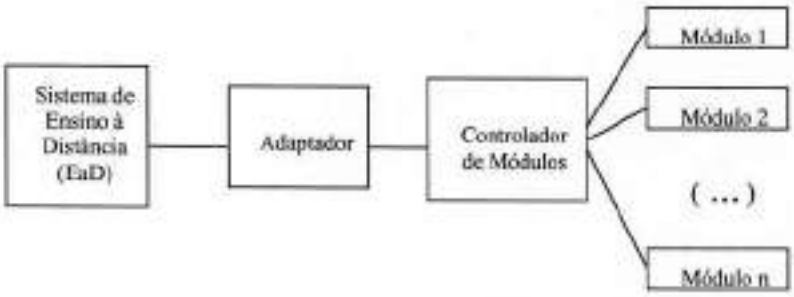


Figura 10 - Modelo Conceitual da Arquitetura dos Módulos Pedagógicos

Abaixo são listados os padrões de projetos que foram adotados nessa arquitetura. O modelos conceituais para cada um dos padrões são apresentados no apêndice III.

Façade GoF design *pattern* – De acordo com Gamma et al (1995): "Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use". Permite a simplificação do uso de um sistema complexo, com o objetivo de expor um subconjunto deste sistema ou usar o sistema de uma forma particular.

Template Method GoF design *pattern* – A função deste padrão é ajudar na abstração de um processo comum para diferentes funcionalidades. O padrão *Template Method* define o esqueleto de um algoritmo para alguma operação, delegando alguns passos para sub-classes.

Strategy GoF design pattern – O padrão *Strategy* define uma família de algoritmos, encapsula cada um deles e os torna intercambiáveis, permitindo que um algoritmo varie independentemente dos clientes o usam (GAMMA et al, 1995). Na prática, o padrão *Strategy* permite a utilização de diferentes lógicas de negócio ou algoritmos dependendo do contexto em que eles ocorrem. Isto é possível através da separação da seleção do algoritmo de sua implementação.

5.2.5 Interface com o Módulo de Infra-Estrutura

Um item importante na arquitetura do sistema SARDA é o módulo que realiza a interface entre o gerenciador de módulos pedagógicos (MADE) e o sistema de gerenciamento de aprendizagem (EaD). O principal objetivo a ser alcançado por esta

interface é a possibilidade de utilização de módulos pedagógicos por sistemas de ensino à distância já existentes, com um mínimo esforço e com alto grau de reutilização.

Durante a fase de projeto da interface entre os MADEs e o EaD, optou-se por uma solução que garantisse baixo acoplamento (i.e.: interdependência) e alta coesão (i.e.: independência) entre os módulos. Gamma et al (1995) e outros autores afirmam que preferência deve ser dada à instanciação de objetos em detrimento ao uso de herança de classes. Este princípio foi aplicado em conjunto com alguns *design patterns* - em especial o padrão *Template Method*. Os diagramas de classes da solução proposta são apresentados nos anexos L e M.

6 IMPLEMENTAÇÃO DO SISTEMA

A implementação do sistema proposto teve início na segunda disciplina de projeto de formatura, PCS 2502 – Projeto de Formatura II, e teve como base o documento[20] com a especificação do projeto elaborado anteriormente.

Nas seções que seguem, são apresentadas as diversas etapas, juntamente com as dificuldades enfrentadas e as decisões tomadas durante a implementação do sistema.

6.1 Tecnologia Adotada

O sistema proposto foi desenvolvido em Java, cuja escolha foi feita pelos seguintes motivos[21]:

- Em primeiro lugar, devido ao fato de que os integrantes do grupo já possuíam know-how no desenvolvimento de software utilizando Java;
- Em segundo, deve-se ressaltar que JAVA é orientada a objetos e disponibiliza mecanismos poderosos e padronizados tais como, encapsulamento, herança, polimorfismo, tratamento de exceções, gerenciamento de *threads*, dentre outros, para a construção de softwares robustos e independentes de plataforma;
- Por fim, existe uma grande quantidade de APIs e componentes prontos disponibilizados pela comunidade Java, além da variedade de ferramentas de suporte ao desenvolvimento usando essa tecnologia.

---

[20] O documento de ERS (Especificação de Requisitos de Software) foi adicionado ao CD e foi designado por "sarda.sw.rs.1.0.doc".

[21] No CD há um artigo intitulado "Características do Java", que lista alguns dos pontos fortes da adoção de linguagem.

6.2 Plataforma e Ambiente de Desenvolvimento

Antes do início da implementação em si, realizou-se um processo de preparação do ambiente de desenvolvimento. Esta preparação foi feita em três etapas:

- Pesquisa e avaliação de ferramentas;
- Escolha das ferramentas;
- Instalação do ambiente de desenvolvimento.

Durante a fase de pesquisa, foram feitos levantamentos de alguns *plug-ins* e *frameworks* voltados para o desenvolvimento de sistemas *Web* em Java e que fossem suportados pelos ambientes de desenvolvimento que estavam sendo avaliados para utilização no projeto (Eclipse e Netbeans). Optou-se pela utilização do *framework* Struts e do plugin WTP (*Web Tool Plataform*), que são descritos nas próximas sessões.

O processo de escolha das ferramentas levou em consideração os seguintes critérios:

- Opção por ferramentas gratuitas e de código aberto;
- Habilidade dos membros do grupo no manuseio das ferramentas;
- Conjunto de funcionalidades e facilidades disponibilizadas pelas ferramentas;
- Portabilidade e compatibilidade entre as ferramentas.

Pode-se dizer que os dois primeiros critérios listados anteriormente foram os que nortearam de maneira mais significativa o processo de escolha das ferramentas.

A opção por de ferramentas disponibilizadas gratuitamente sob licenças GPL, LGPL e Apache adveio não apenas por desonerar o grupo dos custos inerentes à aquisição de softwares pagos, mas também pelo nível de excelência em que se encontram muitos dos projetos *open-source* de hoje, os quais produzem ferramentas de alta qualidade.

A tabela 1 apresenta um resumo com as ferramentas utilizadas na implementação do sistema:

Tabela 1 - Ferramentas de Desenvolvimento

Tipo de recurso	Produto	Fornecedor
Plataforma de desenvolvimento	*J2SE Software Development Kit* 1.5.0_4	*Sun Microsystems*
Ambiente de desenvolvimento	Eclipse 3.1	Eclipse.org
Plug-in para desenvolvimento web	*Web Standard Tool**	Eclipse.org
Ferramenta para automação de *builds*	*Apache Ant 1.6.5*	*Apache Software Foundation*
Servidor / Container web	*Jakarta Tomcat 5.0.28*	*Apache Software Foundation*
RDBMS	*MySQL Server 4.1*	*MySQL.org*
Driver do banco de dados	*MySQL Connector/J 3.1*	*MySQL.org*
Framework MVC	Struts 1.2.7	*Apache Software Foundation*
Biblioteca de *custom tags*	*JSTL / Standard (JavaServer Pages Standard Tag Library 1.1)*	*Apache Software Foundation*
Bibliotecas de componentes	*Jakarta commons*	*Apache Software Foundation*
API para geração de gráficos	JFreeChart 1.0	JFreeChart.org
Biblioteca de *tags* para manipulação de gráficos em aplicações web	Cewolf 0.12	Cewolf.sourceforge.net
Ferramenta para *Stress Test*	JMeter 2.1.1	*Apache Software Foundation*

* O *plug-in* WST depende de outros *plug-ins* (EMF, GEF e JEM *drivers*) da Eclipse.org para seu funcionamento (veja instruções no sítio www.eclipse.org).

No apêndice III é apresentada uma lista com endereços dos fornecedores das ferramentas na Internet.

Além das ferramentas citadas, também foram criadas duas contas em *sites* especializados:

- *Yahoo* (*Yahoo* Grupos), que serviu como repositório de códigos e documentos;
- Locaweb (empresa especializada na hospedagem de *websites*), onde foi instalado o ambiente de execução[22] da aplicação.

[22] Um segundo ambiente de execução foi instalado em um servidor do LSA-POLI.

6.3 Módulo de Infra-Estrutura

A seguir são apresentados os pontos relevantes das técnicas e estratégias adotadas na implementação do módulo de infra-estrutura.

6.3.1 Struts: *Framework* MVC

A implementação do módulo de infra-estrutura foi feita utilizando-se o *framework* MVC Struts, distribuído gratuitamente sob a licença *Apache*[23].

Os aspectos conceituais desse *framework*, importantes para melhor entendimento dos detalhes de implementação, estão apresentados no capítulo 3.

6.3.1.1 Struts-Blank *Web Application*

A implementação do sistema de infra-estrutura proposto foi baseada em uma aplicação denominada strus-blank.war[24], que é fornecida junto com o *framework*. Essa aplicação é bastante útil para novos usuários do *framework*, pois já fornece a estrutura de diretórios básica para uma aplicação *Web* em Java e também inclui TLDs (*Tag Library Descriptors*), bibliotecas[25] (JARs) com componentes essenciais do *framework* e arquivos de configuração imprescindíveis. A idéia desta aplicação é fornecer a estrutura e as configurações básicas para a utilização do Struts. Abrindo-se os arquivos de configuração inclusos nessa aplicação, pode-se notar uma enorme quantidade de comentários exemplificando operações comuns, tais como mapeamento de *taglibs* e servlets fornecidos pelo *framework*.

---

[23] A última versão licença da *Apache Software Foundation* pode ser encontrada em http://www.apache.org/licenses/LICENSE-2.0

[24] Essa e outras aplicações de exemplo são fornecidas junto com o *framework* e podem ser encontradas no diretório *webapps*, que é gerado após a descompactação do *framework*.

[25] O arquivo struts.jar contém todos os TLDs e componentes essenciais do *framework*.

6.3.1.2 Componentes do *Framework*

A seguir são apresentados os componentes padrões que o Struts fornece e que foram utilizados para a implementação do módulo de infra-estrutura do SARDA.

O foco do Struts está no fornecimento de componentes para as camadas de controle e de visualização. Os componentes da camada de negócio são muito dependentes do contexto da aplicação e foram implementados utilizando o padrão *JavaBeans*.

6.3.1.2.1 Camada de Controle

O Struts já fornece uma implementação do *Front Controller*, que é o componente principal da camada de controle, responsável por receber as requisições com dados de entrada do cliente, chamar a camada de negócios e enviar resposta com dados de apresentação para os usuários. O *Front Controller* do Struts é denominado *ActionServlet*[26], o qual é um servlet como outro qualquer e, portanto, deve ser mapeado no arquivo de configuração da aplicação (web.xml) para atender requisições de usuários que sigam um determinado padrão[27] (*url-pattern*).

Outros componentes do Struts pertencentes à camada de controle e essenciais ao entendimento da aplicação são as *Actions*, as quais podem ser vistas como extensões do *FrontController*. Elas desacoplam a requisição do cliente da lógica de negócios.

A implementação de uma *Action* é feita estendendo-se a classe *Action* fornecida pelo *framework*. Essa classe tem muitos métodos, mas o mais importante deles é o método execute, o qual deve ser sobrescrito de maneira a fazer a ligação entre a requisição enviada pelo cliente e a camada de negócio. Cada *action* é criada com a finalidade de

---

[26] O *ActionServlet* estende a classe *HttpServlet* fornecida pela API de servlets (pacote *javax.servlet.http*).

[27] Na aplicação "*struts-blank.war*", o *ActionServlet* já vem configurado para atender as requisições que terminam com ".do", e esse mapeamento foi mantido na implementação do projeto.

executar uma dada tarefa [28](por exemplo, inserir um usuário), e deve ser mapeada no arquivo struts-config.xml para atender a determinadas requisições (por exemplo, requisições cuja URL sejam "/inserirUsuario.do").

Quando o controlador recebe uma requisição de um usuário, ele acessa o arquivo struts-config.xml, verifica, seleciona e instancia[29] a classe (*action*) apropriada para atender a essa requisição, e então chama o método *execute* da classe. Note que o controlador vale-se de uma chamada polimórfica ao método execute da classe *Action* que, quando ativado, executa o código especificado na sobrescrita desse método na implementação de uma *action*.

As *actions* também são responsáveis por especificar ao *controller* o destino para onde deve ser feito o forward da requisição contendo os dados a serem apresentados ao usuário. O mapeamento do recurso para o qual é feito o forward, normalmente um servlet ou JSP, encontra-se também no arquivo struts-config.xml, o que minimiza o acoplamento entre a aplicação e um recurso físico.

Além do *controller* e das *actions*, há um terceiro grupo de componentes da camada de controle, que são denominados *action forms*. Os *action forms* nada mais são que simples VOs (*Value Objects*) utilizados pra trocar informações entre a camada de visualização e camada de controle. Eles representam um formulário HTML e são implementados através de *JavaBeans* que estendem a classe *ActionForm*[30] fornecida pelo *framework*. Um *action form* deve conter um atributo para cada campo de dado

---

[28] Algumas vezes utiliza-se uma mesma action para executar todas as tarefas referentes ao CRUD (*Create, Retrieve, Update e Delete*) de uma dada entidade (usuário, por exemplo). O mais comum, no entanto, é dividir essas atividades em várias *actions* (o que aumenta a flexibilidade).

[29] Nem sempre é necessário que se crie uma instância, podendo-se utilizar uma previamente instanciada. Isso se deve ao fato de que o *controller* utiliza uma única instância para atender múltiplas requisições.

[30] O Struts também fornece as classes *DynaActionForm* e *ValidatorForm*, que são subclasses de *ActionForm* e podem ser estendidas para implementação de um *action form*. Cada uma dessas classes fornecem vantagens adicionais à classe *ActionForm*.

do formulário que ele representa, assim como os respectivos métodos de acesso a esses atributos (*getters* e *setters*). Um *action form* pode estar associado a uma ou mais *actions*, que usam os dados contidos nesses *beans* (ou VOs) para executarem suas ações. A associação entre *action forms* e *actions* também é feita através do arquivo struts-config.xml.

Quando um formulário é submetido por um usuário da aplicação, o *framework* automaticamente carrega o *action form* apropriado com os dados submetidos e então, repassa esse *action form* para a *action* que irá processar a requisição.

O *framework* também disponibiliza mecanismos de validação de formulários através dos *action forms*, de maneira que se um usuário deixa de preencher um campo obrigatório ou preenche um campo incorretamente, ele será notificado automaticamente. Na sessão X são apresentados detalhes do mecanismo de validação.

A figura 11 ilustra a interação dos componentes da camada de controle com as demais camadas.

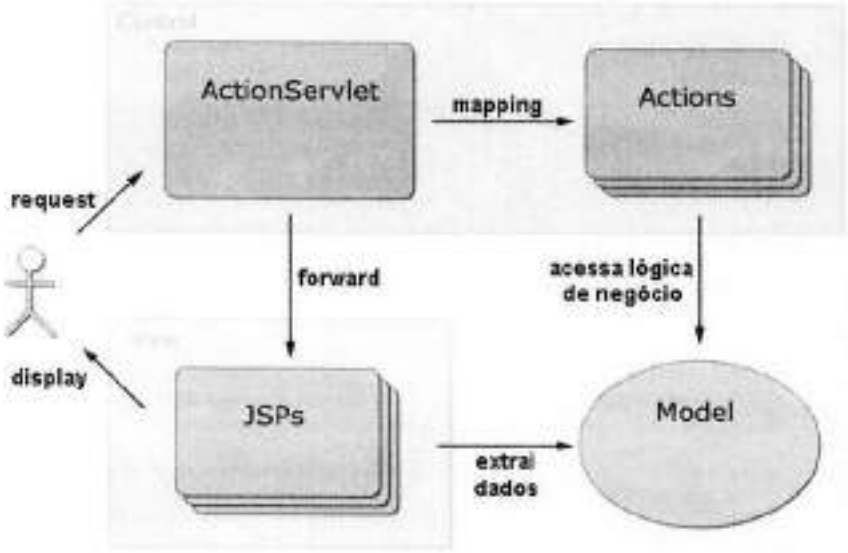


Figura 11 - Modelo MVC no Struts

Como complemento à figura 11 é apresentado, a seguir, um resumo do fluxo de ações iniciado por uma requisição do usuário.

O *ActionServlet*, que faz o controle centralizado, recebe a requisição e aciona a *action* apropriada para tratar a solicitação do usuário. Se a requisição contém dados de um formulário preenchido pelo usuário, o *controller* carrega esses dados no *action form* associado à *action*, antes de acioná-la.

A *action*, por sua vez (tendo recebido um *action form* ou não), acessa a camada de modelo, que contém a lógica de negócios, solicita a execução de uma tarefa e fica aguardando uma resposta. Os elementos de modelo contendo a resposta, normalmente *JavaBeans*, são retornados para a *action*, que então os armazena na requisição a ser encaminhada para a camada de visualização. Após especificar o destino da requisição contendo a resposta ao usuário, a *action* retorna o processamento ao *ActionServlet* para que este faça o encaminhamento.

Por fim, o elemento da camada de visualização (JSP) que recebeu a requisição, acessa os elementos de modelo que contêm a resposta, extrai os dados e os apresenta ao usuário de maneira formatada.

É importante notar, que as interações entre os componentes desse modelo MVC são configuradas nos arquivo struts-config.xml, mencionado anteriormente. Esse arquivo é responsável por boa parte da flexibilidade adquirida pelas aplicações que utilizam o *framework*, permitindo que alterações sejam feitas sem a necessidade de se modificar e recopilar o código. O anexo P apresenta um exemplo de configuração de uma *action* feito nesse arquivo.

6.3.1.2.2 *Camada de visualização*

Para camada de visualização, o Struts fornece um conjunto de *taglibs*[31] com muitas *custom tags* úteis ao desenvolvimento de JSPs.

---

[31] *Taglib* é a abreviação de *Tag Library*, que engloba um conjunto de *custom tags* logicamente relacionadas.

O Struts contém quatro bibliotecas de *tags* principais: *html*, *bean*, *logic* e *nested*.

A três bibliotecas que foram utilizadas na implementação do módulo de infra-estrutura são descritas a seguir:

- *html* → a biblioteca *html* está associada à URI http://jakarta.apache.org/struts/tags-html e pode ser acessada através do prefixo 'html'. Ela contém *tags* para *output* de HTML padrão, incluindo *forms*, *textfields*, *checkboxes*, *radio buttons*, dentre outros;
- *bean* → a biblioteca *bean* está associada à URI http://jakarta.apache.org/struts/tags-bean e pode ser acessada através do prefixo 'bean'. Ela contém *tags* para acesso a *JavaBeans* e suas propriedades. Pode ser usado também para definir novos *beans* e para configurar (*set*) propriedades;
- *logic* → a biblioteca *logic* está associada à URI http://jakarta.apache.org/struts/tags-logic e pode ser acessada através do prefixo 'html'. Ela contém *tags* para controle de fluxo, incluindo *tags* para geração de *output* condicional e iteração (criação de *loops*).

É importante mencionar que também foram utilizadas as *tags* da JSTL (*JavaServer Pages Standard Tag Library*) neste projeto, as quais foram priorizadas em relação às *tags* do Struts com funcionalidades semelhantes.

A JSTL é um padrão definido pelo JCP (*Java Community Process*) na JSR 52 (*Java Specification Request*). A implementação de referência dessa especificação, que foi utilizada no projeto, é disponibilizada pela ASF (*Apache Software Foundation*) sob o nome de *Standard Taglib*. A seguir é apresentada uma breve descrição das bibliotecas da JSTL:

- *core* → a biblitoteca *core* está associada à URI http://java.sun.com/jsp/jstl/core e pode ser acessada através do prefixo 'c'. Essa é a biblioteca principal da JSTL e foi mais utilizada na implementação do SARDA. Ela contém *tags* de propósito geral para realizar atividades como avaliação e *output* de expressões (<c:out>), configuração de variáveis

de escopo[32] e propriedades de *beans* (<c:set>), remoção de váriaveis de escopo (<c:remove>), tratamento de exceções (<c:catch>), execução condicional de uma ação (<c:if>), execução condicional de ações mutuamente exclusivas (<c:choose>, <c:when> e <c:otherwise>), iteração (<c:forEach>), *import* de recursos via URLs (<c:import>), construção de URLs com parâmetros (<c:url> e <c:param>), dentre outras;

- *format* → a biblitoteca *format* está associada à URI http://java.sun.com/jsp/jstl/fmt e pode ser acessada através do prefixo 'fmt'. Essa biblioteca contém *tags* para formatação de números, datas e horários, além de *tags* de suporte a internacionalização;
- *functions* → a biblitoteca *functions* está associada à URI http://java.sun.com/jsp/jstl/functions e pode ser acessada através do prefixo 'fn'. Essa biblioteca inclui *tags* que disponibilizam funções para diversos tipos de tarefas. Há funções para manipulação de objetos da classe *java.lang.Strings* (fn:toLowerCase, fn:toUpperCase, fn:substring, fn:replace, fn:trim, fn:split, fn:join, etc), para retornar o tamanho de um objeto que implementa a interface *java.util.Collection*, dentre outras;
- *xml* → a biblitoteca *xml* está associada à URI http://java.sun.com/jsp/jstl/xml e pode ser acessada através do prefixo 'x'. Essa biblioteca contém *tags* para manipulação de arquivos XML (*eXtensible Markup Language*);
- *sql* → a biblioteca *sql* está associada à URI http://java.sun.com/jsp/jstl/sql e pode ser acessada através do prefixo 'sql'. Essa biblioteca é utilizada para acesso à banco de dados e contém *tags* para executar consultas no banco de dados, acessar o resultado de consultas SQL, realizar atualizações no banco de dados e agrupar operações em um transação.

As três primeiras bibliotecas de tags da JSTL citadas anteriormente (*core*, *format* e *functions*) foram utilizadas, as demais foram descritas apenas por fins ilustrativos.

---

[32] Em uma aplicação *Web* em Java há quatro escopos: *application*, *session*, *request* e *page scope*.

Por fim, deve-se mencionar que a utilização de *taglibs* exige algumas pré-configurações. É necessário adicionar à aplicação os TLDs [33] (*Tag Library Descriptors*) de cada *taglib* e fazer o mapeamento desses TLDs no arquivo web.xml. Também é necessário a inclusão do JAR[34] contendo as classes[35] que implementam as *custom tags* dessa *taglib*. Antes de utilizar as *custom tags* em uma JSP, ainda é necessário declarar o uso da *taglib* que contém essas *tags*. Isso é feito através de uma diretiva, que de ser inclusa no início da JSP. O anexo Q apresenta um exemplo de utilização de uma *taglib*.

A primeira vista, o uso de *taglibs* pode parecer um tanto complicado. Porém, a utilização das *taglibs* do Struts e da JSTL é mais simples do que o descrito anteriormente. Tanto o Struts quanto a JSTL fornecem os recursos de suas *taglibs* (TLDs e classes das *tags*) compactados em arquivos .jar. Portanto, para o uso dessas *taglibs* basta adicionar-se os arquivos struts.jar (Struts), jstl.jar e standard.jar (JSTL) ao diretório lib da aplicação *Web*. Também não é necessário fazer nenhum mapeamento no arquivo web.xml, conquanto que se use as URIs e prefixos citados anteriormente.

6.3.2 Estrutura da Aplicação

A seguir são apresentadas a estrutura requerida por uma aplicação *Web* em Java, a estrutura de diretórios utilizada para organizar a aplicação no ambiente de desenvolvimento, e a estrutura de pacotes adotada para o código fonte da aplicação.

---

[33] *Tag Library Descriptors* são arquivos que usam a notação XML e contêm descrições das diversas *tags* de uma *taglib*. A descrição uma *tag* inclui, por exemplo, definições dos atributos dessa *tag*.

[34] Um *jar* (JavaArchive) é um arquivo compactado, tal como um *zip*, que contém os arquivos binários (*bytecodes* ou *.class*) de classes Java. O kit de desenvolvimento da SUN disponibiliza uma ferramenta de linha de comando, denominada JAR, para geração desse tipo de arquivo. Porém, um *jar* pode ser gerado usando qualquer ferramenta de compactação.

[35] Uma *custom tag* é implementada através de uma classe que estende *javax.servlet.jsp.tagext.TagSupport*, que faz parte das APIs do J2EE.

6.3.2.1 Estrutura Básica de uma Aplicação *Web*

Aplicações *Web* em Java devem seguir uma estrutura padrão para que possam ser instaladas em servidores (*containers*) que seguem as especificações J2EE definidas pelo JCP (Figura 12). Caso a aplicação não siga a estrutura definida, o servidor pode não reconhecê-la.

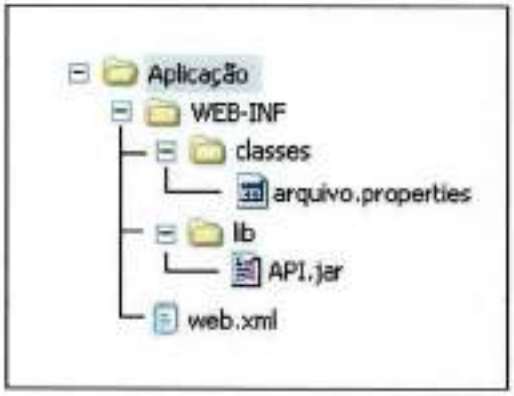


Figura 12 - Estrutura Padrão de uma Aplicação *Web*

Toda aplicação *Web* deve possuir um diretório chamado WEB-INF. Na raiz desse diretório deve ser colocado o *deployment descritor* da aplicação, que é um arquivo XML contendo configurações para o *deploy*[36] *da* aplicação. Neste arquivo, que é denominado web.xml, são feitas declarações e mapeamentos de componentes que fazem parte ou são utilizados pela aplicação, tais como *filters*, *listeners*, *servlets*, *taglibs* e *error pages*. Nesse arquivo também são feitas configurações de *login* e segurança (*security constraints*) para aplicações que utilizam os mecanismos de segurança declarativa do servidor. O anexo R apresenta o arquivo web.xml do SARDA.

Além do arquivo web.xml, que é obrigatório, o diretório WEB-INF pode conter alguns diretórios:

---

[36] *Deploy* é utilizado para designar o ato de se instalar uma aplicação em um servidor, de maneira que os recursos dessa aplicação fiquem disponíveis aos seus clientes.

- **classes** – esse diretório é utilizado para armazenar os *bytecodes* de classes e interfaces Java que fazem parte da aplicação. As classes devem ser armazenadas em uma estrutura de subdiretórios correspondente à estrutura de pacotes declarada nos arquivos dessas classes, ou seja, uma classe cujo *full qualified name* [37] é pacote.subpacote.NomeDaClasse, deve ser armazenada da seguinte maneira (Figura 13):

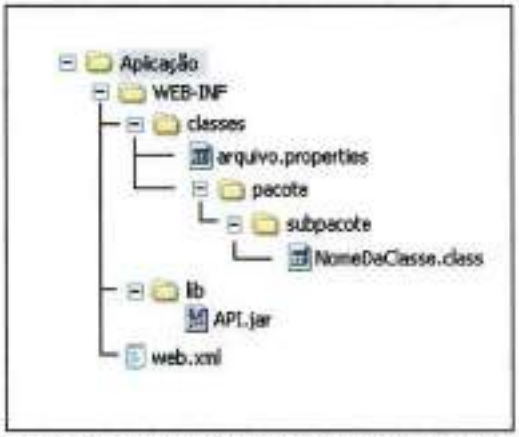


Figura 13 - Estrutura de Subdiretórios do Diretório Classes

O diretório *classes* também é utilizado para armazenar arquivos .properties da aplicação. Um exemplo desses arquivos é o ResourceBundle[38] do Struts, que é denominado MessageResources.properties.

- **lib** – esse diretório é utilizado para armazenar arquivos .jar contendo os *bytecodes* de classes e interfaces de APIs utilizadas pela aplicação. As classes que fazem parte da aplicação também podem ser armazenadas nessa pasta, sendo utilizadas como se fossem uma biblioteca externa. Mas pra isso, precisam ser empacotadas em um jar.

Os recursos *Web* da aplicação, ou seja, páginas HTML, JSPs, JavaScripts, CSS, imagens, dentre outros, devem ser colocados na raiz da aplicação. Esses recursos

---

[37] Nome completo contendo toda a estrutura de pacotes na qual a classe está inserida.

[38] ResourceBundles são arquivos do tipo chave-valor utilizados para internacionalização da aplicação

podem ser organizados em estruturas de diretórios conforme desejado, porém a estrutura de diretórios deve ser respeitada nas URLs de acesso a tais retais recursos.

6.3.2.2 Estrutura Adotada para Aplicação em Ambiente de Desenvolvimento

A estrutura adotada para a aplicação no ambiente de desenvolvimento não precisa ser a mesma usada quando se instala a aplicação no servidor. A maioria dos IDEs atuais possuem ferramentas para fazer o *deploy* da aplicação no servidor ou para empacotar a aplicação *Web* em um arquivo WAR[39].

O desenvolvimento do SARDA foi feito utilizando-se o IDE Eclipse e a aplicação foi estruturada da seguinte forma:

Figura 14 - Estrutura da Aplicação no IDE

[39] Um WAR (*WebARchive*) é um arquivo do tipo do JAR, porém contém a estrutura de diretórios obrigatória a uma aplicação *Web*, ou seja, é uma aplicação *Web* compactada.

Os principais diretórios são descritos a seguir:

- **src** – contém os códigos fontes da aplicação;
- **conf** – contém as bibliotecas e os arquivos de configuração utilizados pela aplicação;
- **docs** – contém arquivos com a documentação da aplicação, incluindo modelos UML;
- **web** – contém os arquivos *Web* da aplicação, incluindo CSS, JavaScripts, JSPs e imagens.

Observando-se a figura 14 também se pode perceber a presença de vários arquivos soltos na raiz da aplicação. Os arquivos com terminação .jar são APIs adicionadas ao projeto para compilação da aplicação. Também há os arquivos ant.properties e build.xml, que são utilizados pela ferramenta Ant, embutida no IDE. O Ant é uma ferramenta para execução de *cross-platform scripts* (*scripts* independentes de plataforma). As tarefas executadas pelo Ant são descritas no arquivo de *script* build.xml, que pode utilizar variáveis definidas em um outro arquivo (ant.properties).

No projeto SARDA o Ant foi utilizado para executar as seguintes tarefas:

- **compilação** – foi criada uma *task* (tarefa) denominada *compile* através da qual o Ant faz a compilação de todo o código fonte do projeto[40];
- ***build*** – o Ant é o responsável por traduzir a estrutura da aplicação em ambiente de desenvolvimento, para a estrutura aceita pelo *container Web*, ou seja, é ele quem monta a aplicação;
- ***deploy*** – o Ant também é utilizado para fazer o *deploy* da aplicação, transferindo os arquivos do diretório temporário utilizado para *build* para o diretório de execução do servidor (no utilizou-se o Tomcat).

---

[40] A palavra projeto é utilizada aqui para designar um projeto Java do ambiente de desenvolvimento.

Para utilizar o *script* (build.xml) desenvolvido para o SARDA em máquinas diferentes é necessário alterar-se apenas as váriaveis do arquivo ant.properties, que definem a localização do servidor e dos diretórios temporários utilizados.

O anexo S apresenta os arquivos build.xml e ant.properties utilizados no projeto.

6.3.2.3 Estrutura do Código Fonte

A figura 14 apresenta a estrutura de pacotes do código fonte do SARDA:

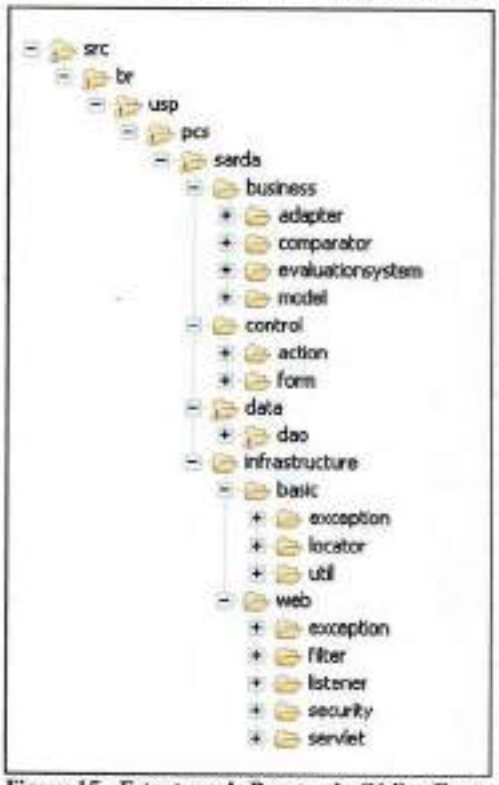


Figura 15 - Estrutura de Pacotes do Código Fonte

A hierarquia de pacotes começa com br.usp.pcs.sarda, sendo br o pacote raiz. O pacote sarda contém subpacotes que agrupam os componentes segundo as camadas da arquitetura da aplicação. Os subpacotes do sarda são:

- *business* – contém os componentes da camada de negócio, estruturando-os em diversos subpacotes. A camada de negócio engloba os *JavaBeans* do modelo (*model*) e os componentes que processam a lógica de negócio. Dentre os componentes de processamento, temos classes para ordenação de

JavaBeans do modelo segundo um dado atributo (*comparator*) e também as classes que constituem módulo de diagnóstico (*evaluationsystem*). A seguir, a figura 16 apresenta as classes do modelo:

Figura 16 - Classes do Modelo

- *control* – este pacote contém os componentes da camada de controle, que estão agrupados em dois grandes grupos: *actions* e *forms*. Tanto as actions quanto os *forms* são implementados utilizando-se classes fornecidas pelo *frameworks* Struts. As actions estão organizadas em pacotes segundo os componentes do modelo que são manipulados por elas. Veja a estrutura utilizada (figura 17) a seguir:

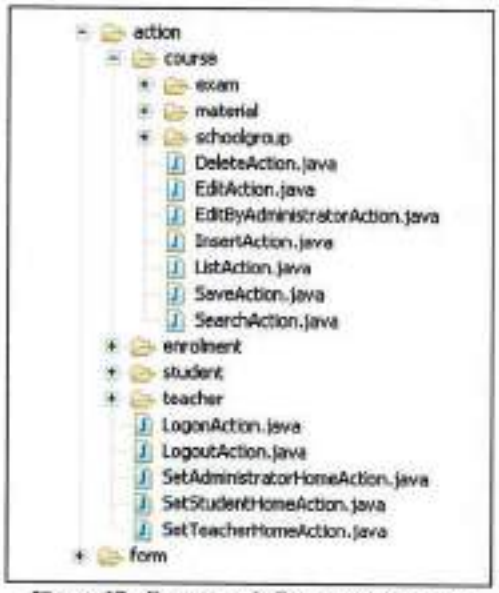


Figura 17 - Estrutura de Pacotes das Actions

- *data* – esta camada contém os componentes para acesso à camada de dados da aplicação. Os componentes dessa camada são denominados DAOs (*Data Access Objetcs*) e são responsáveis pelo acesso ao banco de dados. Esses componentes são detalhados no item X. O pacote *datai* também inclui classes para testes unitários dos DAOs. A figura 18 apresenta os componentes desse pacote:

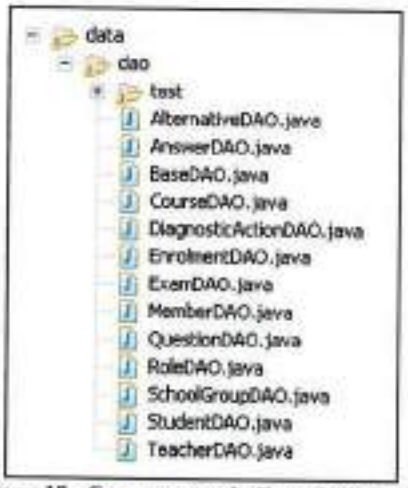


Figura 18 - Componentes da Camada de Dados

- *infrastructure* – este pacote contém diversos componentes de suporte à aplicação, que estão divididos em dois grandes subpacotes: *basic* e *web*. O subpacote *basic* contém componentes de suporte que podem ser usados em qualquer tipo de aplicação (*Web* ou *stand-alone*), incluindo classes de exceções, classes para geração de gráficos, classes para localização e conexão a um *DataSource*[41], além de classes utilitárias para realizar tarefas como *log*, formatação de datas e encriptação de dados. Já o subpacote *web*, contém classes de suporte a aplicações *Web*, incluindo *filters*, *listeners*, *servlets* e *exceptions*. A figura 19 apresenta a estrutura do pacote *infrastructure*:

[41] O único *DataSource* (fonte de dados) utilizado nesse projeto é um *pool* de conexões configurado no Tomcat para acesso ao banco de dados.

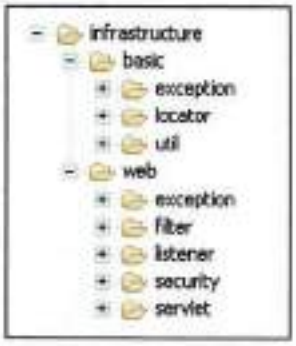


Figura 19 – Estrutura do Pacote *infrastructure*

6.3.3 Mecanismos de Segurança

O sistema desenvolvido utiliza segurança programática ao invés de utilizar segurança declarativa. Na segurança programática, os mecanismos de segurança são implementados em código, diferentemente da segurança declarativa, que deixa os mecanismos de segurança a cargo do servidor. A segurança declarativa utilizando Tomcat basea-se em *roles* (papéis) e *security constraints* que são declarados no arquivo web.xml. O Tomcat possui recursos de segurança bastante robustos e interessantes, porém optou-se pela segurança programática pela flexibilidade[42] que esta oferece.

Os mecanismos de segurança implementados envolvem autenticação do usuário e autorização de acesso a conteúdos. A auntenticação do usuário é feita pela classe br.usp.pcs.sarda.control.action.LogonAction e, é baseado na verificação do e-mail e da senha do usuário. A autorização é feita pela classe br.usp.pcs.sarda.infrastructure.web.UserAuthenticationFilter, que verifica o *role* do usuário e checa se ele tem permissão para acessar o conteúdo solicitado. Esta classe, como pode-se perceber pelo nome, é um filtro de pré-processamento, que intercepta todas as requisições feitas pelo usuário e faz a autorização e o *log* do acesso.

---

[42] Para usar a segurança declarativa também é necessário fazer algumas configurações no arquivo sever.xml do Tomcat, o qual não nem sempre é dispinibilizado por sites que hospedam a aplicações *Web* em Java.

6.3.4 Interface com o Usuário

A seguir são apresentados alguns exemplos de telas mostrando a aparência da IHC do SARDA. Estas telas seguem o projeto da IHC que foi detalhado na seção 5.1.4.

Figura 20 - Tela para *logon* no Sistema

Figura 21 - Ambiente do Estudante: Tela para Realização de exames

Figura 22 - Ambiente do Professor: Tela para Busca de Alunos

Figura 23 - Ambiente do Administrador: Tela para Busca de Professores

6.4 Módulo de Apoio ao Diagnóstico de Estudantes

A implementação do MADE foi feita em duas etapas:

1. Geração automática de código por ferramenta CASE
2. Implementação do algoritmo de avaliação

6.4.1 Geração automática de código

Considerando-se que a maior parte do trabalho envolvido na concepção da interface entre o módulo EAD e o MADE residiu na definição de uma interface extensível e genérica, temos que a implementação de tal interface consistiu basicamente na transposição da modelagem de classes feita na fase de projeto em código-fonte Java. Tal transposição foi feita através de geração automática de código.

Com o código gerado, a principal tarefa realizada foi comentar o código, identificando cada objeto e sua responsabilidade dentro das tarefas de interfaceamento EAD-MADE.

6.4.2 Implementação do algoritmo de avaliação

Durante a implementação do algoritmo de avaliação, percebeu-se a necessidade de algumas alterações na interface EAD-MADE, visando fornecer todas as informações necessárias para o algoritmo. O impacto de tal alteração no projeto foi muito pequeno, resultado de um projeto baseado em boas práticas.

O algoritmo proposto para a realização de diagnósticos de aprendizagem para exames respondidos por alunos é apresentado na forma de fluxograma: (Figura 14)

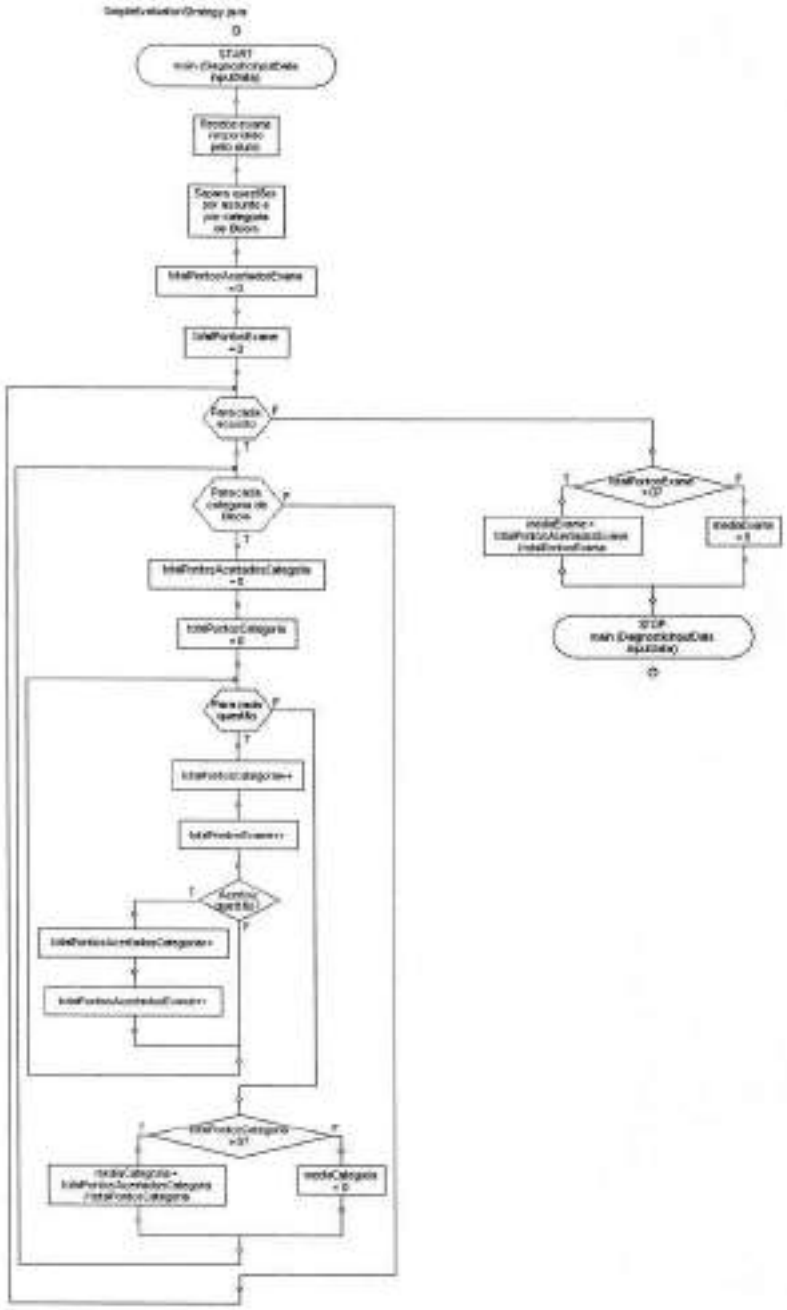


Figura 24 - fluxograma do algoritmo de realização de diagnósticos

Em poucas palavras, o algoritmo apresentado extrai a média ponderada de acertos de questões para todo o exame e também por assunto e categoria de Bloom. A cada

questão é associado um peso, que é a nota que o estudante receberá pela questão em caso de acerto. Também são associados a questão um tema e um objetivo pedagógico, abstraído sob a forma de categorias de Bloom (capítulo 2.3).

O algoritmo implementado neste trabalho é um algoritmo simples, porém demonstra a possibilidade de incorporação futura de algoritmos mais complexos. Maia (2004, p. 170) propõe métodos para a realização de diagnósticos de aprendizagem baseados em reforço diante de histórico do aproveitamento do aluno durante um dado curso. Tais métodos poderiam ser realizados a partir de algoritmos encapsulados em MADE e incorporados ao SARDA.

7 TESTES E AVALIAÇÃO

7.1 Estratégia de testes

O objetivo de qualquer estratégia de testes é encontrar a maior quantidade de erros quanto é possível em um curto espaço de tempo e com o mínimo de recursos. Para encontrar o maior número possível de erros, os testes devem ser conduzidos sistematicamente e casos de teste devem ser projetados usando técnicas disciplinadas.

Para tal, software é testado de duas perspectivas diferentes. A lógica interna do programa é exercitada usando-se técnicas de projetos de casos de teste denominados *testes de caixa branca*. Já requisitos de software são exercitados usando-se técnicas de projeto de casos de teste *caixa preta*.Em ambos os casos, o objetivo é encontrar o maior número de erros com a menor quantidade de esforço e tempo.

Em geral os primeiros testes planejados e executados concentram-se nos componentes individuais. À medida que o teste progride, o foco se desloca numa tentativa de encontrar erros em conjuntos integrados de componentes e finalmente em todo o sistema.

É importante frisar que a qualidade de software é um atributo intrínseco que depende fortemente do processo de obtenção de requisitos, análise, projeto e geração de código. Estratégias de teste podem detectar erros em sistemas para que estes possam ser eventualmente corrigidos, mas não conseguem combater os erros em sua origem.

7.2 Descrição dos testes realizados

Os principais tipos de testes realizados durante o ciclo de vida do SARDA foram:

Testes de caixa-preta:

- Testes de classes (testes unitários)
- Testes de casos de uso
- Testes de cenários de uso.

Testes de caixa branca

- Inspeção de código

Os tipos de teste realizados são descritos abaixo. O detalhamento de cada tipo de teste realizado é apresentado em seqüência.

Testes de classes (unitários):

A maioria dos testes de classes foi realizada através da criação e execução de *scripts* criados para tal finalidade. Um exemplo consiste no teste de objetos de acesso a dados (*Data Access Objects* – ou *DAOs*). Cada *DAO* encapsula operações básicas em objetos de banco de dados, como inserção, busca e exclusão de registros. Para cada *DAO* criou-se um script de teste correspondente, com a finalidade exclusiva de verificar as funcionalidades de cada operação.

Após a criação dos primeiros scripts de teste, percebeu-se que eles guardavam grande correlação e que a criação de um modelo de script de teste pouparia tempo e aumentaria a qualidade dos testes através da padronização de procedimentos.

Percebeu-se também que durante a realização de alterações estruturais no projeto – como, por exemplo, a alteração de tabelas no banco de dados – os testes precisaram ser refeitos, já que algumas operações possivelmente passariam a ter comportamentos indesejados.

Testes de casos de uso

Os testes de casos de uso foram baseados nas seqüências de ações contidas nos cartões de casos de uso. O teste consiste basicamente na execução da seqüência definida de eventos nas situações normais de operação e também em casos particulares (que em geral são extensões dos casos de uso).

Testes de cenário

O teste baseado em cenário concentra-se no que o usuário faz, e não no que produto faz. Isto significa detectar as tarefas que o usuário tem que realizar e depois aplicá-las, bem como suas variantes, aos testes (PRESSMAN, 2002).

Como o sistema possui mais de 60 casos de uso catalogados, a quantidade de cenários de teste pode ser muito grande. Para aumentar a eficiência dos testes optou-

se por priorizar a criação e execução de cenários de teste. Tal priorização foi embasada nos requisitos funcionais do software.

Inspeção de código

Testes de caixa branca foram realizados em sua maioria nos módulos de avaliação, uma vez que o sistema EAD é fortemente orientado a dados, e portanto não possui algoritmos complexos.

Os testes de caixa branca realizados no algoritmo de avaliação pedagógica consistem na simulação dos estados possíveis para variáveis e fluxo. Exemplo: o algoritmo possui um trecho no qual calcula uma divisão. Pensou-se então em situações hipotéticas nas quais houvesse uma divisão por zero. Estas situações foram testadas e descobriu-se que uma divisão por zero seria possível nos seguintes casos:

1. O exame aplicado ao aluno não possui nenhuma questão
2. O exame aplicado ao aluno possui apenas questões cujo peso (nota) é zero. Desta forma foi possível aumentar a estabilidade do algoritmo através da indução de erros.

7.3 Resultados

Os testes do SARDA foram realizados em cinco categorias principais:

1. EAD - Objetos de acesso a dados (DAOs)
2. Testes de casos de uso
3. EAD - Testes de interface (cenários de uso)
4. MADE – Algoritmo de auxílio à avaliação de aprendizagem
5. Testes em um cenário real de aplicação

7.3.1 Testes de objetos de acesso a dados

Para a realização dos testes de objetos de acesso a dados (*DAOs*) , duas estratégias principais foram planejadas:

1. Testes de caixa preta, os quais consistiram em *scripts* de teste;

2. Testes de caixa branca, baseados principalmente em inspeção de algoritmos.

Testes de caixa preta:

Todos os *DAOs* utilizados no sistema foram encapsulados em classes. Para cada *DAO* criado, uma classe de teste correspondente foi criada. As classes de teste foram criadas com base em uma classe de teste padrão. O objetivo das classes de teste foi a execução de *scripts* de avaliação de cada operação efetuada pelos *DAOs* (Tabela 2).

Tabela 2 - Procedimento padrão de teste para *DAOs*

Transação	Resultado esperado		Resultado final
	Antes	Depois	
1. Inserir registro	Console solicita dados a serem inseridos.	1 elemento inserido na tabela. Console exibe a mensagem "1 registro inserido"	(Preencher esta coluna com o resultado do teste)
2. Visualizar registro existente	Console solicita a identificação do registro desejado.	Retorna os dados do registro inserido na transação 1.	
3. Visualizar elemento não existente	Console solicita a identificação do registro desejado.	Console exibe a mensagem "registro solicitado não existe"	
4. Editar elemento existente	Console solicita dados de edição.	Console exibe a mensagem "1 registro editado"	
5. Editar elemento inexistente	Console solicita dados de edição.	Console exibe a mensagem "0 registros editados"	
6. Remover elemento existente	Console solicita a identificação do registro desejado.	Console exibe a mensagem "1 registro removido"	
7. Remover elemento inexistente	Console solicita a identificação do registro desejado.	Console exibe a mensagem "0 registros removidos"	
8. Inserir registro com chave-primária já existente *	Console solicita dados a serem inseridos.	Console exibe a mensagem "exceção encontrada: registro já existe"	
9. Remover registro com	Console solicita a identificação do registro desejado.	Console exibe a mensagem 'exceção	

dependênci a em outras tabelas **		encontrada: registro possui dependência na tabela <nome da tabela>"	

** Aplicável apenas quando a chave primária da tabela não for criada automaticamente*
*** Aplicável apenas quando o registro possuir dependências em outras tabelas. Repetir este teste para cada tabela dependente.*

Testes de caixa branca

Optou-se pela não execução dos testes "caixa branca" pelos seguintes motivos:

- As funções compartilhadas pelos objetos de acesso a dados (abrir e fechar conexão, executar comando) possuem baixa complexidade;
- As funcionalidades compartilhadas foram utilizadas muitas vezes em situações distintas, o que por si só já representa um teste;
- O tempo para execução de testes no projeto foi limitado. Optou-se pela execução de outros testes que pudessem localizar mais erros com menos tempo e recursos.

7.3.2 Testes de Casos de Uso

Os testes de casos de uso consistiram simplesmente da execução dos passos descritos em cada cartão de caso de uso (Anexo H). Também foram testadas as extensões, bem como as pré e pós-condições.

7.3.3 Testes de Interface Homem-Computador (IHC)

Este teste consiste no teste da interação do usuário com o sistema EaD, através da simulação de cenários típicos de uso do sistema. Tais cenários são composições dos casos de uso do sistema (Tabela 3).

Tabela 3 - Cenários de teste para o SARDA

Cenário	Ator	Descrição
1	Administrador	1. Autenticação 2. Cadastro de professor 3. Cadastro de aluno 4. Edição de dados pessoais 5. Sair do sistema
2	Professor	1. Autenticação 2. Criação de curso 3. Criação de turma 4. Matrícula de alunos. 5. Edição de dados pessoais 6. Sair do sistema
3	Professor	1. Autenticação 2. Cadastro de material para um curso 3. Criação de exame para um curso 4. Criação de questões para um exame 5. Criação de alternativas para uma questão 6. Sair do sistema
4	Aluno	1. Autenticação 2. Edição de dados pessoais 3. Execução de um exame 4. Visualização de resultado do exame 5. Visualização de material para o exame. 6. Sair do sistema.
5	Visitante	1. Autenticação com usuário inválido 2. Autenticação com usuário correto e senha inválida 3. Autenticação com usuário e senha corretos 4. Sair do sistema.

A seguir é apresentado um cenário de teste (Tabela 4). Os demais cenários de teste estão disponíveis no CD.

Tabela 4 – Descrição do Cenário 1

Ação	Resposta esperada	Resposta encontrada
Usuário acessa o sistema pelo endereço: http://www.rdeducacao.com.br/	Sistema exibe página solicitando usuário e senha.	(Preencher durante a execução dos testes)
Usuário digita usuário *rodrigo.maia@poli.usp.br* e senha *filev*.	Sistema exibe tela principal de sistema com boas-vindas.	
Usuário escolhe a opção *cadastro de professores*.	Sistema exibe tela de listagem de professores	

	cadastrados.	
Usuário escolhe a opção *cadastrar professor*.	Sistema exibe formulário de inserção de dados do professor.	
Usuário preenche dados do professor a ser inserido.	Sistema retorna mensagem *professor inserido com sucesso.*	
Usuário escolhe a opção *cadastro de alunos.*	Sistema exibe tela de listagem de alunos cadastrados.	
Usuário escolhe a opção *cadastrar aluno.*	Sistema exibe formulário de inserção de dados do aluno.	
Usuário preenche dados do aluno a ser inserido.	Sistema retorna mensagem *aluno inserido com sucesso.*	
Usuário escolhe a opção *editar dados pessoais*	Sistema exibe formulário de edição de dados pessoais.	
Usuário altera dados pessoais	Sistema exibe mensagem *dados pessoais editados com sucesso.*	
Usuário escolhe opção *sair do sistema*	Sistema exibe mensagem *Você saiu do sistema. Clique aqui para entrar novamente.*	

Tabela 5 - Descrição do cenário 5

Ação	Resposta esperada	Resposta encontrada
Usuário acessa o sistema pelo endereço: http://www.rdeducacao.com.br/	Sistema exibe página solicitando usuário e senha.	(Preencher durante a execução dos testes)
Usuário digita e-mail *inexistente@inexistente.com* e senha *inexistente*.	Sistema exibe mensagem *usuário ou senha inválidos.*	
Usuário digita e-mail *rodrigo.maia@poli.usp.br* e senha *inexistente*.	Sistema exibe mensagem *usuário ou senha inválidos.*	
Usuário digita e-mail *rodrigo.maia@poli.usp.br* e senha *filev*.	Sistema exibe a tela principal.	
Usuário escolhe opção *sair do sistema*	Sistema exibe mensagem *Você saiu do sistema. Clique aqui para entrar novamente.*	

7.3.4 Testes do algoritmo de avaliação de exames

O SARDA é por sua natureza um sistema centrado a dados, e não apresenta algoritmos complexos, com exceção do algoritmo de avaliação de exames.

A estratégia de teste para o algoritmo consiste em:

1. Inspeção do algoritmo e localização de possíveis pontos de falha
2. Listagem dos cenários em que tais falhas seriam possíveis e execução destes cenários
3. Planejamento do teste de estados de variáveis locais do algoritmo para cada passo, e comparação com os valores obtidos em simulação.

Os cenários de falha identificados para o algoritmo são:

- O objeto curso passado como parâmetro é nulo ou não contém exames
- O exame passado como parâmetro é nulo ou não contém questões
- As questões do exame são nulas ou não contém alternativas
- As questões do exame não possuem os parâmetros necessários para a execução do algoritmo (tema, categoria de Bloom e peso)
- As alternativas de cada questão são nulas.
- Os pesos de todas as questões são nulos (i.e: a soma de todos os pesos é igual a zero, o que ocasionaria divisão por zero) ou algumas questões possuem pesos negativos (o que também ocasionaria divisão por zero).

A partir de tais cenários, foram introduzidas verificações no algoritmo que lançam uma exceção. Após isto, testes foram realizados para a averiguação da eficácia de tais verificações.

7.3.5 Testes em um cenário real de aplicação

Finalmente, decidiu-se testar a aplicação em um cenário real. Para isto solicitou-se a colaboração de uma turma do curso de redes de uma faculdade de São Paulo. Os alunos desta turma foram cadastrados previamente no sistema. Uma data e horário

foram marcados para que os alunos respondessem a um teste sobre roteamento OSPF.

Na data e horário marcados, monitorou-se a execução do sistema, de modo a garantir que este teste transcorresse sem problemas. Ao final, as respostas da avaliação foram coletadas para análise. Trinta e um alunos responderam ao exame. Os resultados obtidos foram os seguintes:

Média: 3.29 de 8 = 41% de acerto

Mediana: 4 acertos

Moda: 3 acertos

O histograma de questões acertadas é apresentado a seguir (Figura 17).

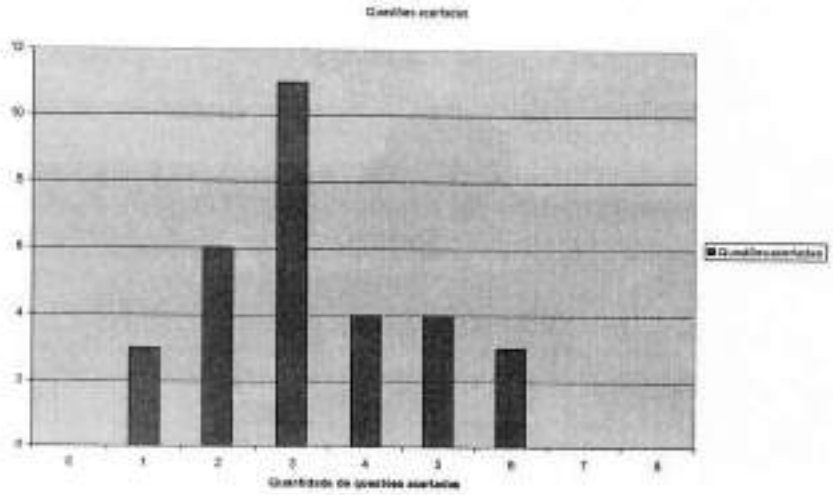


Figura 25 – Histograma de questões acertadas

Os dados obtidos também foram úteis para a obtenção de informações relativas ao número de acertos de cada questão. Nota-se que algumas questões tiveram poucos acertos (da ordem de 25%), enquanto algumas questões obtiveram percentuais bem acima da média (cerca de 75%).

Tais dados podem auxiliar o professor a reavaliar o seu método de ensino. No exemplo aplicado neste trabalho, a questão 6, sobre *Link State Advertisement* pode

conter alguma ambigüidade, trazer dificuldade de compreensão de seu enunciado, ou o material relacionado a este tema precisa ser revisto (Tabela 6).

Tabela 6 - Resultados do exame

Questão	Quantidade de respostas	Quantidade de acertos
6. Pode-se afirmar sobre LSA (Link State Advertisement):	31	8
8. Uma empresa possui 2 prédios, cada qual tendo uma conexão com a Internet. Em ambos os prédios, há servidores que se comunicam entre si. Assinale a alternativa mais apropriada:	31	9
2. O protocolo OSPF utiliza como métrica:	34	10
4. Em uma rede que opere com os protocolos RIP e OSPF (simultaneamente) é correto afirmar que:	32	10
3. Uma característica da área 0 é:	33	11
7. A "distância administrativa" é:	31	11
5. Considere uma rede administrada por uma única entidade (ex. banco A), e o qual decide subdividir o domínio de roteamento em diversas áreas. Assinale a alternativa que melhor justifica esta decisão:	30	18
1. O protocolo de roteamento OSPF utiliza o algoritmo:	35	25

Por fim, notou-se que algumas questões obtiveram mais respostas que outras, e que dois alunos conseguiram responder a mesma questão duas vezes. Este fato levantou a necessidade da aplicação de restrições de inserção de dados duplicados na tabela que guarda as respostas dos usuários, apesar de que a aplicação já verifica se um aluno está tentando responder um exame pela segunda vez.

8 CONCLUSÕES

8.1 Balanço do trabalho

Em termos gerais, o projeto apresentado neste documento pode ser considerado um sucesso. Os principais objetivos atingidos foram:

- O desenvolvimento de um ambiente de ensino à distância (EAD) simplificado, porém funcional;
- Um MADE desenvolvido e testado;
- Conhecimentos sobre diagnósticos de aprendizagem e ensino, devidamente catalogados neste documento;
- Novas possibilidades de continuidade dos trabalhos realizados neste projeto.

Enquanto os principais objetivos foram atingidos, alguns objetivos secundários mereceriam maior atenção, dentre eles:

- Desenvolvimento de MADEs mais complexos, envolvendo heurísticas e outras técnicas avançadas;
- Enfoque maior em tecnologias já existentes para ensino à distância (SCORM, LOM);
- Melhoria da interface com o usuário, incluindo aspectos de usabilidade;
- Mecanismos que facilitem a comunicação professor-aluno (fórum, chat, lista de tarefas);
- Testes e simulações.

8.2 Lições aprendidas

Em geral, as tarefas realizadas durante a execução do projeto de formatura foram bem sucedidas. Pode-se citar:

- Montagem da infra-estrutura de aplicação

- Atividades de desenvolvimento, especialmente no segundo semestre.
- Engenharia de software
- Documentação

Caso o projeto fosse repetido desde o início, as principais tarefas que seriam realizadas de forma diferente seriam:

- O tempo de desenvolvimento do módulo de EAD foi muito grande. Isto poderia ser evitado através da obtenção de estruturas de aplicação já prontas e ferramentas de geração automática de código.
- O tema do projeto foi definido muito tarde. Isto atrasou o início do projeto.

8.3 Trabalhos futuros

Como trabalhos futuros, pode-se apontar:

- Construção de mais MADEs:

  Com a incorporação de mais MADEs ao SARDA, seria possível avaliar a eficácia da interface de comunicação entre o EAD e os MADEs.

- Módulos auxiliares no EAD, para auxiliar na comunicação professor-aluno:

  Funcionalidades de comunicação em sistemas *web*, tais como serviço de mensagens instantâneas, vídeo e áudio-conferência, mural e fórum de mensagens, poderiam ser inclusas ao módulo EAD, propiciando o aumento da interatividade aluno-professor-sistema.

- Interface facilitada para o professor:

  Seria interessante que o professor, durante sua interação com o SARDA, pudesse especificar os objetivos educacionais de um dado curso através de interfaces simplificadas, nas quais a associação de materiais a questões e alternativas, a definição de quais aptidões são solicitadas ao aluno em um determinado teste e a obtenção de

relatórios pertinentes ao andamento do curso fossem realizadas diretamente pelo professor.

8.4 Comentário dos componentes do grupo

Os benefícios alcançados pelos integrantes do grupo foram valiosos; alguns deles talvez difíceis de expressar em um documento. No entanto citamos alguns dos benefícios os quais julgamos importantes para o nosso amadurecimento pessoal e profissional:

- O exercício de práticas de engenharia e gerenciamento de projetos de software;
- O respeito a opiniões divergentes;
- A aquisição de conhecimento sobre diversos temas, nas áreas de engenharia, pedagogia e ciência da computação;
- A prática da elaboração de documentos técnicos, muito útil tanto para engenheiros como para quaisquer outros profissionais;
- A importância do estabelecimento de metas e cumprimento de prazos;
- A aplicação de diversas disciplinas aprendidas durante a graduação, tais como engenharia de software, gerenciamento de projetos, bancos de dados, requisitos de sistemas computacionais e engenharia de informação;
- A crença na superação de limites, na importância da engenharia como atividade possibilitadora de melhorias para a sociedade e no potencial do ser humano.

Esperamos que o sistema proposto neste trabalho possa servir de base para o desenvolvimento de outros trabalhos visando o aperfeiçoamento do ensino a distancia baseado em sistemas *Web*. De nossa parte, estamos certos de que este projeto muito nos serviu no desenvolvimento técnico e pessoal, seja no estabelecimento e cumprimento de objetivos, seja na superação de obstáculos, no poder da comunicação e da vontade de servir.

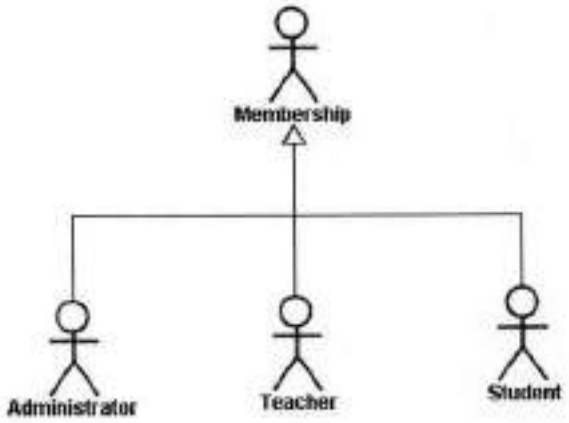
ANEXO A – Cronograma Inicial

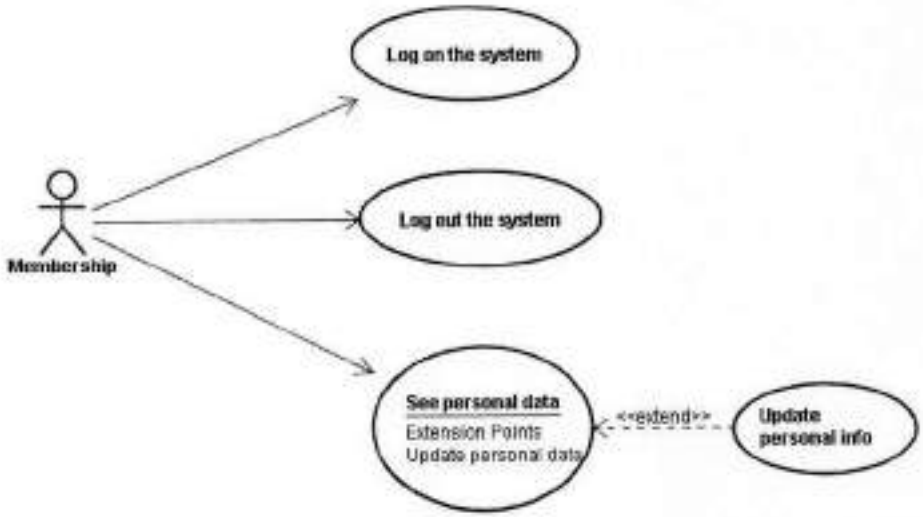
	Task Name	Duração	Início	Término	Predecessor
1	Estudo de padroes tecnologicos para EAD	5 dias	Qua 1/6/05	Ter 7/6/05	
2	Definicao de plataformas tecnologicas para desenvolvimento	5 dias	Qua 1/6/05	Ter 7/6/05	
3	**Engenharia de Software**	**45 dias**	**Qua 8/6/05**	**Ter 9/8/05**	1
4	**Requisitos de Software**	**10 dias**	**Qua 8/6/05**	**Ter 21/6/05**	
5	Analise de Casos de Uso (infra-estrutura)	1 dia	Qua 8/6/05	Qua 8/6/05	
6	Criacao do Cadoes de Uso	2 dias	Qui 9/6/05	Sex 10/6/05	5
7	Diagrama de Classes	2 dias	Seg 13/6/05	Ter 14/6/05	6
8	Analise de empacotamento (componentizacao)	5 dias	Qua 8/6/05	Ter 14/6/05	
9	Definicao das interfaces entre Infra-Estrutura e Modulos Pedagogicos	10 dias	Qua 8/6/05	Ter 21/6/05	
10	**Análise de Software**	**35 dias**	**Qua 22/6/05**	**Ter 9/8/05**	2;4
11	definicao de palavra	15 dias	Qua 22/6/05	Ter 12/7/05	
12	Analise de interface (desenho preliminar)	5 dias	Qua 13/7/05	Ter 19/7/05	11
13	Diagrama de Sequencia	10 dias	Qua 20/7/05	Ter 2/8/05	11;12
14	Grafico de Estados	5 dias	Qua 3/8/05	Ter 9/8/05	13
15	**Implementacao do Prototipo**	**40 dias**	**Qua 15/6/05**	**Ter 4/10/05**	10
16	Estabelecer ambiente de desenvolvimento	10 dias	Qua 15/6/05	Ter 28/6/05	
17	Instalacao do CVS	3 dias	Qua 15/6/05	Sex 17/6/05	
18	**Codificacao dos modulos da Infra-Estrutura**	**40 dias**	**Qua 15/6/05**	**Ter 4/10/05**	
19	Implementacao de cadastros e atividades adm	15 dias	Qua 15/6/05	Ter 5/7/05	
20	Validacao e Ajustes Eventuais	5 dias	Qua 31/8/05	Ter 6/9/05	19
21	Desenvolvimento de Interfaces	10 dias	Qua 15/6/05	Ter 28/6/05	
22	desenvolvimento das interfaces entre modulos	15 dias	Qua 7/9/05	Ter 27/9/05	20
23	Validacao e testes finais	5 dias	Qua 28/9/05	Ter 4/10/05	22
24	**Modulo Pedagogico**	**30 dias**	**Qua 5/10/05**	**Ter 1/11/05**	23
25	definicao de heuristicas (1)	5 dias	Qua 5/10/05	Ter 11/10/05	
26	Engenheira de Software	5 dias	Qua 12/10/05	Ter 18/10/05	25
27	Implementacao	10 dias	Qua 19/10/05	Ter 1/11/05	26
28	**Integracao e testes**	5 dias	Qua 2/11/05	Ter 8/11/05	27
29	**Documentacao final**	**35 dias**	**Qua 9/11/05**	**Ter 13/12/05**	28
30	Versao Preliminar	15 dias	Qua 9/11/05	Ter 29/11/05	
31	Revisao	5 dias	Qua 30/11/05	Ter 6/12/05	30
32	Entrega Final	5 dias	Qua 7/12/05	Ter 13/12/05	31

ANEXO B – Cronograma Revisado do Segundo Semestre

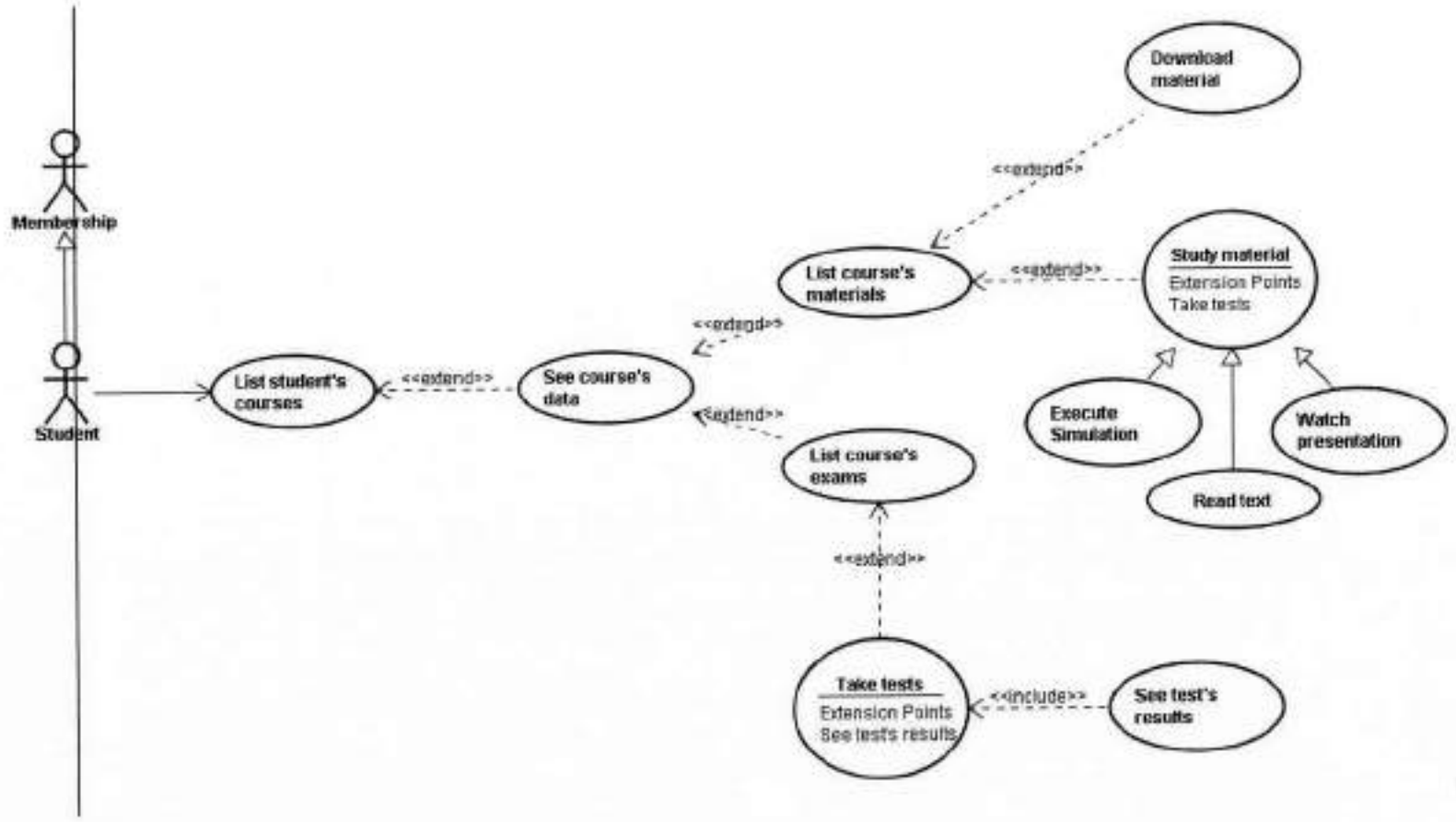
Atividade / Tarefa	Estimativa de conclusão
Plataforma *WEB* / Infra-estrutura	10/10/2005
Desenvolvimento do módulo de avaliação	30/10/2005
Avaliação dos resultados obtidos (comparação de desempenho de alunos, geração de relatórios).	05/11/2005
Draft da documentação final, para apreciação dos orientadores	22/11/2005

ANEXO C – Hierarquia dos Atores e Diagrama Casos de Uso do *Member*



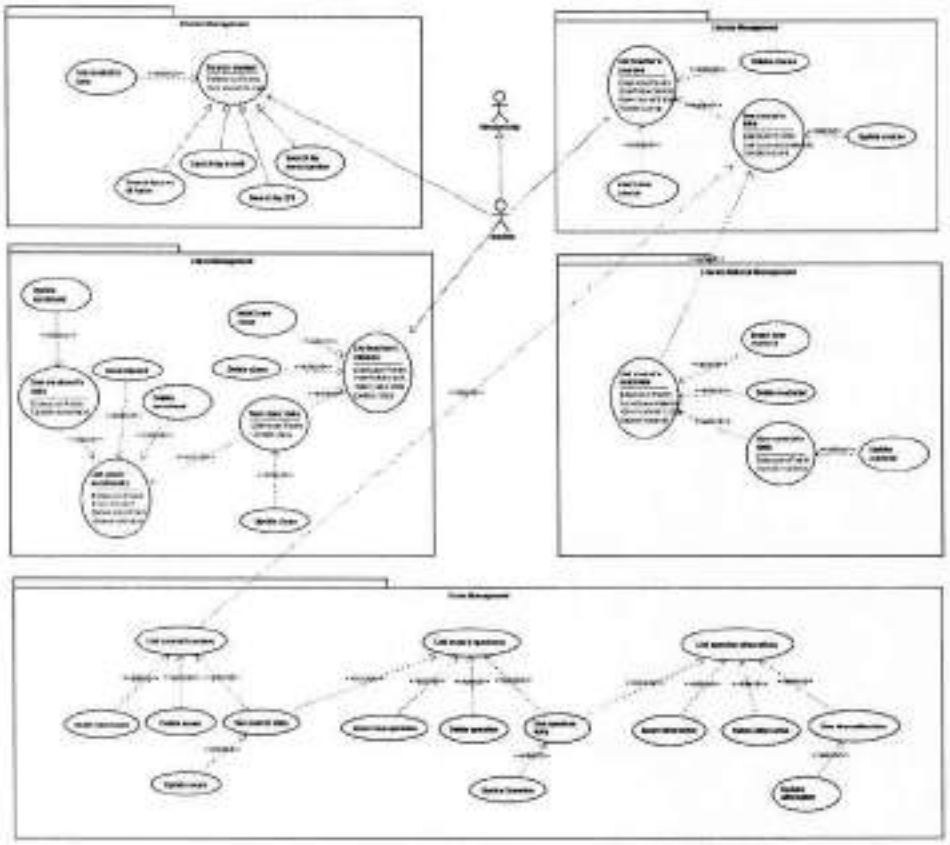


ANEXO D – Diagrama Casos de Uso do ator *Student*

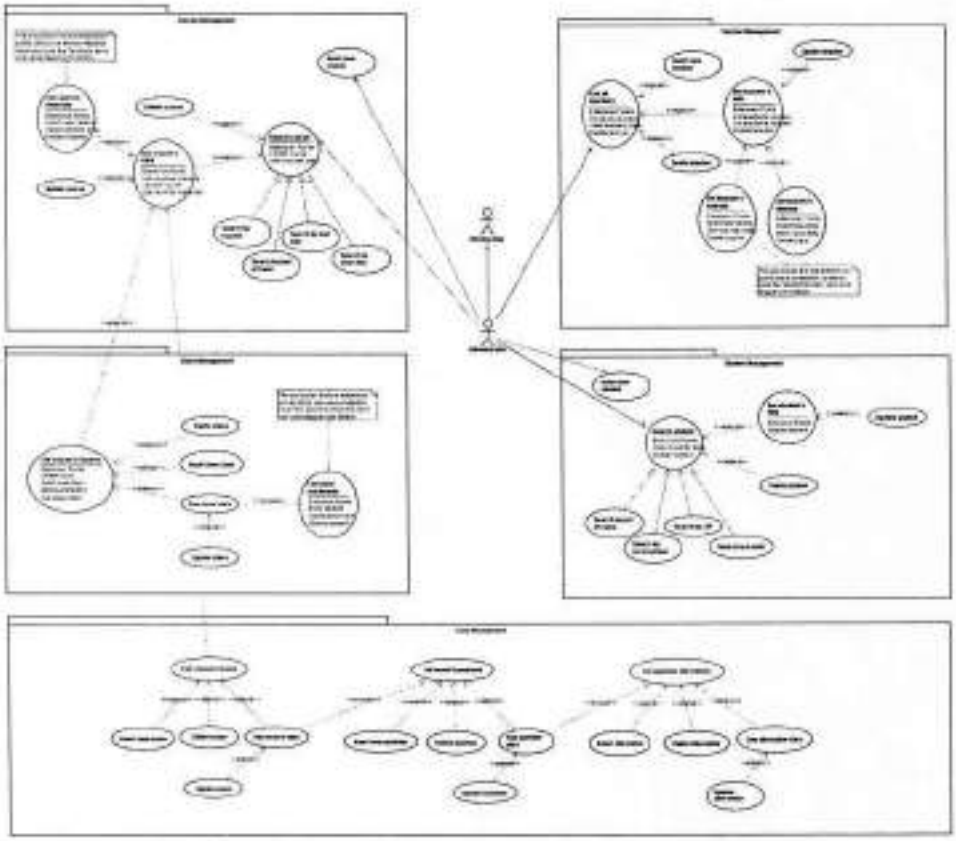


Excluido: ¶
¶

ANEXO E – Diagrama Casos de Uso do ator *Teacher*



ANEXO F – Diagrama de Casos de Uso do ator *Administrator*



ANEXO G – Tabela de Casos de Uso

Ator	Nome do caso de uso	Pacote
Visitante	*GC01: Log on the system*	Autenticação
Membro (usuário autenticado)	*ME01: Log out the system*	
	ME02: See personal data	Gerenciamento de dados pessoais
	ME03: Update personal data	
Estudante (*student*)	*ST01: See list of materials*	Gerenciamento de materiais
	ST02: Study Material	
	ST02a: Execute simulation	
	ST02b: Watch presentation	
	ST02c: Read text	
	ST03: Take test	Gerenciamento de exames
	ST04: View test results	
	ST05: View performance	
	ST06: See course info	Gerenciamento de cursos
	ST07: Contact teacher	
Administrador (*administrator*)	*AD01a: List teachers*	Gerenciamento de professores
	AD01b: View teacher data	
	AD01c: Insert teacher	
	AD01d: Update teacher data	
	AD01e: Delete teacher	
	AD02a: List teacher courses	Gerenciamento de cursos de professores
	AD02b: View teacher course data	
	AD02c: Insert teacher course	
	AD02d: Update teacher course data	
	AD02e: Delete teacher course	
	AD03a: List course classes	Gerenciamento de turmas
	AD03b: View class data	
	AD03c: Insert class	
	AD03d: Update class data	
	AD03e: Delete class	
	AD04a: Search students	Gerenciamento de alunos
	AD04b: View student data	
	AD04c: Insert student	
	AD04d: Update student data	
	AD04e: Delete student	
	AD05a: Search course	Gerenciamento de cursos
	AD05b: View course data	
	AD05c: Insert course	
	AD05d: Update course data	
	AD05e: Delete teacher course	
Professor (*teacher*)	*TE02b: View course data*	Gerenciamento de cursos
	TE02c: Insert course	
	TE02d: Update course data	
	TE02e: Delete course	
	TE03a: List course material	
	TE03b: View material	Gerenciamento de materiais
	TE03c: Insert material	
	TE03d: Update material data	
	TE03e: Delete material	
	TE04a: List teacher's classes	Gerenciamento de turmas
	TE04b: View teacher class	
	TE04c: Insert teacher class	
	TE04d: Update teacher class data	
	TE04e: Delete teacher class	
	TE04f: List class enrollments	Gerenciamento de matrículas
	TE04g: View enrollment	
	TE04h: Enroll student	
	TE04i: Update enrollment	

	TE04j: Delete enrollment	
	TE05a: List course exams	Gerenciamento de exames
	TE05b: View exam	
	TE05c: Edit exam	
	TE05d: Update exam	
	TE05e: Delete exam	
	TE06a: List questions	Gerenciamento de questões
	TE06b: View question	
	TE06c: Edit question	
	TE06d:Update question	
	TE06e: Delete question	
	TE07a: List alternatives	Gerenciamento de alternativas
	TE07b:View alternative	
	TE07c:Edit alternative	
	TE07d:Update alternative	
	TE07e:Delete alternative	

ANEXO H – Especificação dos Casos de Uso do ator *Teacher*

Caso de uso TE01a: *Search student**

Descrição: Este caso de uso descreve a busca por um aluno cadastrado.

Evento Iniciador: Ator solicita uma busca.

Atores: Professor (*Teacher*) e administrador (*Administrator*).

Pré-condição: Usuário autenticado como professor (*Teacher*) ou administrador (*Administrator*).

Seqüência de Eventos:

1. Sistema apresenta tela com opções dos vários tipos de buscas disponíveis (por nome, por e-mail, por número de matrícula ou por CPF), seguido de um campo para inserção da palavra-chave correspondente à busca a ser efetuada.
2. Ator seleciona o tipo de busca desejado e fornece a palavra-chave para a busca;
3. Sistema apresenta lista com alunos encontrados. Para cada aluno encontrado há uma linha correspondente na lista com um resumo dos dados do mesmo (ex: número de matrícula, nome, e-mail, etc) e um *link* para remoção deste aluno. Ao final da listagem, o sistema dispõe um botão para adição de um novo aluno.

Pós-Condição: Lista com alunos encontrados apresentada na tela ou mensagem de aluno não encontrado.

Extensões:

1. Se o usuário for um professor, as seguintes funcionalidades estarão alteradas: os botões de inclusão e exclusão ficarão desabilitados.
2. O sistema não encontra aluno correspondente ao tipo de busca e palavra-chave especificados e então exibe uma mensagem de aluno não encontrado na tela (passo 3).

Inclusões: Não há.

Caso de uso TE01b: View student data*

Descrição: Este caso de uso descreve a consulta aos dados de um aluno.

Evento Iniciador: Ator seleciona o aluno desejado.

Atores: Professor (*Teacher*) e administrador (*Administrator*).

Pré-condição: Exibição na tela da lista com alunos encontrados em uma busca efetuada previamente.

Seqüência de Eventos:
1. Ator escolhe aluno desejado (clicando em link ou botão disponibilizado).
2. Sistema exibe tela com dados detalhados do aluno selecionado, além das opções de voltar à página anterior (lista de alunos), salvar possíveis alterações feitas nos dados e excluir aluno;

Pós-Condição: Formulário com dados do aluno apresentado na tela.

Extensões:
1. Se o usuário for um professor, as seguintes funcionalidades estarão alteradas: os botões de salvar alteração e exclusão ficarão desabilitados.

Inclusões: Não há

Caso de uso TE02: Course Management

Caso de uso TE02: List teacher's courses*

Descrição: Este caso de uso lista todos os cursos cadastrados para um professor.

Evento Iniciador: Professor solicita exibição dos cursos.

Atores: Professor (*Teacher*).

Pré-condição: Usuário autenticado como professor.

Seqüência de Eventos:

1. Professor solicita a listagem de seus cursos.
2. Sistema apresenta lista de cursos encontrados. Para cada curso encontrado há uma linha correspondente na lista com um resumo dos dados do mesmo (ex: o nome, data de início do curso, data de término, etc) e um *link* para remoção do curso. Ao final da listagem, o sistema dispõe um botão para adição de um novo curso.

Pós-Condição: Lista com cursos encontrados apresentada na tela ou mensagem de curso não encontrado.

Extensões:

1. O sistema não encontra nenhum curso cadastrado e então exibe uma mensagem de curso não encontrado na tela (passo 3).

Inclusões: Não há

Caso de uso TE04e: Delete teacher's class*

Descrição: Este caso de uso descreve o processo de exclusão de uma turma pelo professor responsável.

Evento iniciador: Professor aciona a opção de remover turma.

Atores: Professor (*Teacher*).

Pré-condição: Sistema exibindo a tela de listagem de turmas associadas ao professor que está correntemente autenticado no sistema.

Seqüência de eventos:
1. Professor aciona a opção de remover turma.
2. Sistema remove turma solicitada.
3. Sistema exibe a tela com a lista atualizada das turmas do professor.

Pós-condição: Turma removida e lista atualizada das turmas do professor apresentada na tela.

Extensões:
1. Turma possui referências em outras partes do sistema (como por exemplo, matrículas associadas). Sistema exibe uma mensagem informando que a exclusão não foi possível e quais são as referências que ainda existem.

Inclusões: Não há

Caso de uso TE04f: List class' enrollments*

Descrição: Este caso de uso exibe a lista de alunos matriculados em uma determinada turma.

Evento iniciador: Professor solicita visualização dos dados de uma turma.

Atores: Professor (*Teacher*).

Pré-condição: Sistema exibindo lista de turmas do professor ou de um determinado curso.

Seqüência de eventos:

1. Professor solicita visualização dos dados de um turma.
2. Sistema dados da turma e listagem dos alunos matriculados na mesma. Para cada matrícula encontrada há uma linha correspondente na lista com um resumo dos dados da mesma (ex: código da matrícula, nome do aluno matriculado, nome da turma, data em que foi efetuada a matrícula, etc) e um *link* para remoção da matrícula. O sistema também dispõe de um botão para adição de uma nova matrícula.

Pós-condição: Sistema exibe tela com dados da turma e alunos matriculados na mesma.

Extensões:

1. Não existem alunos matriculados na turma: Sistema exibe uma mensagem informando que não há alunos matriculados.

Inclusões: Não há

Caso de uso TE04h: Enroll student*

Descrição: Este caso de uso descreve o processo de matrícula de um aluno em uma turma previamente cadastrada.

Evento iniciador: Professor solicita a inclusão de uma nova matrícula

Atores: Professor (*Teacher*).

Pré-condição: Sistema exibindo tela com informações da turma.

Seqüência de eventos:
1. Professor seleciona um aluno cadastrado no sistema e ainda não matriculado nesta turma.
2. Sistema processa e armazena os dados recebidos.
3. Sistema acrescenta a matrícula efetuada na lista de matrículas da turma e exclui o aluno matriculado das opções de alunos para novas matrículas.

Pós-condição: Matrícula cadastrada no sistema e tela com dados da turma e lista atualizada de matrículas sendo exibida.

Extensões:
1. Não há alunos cadastrados no sistema: é apresentada uma mensagem ao professor solicitando o cadastramento prévio dos alunos.

Inclusões: Não há

Caso de uso TE04j: Delete enrollment*

Descrição: Este caso de uso permite a exclusão de matrícula em uma turma.

Evento iniciador: Professor solicita de remoção de matrícula

Atores: Professor (*Teacher*).

Pré-condição: Sistema exibindo tela com informações de uma turma, incluindo lista de matrículas efetuadas para essa turma (deve existir pelo menos um elemento nessa lista, para que essa funcionalidade possa ser acessada).

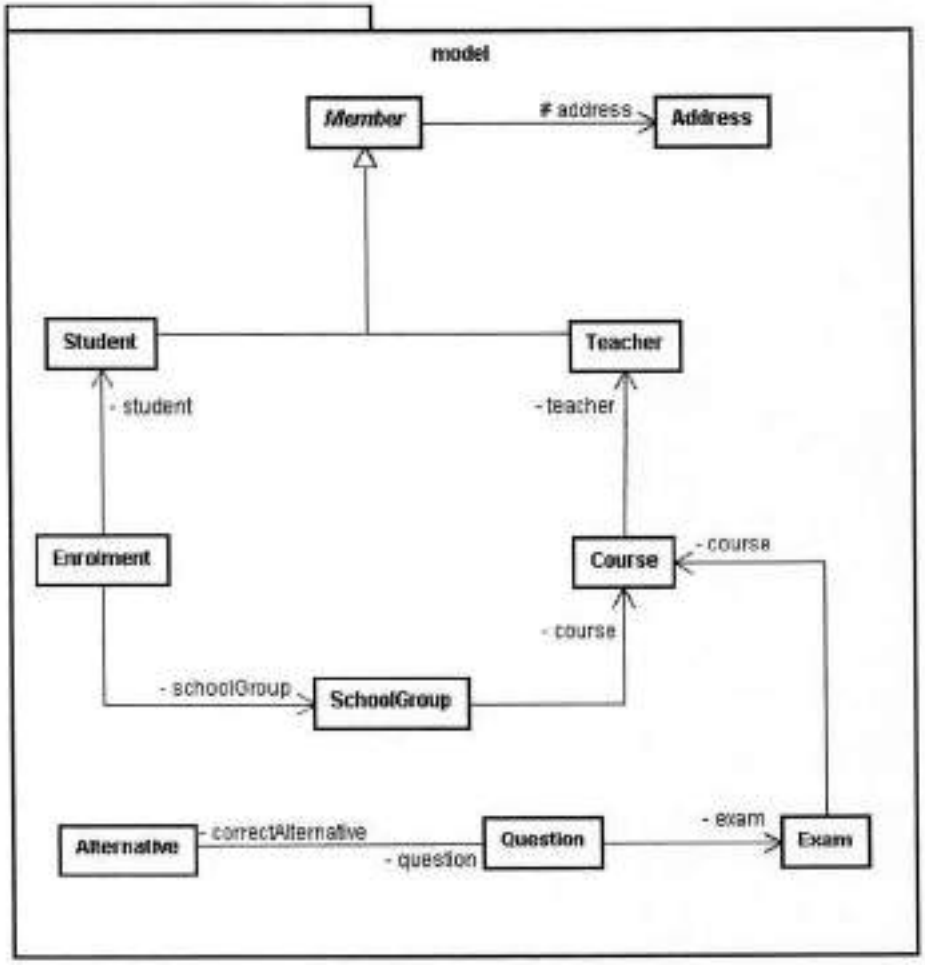
Seqüência de eventos:
1. Professor seleciona a opção de remover matrícula.
2. Sistema remove matrícula selecionada.
3. Sistema exibe a tela com informações da turma e a lista atualizada de matrículas da mesma.

Pós-condição: Matrícula removida e lista atualizada das matrículas da turma apresentada na tela.

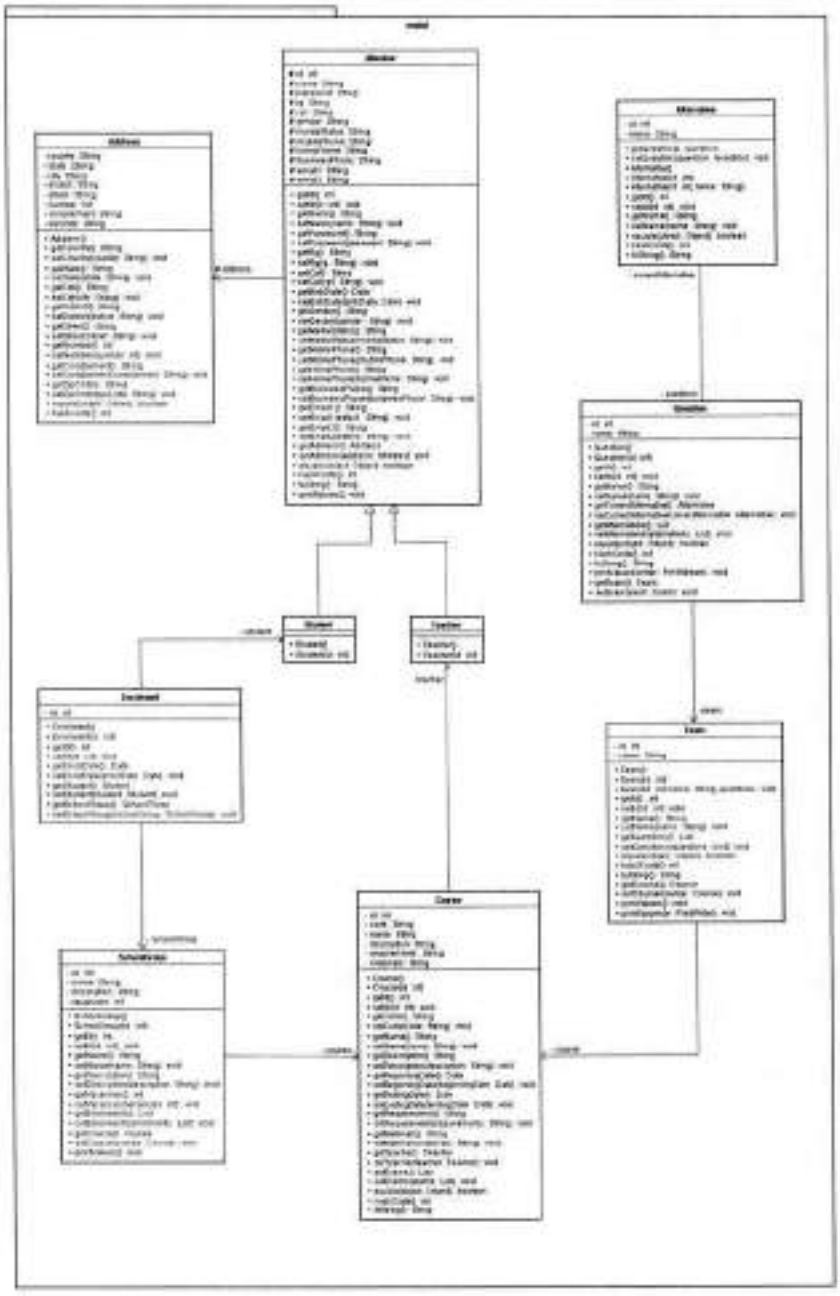
Extensões: Não há.

Inclusões: Não há.

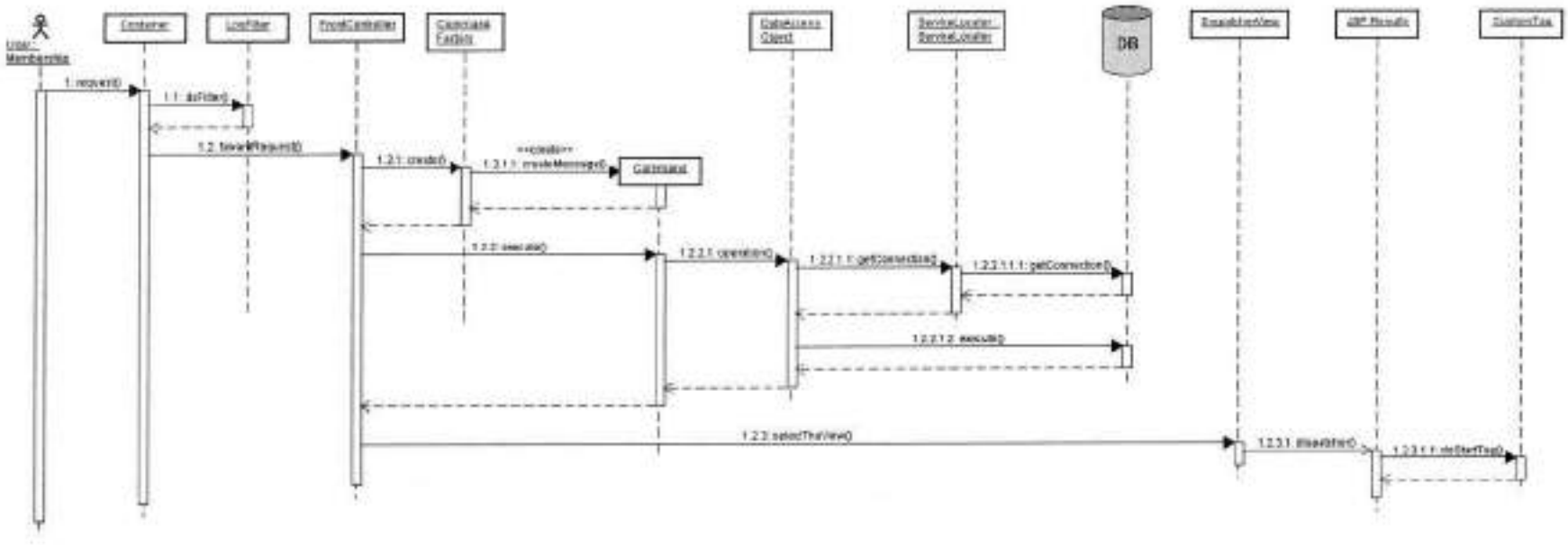
ANEXO I – Diagrama de Classes do Módulo de Infra-Estrutura (nível 0)



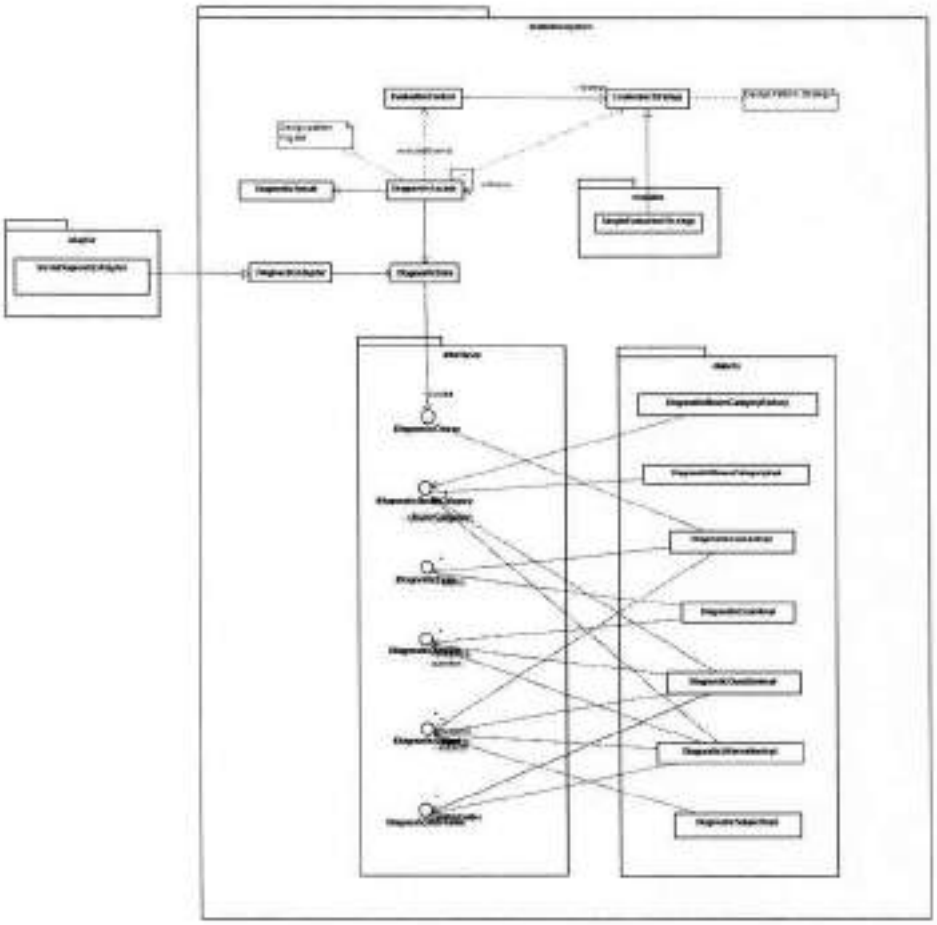
ANEXO J – Diagrama de Classes do Módulo de Infra-Estrutura (nível 1)



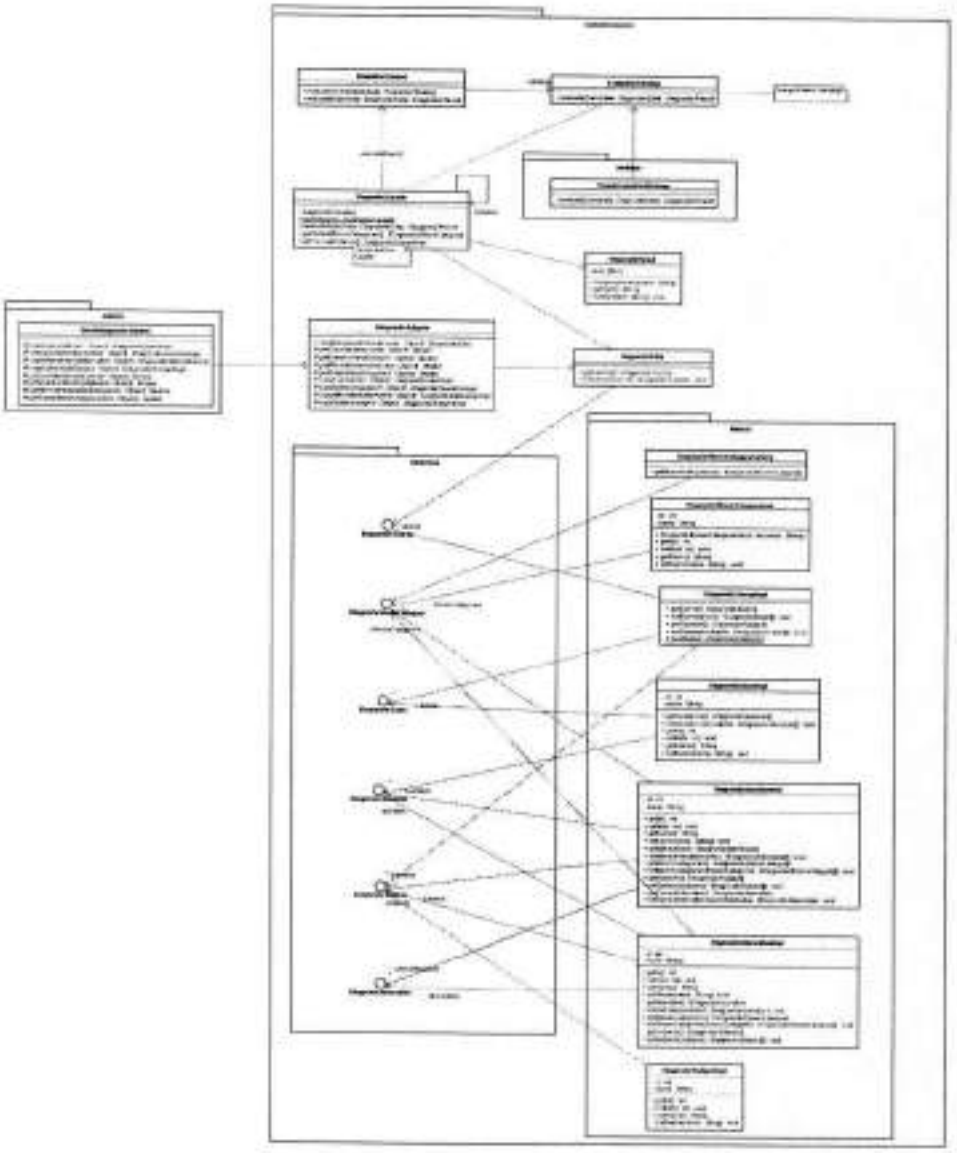
ANEXO K – Diagrama de Seqüência Ilustrativo da Arquitetura Adotada



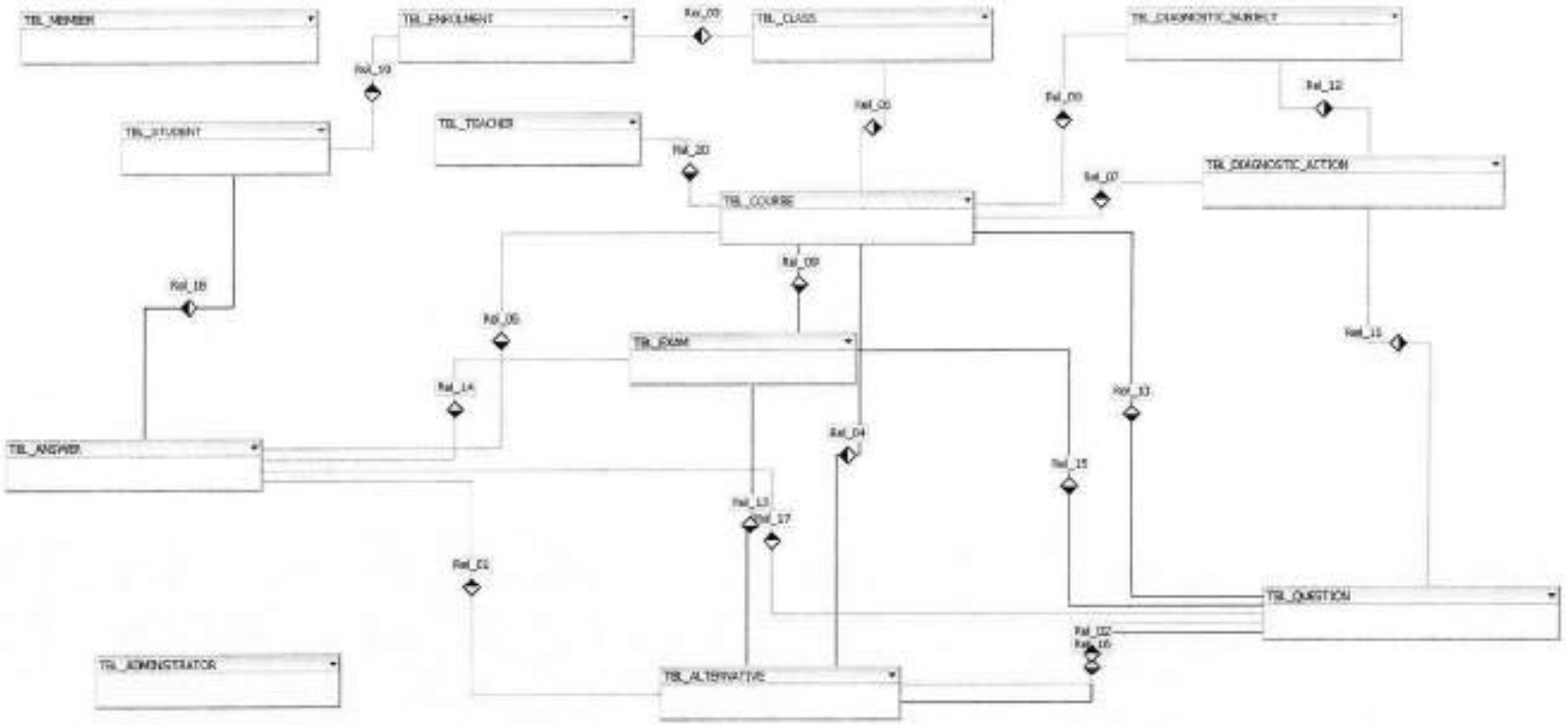
ANEXO L – Diagrama de Classes do Módulo de Infra-Estrutura (nível 0)



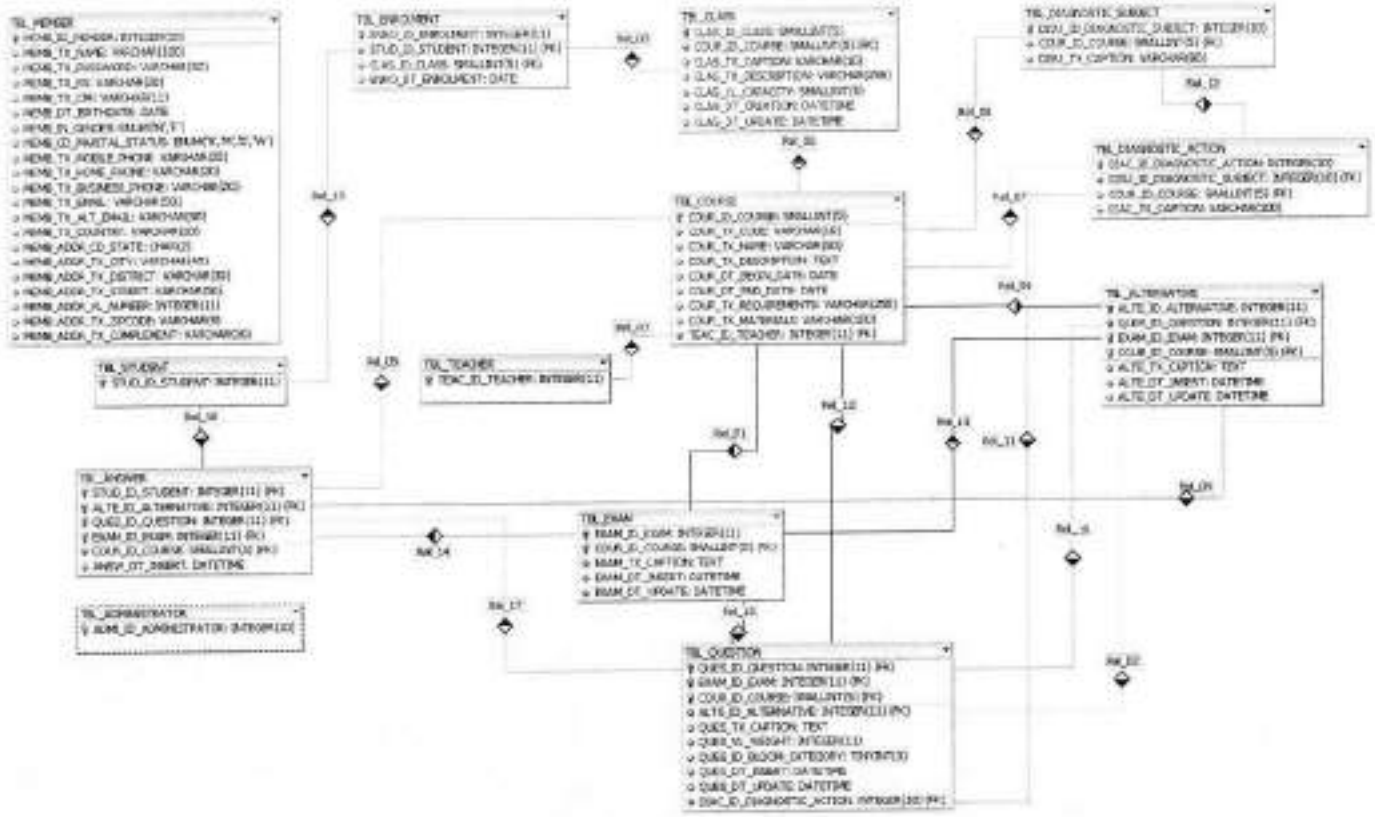
ANEXO M – Diagrama de Classes do Módulo de Infra-Estrutura (nivel 1)



ANEXO N – Modelo Entidade-Relacionamento – nivel 0



ANEXO O – Modelo Entidade-Relacionamento – nível 1



ANEXO P – Trecho do Arquivo struts-config.xml

```
<!-- Action Mapping Definitions -->
<action-mappings>
      <!-- Logon Action -->
      <action
            type="br.usp.pcs.sarda.action.LogonAction"
            path="/logon"
            name="logonForm"
            scope="request"
            input="/logonTable.jsp">

            <forward
                  name="student"
                  path="/private/student/home.do"/>

            <forward
                  name="teacher"
                  path="/private/teacher/home.do"/>

            <forward
                  name="admin"
                  path="/private/administrator/home.do"/>

            <forward
                  name="failure"
                  path="/logonTable.jsp"/>
      </action>
         ...

      <action>
         ...
      </action>
<action-mappings>
```

Esse código, que foi extraído do arquivo struts-config.xml do SARDA, mostra a configuração de uma *action*. No caso, trata-se da *action* responsável pelo *logon* no sistema. O atributo *path*, da *tag* <*action*>, associa a URL */logon* à classe *br.usp.pcs.sarda.action.LogonAction*, ou seja, quando o usuário solicita essa URL, o *ActionServlet* aciona a classe mencionada. O atributo *name*, dessa mesma *tag*, associa um *action form* à *action* em questão e, o atributo *input*, especifica a página que contém o formulário utilizado para preencher o *action form*. As *tags* <*forward*>, representam possíveis encaminhamentos que podem ser feitos a partir dessa *action*, associando um nome a uma URL. A *action* faz referência a esses nomes em seu código e, dessa maneira, para modificar o destino do encaminhamento, basta alterar o mapeamento feito neste arquivo, sem que seja necessário alterar o código.

ANEXO Q – Utilização de uma *TagLib*

```
<%@ taglib
      uri="http://java.sun.com/jsp/jstl/core"
      prefix="c"%>
<%@ taglib
      uri="http://jakarta.apache.org/struts/tags-html"
      prefix="html"%>

<table border="0" width="100%" class="tabelamenu">
<tr>
      <td>
            <c:url value="/private/teacher/home.do" var="url"/>
            <html:link href="${url}">Inicio</html:link>
      </td>
...
```

O código apresentado é um trecho da página private/teacher/menu.jsp do SARDA. Esta página declara o uso de duas *taglibs* (uma da JSTL e uma do Struts) através de diretivas <@ taglib>. Essas diretivas tem que ser colocadas no começo do arquivo e devem especificar a URI e o prefixo para utilização de cada *taglib*.

Essa página exemplifica o uso da *tags* <c:url> e <html:link>. A primeira pertence à biblioteca *core* da JSTL e é usada pra criar e atribuir uma URL à uma variável. A segunda faz parte da biblioteca *html* e tem a mesma finalidade da *tag* <a href>, que é padrão do HTML e serve para criar *links* para recursos especificados através de URLs. É interessante notar que a variável (url) criada pela *tag* da JSTL é utilizada pela *tag* do Struts.

ANEXO R – Arquivo web.xml do SARDA

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
	version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
	http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
	<description>This is the Sarda Web Application</description>
	<display-name>SARDA</display-name>

	<jsp-config>
		<jsp-property-group>
			<url-pattern>*.jsp</url-pattern>
			<scripting-invalid>false</scripting-invalid>
			<el-ignored>false</el-ignored>
			<page-encoding>ISO-8859-1</page-encoding>
			<include-prelude>/forcerefresh.jspf</include-prelude>
		</jsp-property-group>
	</jsp-config>

	<!-- Database Configuration Parameters -->
	<context-param>
		<description>
			Flag to set the use of DB Data Source
		</description>
		<param-name>usingDataSource</param-name>
		<param-value>false</param-value>
	</context-param>
	<context-param>
		<description>Data Base Pooling</description>
		<param-name>dbPoolingName</param-name>
		<param-value>SARDA-Pooling1</param-value>
	</context-param>
	<context-param>
		<description>Data Base URL</description>
		<param-name>dbUrl</param-name>
		<param-value>
			jdbc:mysql://mysql.rdeducacao.com.br:3306/rdeducacao1
		</param-value>
	</context-param>
	<context-param>
		<description>Data Base Driver</description>
		<param-name>dbDriver</param-name>
		<param-value>com.mysql.jdbc.Driver</param-value>
	</context-param>
	<context-param>
		<description>Data Base User</description>
		<param-name>dbUser</param-name>
		<param-value>rdeducacao1</param-value>
	</context-param>
	<context-param>
		<description>Data Base Password</description>
		<param-name>dbPassword</param-name>
		<param-value>s4rd4mySql</param-value>
	</context-param>

	<!-- Filter Responsible to Make the User Authentication -->
	<filter>
		<filter-name>UserAuthenticationFilter</filter-name>
		<filter-class>
			br.usp.pcs.sarda.web.filter.UserAuthenticationFilter
		</filter-class>
	</filter>
	<filter-mapping>
		<filter-name>UserAuthenticationFilter</filter-name>
```

```
        <url-pattern>/private/*</url-pattern>
</filter-mapping>

<!-- Context Listener Responsible to Set the Database Drivers -->
<listener>
        <listener-class>
                br.usp.pcs.sarda.web.listener.ApplicationListener
        </listener-class>
</listener>

<!-- Cewolf Servlet Configuration (with debugging) -->
<servlet>
        <servlet-name>cewolf</servlet-name>
        <servlet-class>de.laures.cewolf.CewolfRenderer</servlet-class>
        <init-param>
                <param-name>storage</param-name>
                <param-value>
                        de.laures.cewolf.storage.TransientSessionStorage
                </param-value>
        </init-param>
        <init-param>
                <param-name>overliburl</param-name>
                <param-value>js/overlib.js</param-value>
        </init-param>
        <init-param>
                <param-name>debug</param-name>
                <param-value>true</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
</servlet>

<!-- Cewolf Servlet Mapping -->
<servlet-mapping>
        <servlet-name>cewolf</servlet-name>
        <url-pattern>/cewolf/*</url-pattern>
</servlet-mapping>

<!-- Standard Action Servlet Configuration (with debugging) -->
<servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>
                org.apache.struts.action.ActionServlet
        </servlet-class>
        <init-param>
                <param-name>config</param-name>
                <param-value>/WEB-INF/struts-config.xml</param-value>
        </init-param>
        <init-param>
                <param-name>debug</param-name>
                <param-value>2</param-value>
        </init-param>
        <init-param>
                <param-name>detail</param-name>
                <param-value>2</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
</servlet>

<!-- Standard Action Servlet Mapping -->
<servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
</servlet-mapping>

<!-- Download Servlet Configuration -->
<servlet>
```

```
        <servlet-name>download</servlet-name>
        <servlet-class>
            br.usp.pcs.sarda.web.servlet.DownloadServlet
        </servlet-class>
    </servlet>

    <!-- Download Servlet Mapping -->
    <servlet-mapping>
        <servlet-name>download</servlet-name>
        <url-pattern>
            /private/student/course/material/download
        </url-pattern>
        </servlet-mapping>

    <!-- Set the number of minutes after which the sessions expire -->
    <session-config>
        <session-timeout>15</session-timeout>
    </session-config>

    <!-- The Usual Welcome File List -->
    <welcome-file-list>
        <welcome-file>private/welcome.jsp</welcome-file>
    </welcome-file-list>

    <!-- Error Pages Mapping -->
    <error-page>
        <error-code>404</error-code>
        <location>/errorPage.jsp</location>
    </error-page>
    <error-page>
        <exception-type>
            br.usp.pcs.sarda.basic.exception.SardaException
        </exception-type>
        <location>/errorPage.jsp</location>
    </error-page>
    <error-page>
        <exception-type>
            br.usp.pcs.sarda.web.exception.SardaWebException
        </exception-type>
        <location>/errorPage.jsp</location>
    </error-page>
</web-app>
```

ANEXO S – Arquivos Utilizados pelo Ant: build.xml e ant.properties

BUID.XML

```
<project name="sarda" default="deploy-hotdeploy-localhost" basedir=".">

	<!-- VARIÁVEIS E PREPARAÇÃO DO AMBIENTE -->
	<property file="ant.properties"/>

	<!-- REMOÇÃO DOS DIRETÓRIOS TEMPORÁRIOS -->
	<target name="base-clean">
		<delete dir="${compile.dir}"/>
		<delete dir="${jars.dir}"/>
	</target>

	<!-- CRIAÇÃO DOS DIRETÓRIOS TEMPORÁRIOS -->
	<target name="base-prepare">
		<mkdir  dir="${compile.dir}"/>
		<mkdir  dir="${jars.dir}"/>
		<mkdir  dir="${compile.dir}/classes"/>
		<mkdir  dir="${compile.dir}/war"/>
	</target>

	<!-- COMPILAÇÃO DOS ARQUIVOS FONTES -->
	<target name="base-compile" depends="base-prepare">
		<javac
			srcdir="src"
			destdir="${compile.dir}/classes"
			debug="on" classpath="${server.lib}/servlet-api.jar"/>
			<copy todir="${compile.dir}/classes">
				<fileset
					dir="conf/properties"
					includes="**/*.properties"/>
			</copy>
	</target>

	<!-- GERAÇÃO DO JAR DA APLICAÇÃO -->
	<target name="create-jar-desenv" depends="base-compile">
	    <jar jarfile="${jars.dir}/${app.name}.jar">
	      <fileset dir="${compile.dir}/classes"></fileset>
	    </jar>
	</target>

	<!-- CONSTRUÇÃO DA ARQUITETURA DA APLICAÇÃO-->
	<target
		name="base-build"
		depends="base-prepare, base-compile, create-jar-desenv">
		<copy todir="${compile.dir}/war/js">
			<fileset dir="web/js"/>
		</copy>
		<copy todir="${compile.dir}/war/css">
			<fileset dir="web/css"/>
		</copy>
		<copy todir="${compile.dir}/war/images">
			<fileset dir="web/images"/>
		</copy>
		<copy todir="${compile.dir}/war">
			<fileset dir="web/pages"/>
		</copy>
		<copy todir="${compile.dir}/war/META-INF">
			<fileset dir="conf/META-INF"/>
		</copy>
```

```
            <copy todir="${compile.dir}/war/WEB-INF">
                  <fileset dir="conf/WEB-INF"/>
            </copy>

            <copy
                  file="${jars.dir}/${app.name}.jar"
                  tofile="${compile.dir}/war/WEB-INF/lib/${app.name}.jar">
            </copy>
      </target>

      <!-- DEPLOY DA APLICAÇÃO NO SERVIDOR -->
      <target name="deploy-hotdeploy-localhost" depends="base-build">
            <copy todir="${server.localhost}/${app.name}">
                  <fileset dir="${compile.dir}/war"/>
            </copy>
      </target>

      <!-- DEPLOY DO WAR DA APLICAÇÃO NO SERVIDOR -->
      <target name="deploy-hotdeploy-localhost-war" depends="base-build">
          <jar jarfile="${compile.dir}/war/${app.name}.war">
                  <fileset dir="${compile.dir}/war"></fileset>
          </jar>
            <move
                  file="${compile.dir}/war/${app.name}.war"
                  todir="${server.localhost}">
            </move>
      </target>

      <!-- UNDEPLOY DA APLICAÇÃO NO SERVIDOR -->
      <target name="env-undeploy-localhost" depends="">
            <delete dir="${server.localhost}/${app.name}"/>
            <antcall target="base-clean"/>
      </target>

</project>
```

ANT.PROPERTIES

```
app.name = sarda
compile.dir = C:/temp/sarda/
jars.dir = ${compile.dir}/jars

tomcat.home = C:/java/jakarta-tomcat-5.0.28
tomcat.lib = ${tomcat.home}/common/lib
tomcat.sharedlib = ${tomcat.home}/shared/lib
tomcat.localhost = ${tomcat.home}/webapps

server.lib = ${tomcat.lib}
server.sharedlib = ${tomcat.sharedlib}
server.localhost = ${tomcat.localhost}
```

APÊNDICE I – Resumo da Biografia de Piaget e Vygotsky

Jean Piaget (1896-1980)

Um pioneiro no estudo da inteligência infantil

Jean Piaget (1896-1980) foi um renomado psicólogo e filósofo suiço, conhecido por seu trabalho pioneiro no campo da inteligência infantil. Piaget passou grande parte de sua carreira profissional interagindo com crianças e estudando seu processo de raciocínio. Seus estudos tiveram um grande impacto sobre os campos da Psicologia e Pedagogia.

Sua Vida

Jean Piaget nasceu no dia 9 de agosto de 1896, em Neuchâtel, na Suíça. Seu pai, um calvinista convicto, era professor universitário de Literatura medieval.

Piaget foi um menino prodígio. Interessou-se por História Natural ainda em sua infância. Aos 11 anos de idade, publicou seu primeiro trabalho sobre sua observação de um pardal albino. Esse breve estudo é considerado o início de sua brilhante carreira científica. Aos sábados, Piaget trabalhava gratuitamente no Museu de História Natural.

Piaget freqüentou a Universidade de Neuchâtel, onde estudou Biologia e Filosofia. Ele recebeu seu doutorado em Biologia em 1918, aos 22 anos de idade.

Após formar-se, Piaget foi para Zurique, onde trabalhou como psicólogo experimental. Lá ele freqüentou aulas lecionadas por Jung e trabalhou como psiquiatra em uma clínica. Essas experiências influenciaram-no em seu trabalho. Ele

passou a combinar a psicologia experimental - que é um estado formal e sistemático - com métodos informais de psicologia: entrevistas, conversas e análises de pacientes.

Em 1919, Piaget mudou-se para a França, onde foi convidado a trabalhar no laboratório de Alfred Binet, um famoso psicólogo infantil que desenvolveu testes de inteligência padronizados para crianças. Piaget notou que crianças francesas da mesma faixa etária cometiam erros semelhantes nesses testes e concluiu que o pensamento lógico se desenvolve gradualmente.

O ano de 1919 foi um marco em sua vida. Piaget iniciou seus estudos experimentais sobre a mente humana e começou a pesquisar também sobre o desenvolvimento das habilidades cognitivas. Seu conhecimento de Biologia levou-o a enxergar o desenvolvimento cognitivo de uma criança como sendo uma evolução gradativa.

Em 1921, Piaget voltou à Suíça e tornou-se diretor de estudos no Instituto J. J. Rousseau da Universidade de Genebra. Lá ele iniciou o maior trabalho de sua vida, ao observar crianças brincando e registrar meticulosamente as palavras, ações e processos de raciocínio delas.

Em 1923, Piaget casou-se com Valentine Châtenay, com quem teve três filhas: Jacqueline (1925), Lucienne (1927) e Laurent (1931). As teorias de Piaget foram, em grande parte, baseadas em estudos e observações de seus filhos que ele realizou ao lado de sua esposa.

Enquanto prosseguia com suas pesquisas e publicações de trabalhos, Piaget lecionou em diversas universidades européias. Registros revelam que ele foi o único suíço a

ser convidado para lecionar na Universidade de Sorbonne (Paris, França), onde permaneceu de 1952 a 1963. Até a data de seu falecimento, Piaget fundou e dirigiu o Centro Internacional para Epistemologia Genética. Ao longo de sua brilhante carreira, Piaget escreveu mais de 75 livros e centenas de trabalhos científicos.

Piaget morreu em Genebra, em 17 de setembro de 1980.

Lev Semynovitch Vygotsky (1986-1934)

Lev S. Vygotsky foi professor e pesquisador contemporâneo de Piaget. Nasceu em Orsha, pequena cidade da Bielorrusia, em 17 de novembro de 1896, num contexto histórico de conflitos políticos (Revolução Russa), sendo este um dos motivos pelo qual sua obra só foi conhecida e valorizada mais recentemente, apesar de sua pesquisa ser de suma importância para a ciências educacionais e psicológicas.

Morou e viveu na Rússia, onde também morreu de tuberculose aos 37 anos. Construiu sua teoria tendo por base o desenvolvimento do indivíduo como resultado de um processo sócio-histórico, enfatizando o papel da linguagem e da aprendizagem nesse desenvolvimento, sendo essa teoria considerada histórico-social. Sua questão central é a aquisição de conhecimentos pela interação do sujeito com o meio.

APÊNDICE II – Estrutura do Tomcat e integração com um Servidor HTTP

O Tomcat é dividido em 3 partes principais:

- Catalina – esse componente é o container *Web* em si (responsável pelo gerenciamento do Servlets), ou seja, é o coração do Tomcat;
- Jasper – esse componente é o compilador de JSPs, ou seja, é o responsável por transformar JSPs em Servlets equivalentes e pela posterior compilação desses Servlets;
- Conectores – implementam a interface do container com protocolos de comunicação utilizados para interagir com os clientes.

No caso específico do Tomcat, a integração com um servidor HTTP é opcional, pois o Tomcat vem configurado para utilizar um conector chamado Coyote, que permite ao container *Web*, responder a requisições HTTP sem a necessidade de integração com um servidor *Web*. Esse conector é um aplicativo Java, que faz a interface do container com o protocolo HTTP, convertendo as requisições e respostas HTTP em objetos Java que são manuseados pelo container (HttpRequest e HttpResponse).

A integração do Tomcat com um servidor HTTP é recomendada no caso em que a aplicação *Web* desenvolvida é utilizada para servir uma grande quantidade de recursos *Web* estáticos, como páginas HTML, cujo conteúdo não é gerado automaticamente. Isso se deve ao fato de que o Tomcat não é tão otimizado para essas tarefas, e portanto sugere-se sua integração com um servidor HTTP especializado através de conectores específicos (como é o caso da integração do Tomcat com o Apache, via o conector mod_jk2). Dessa maneira, o servidor *Web* se responsabiliza pelo atendimento das requisições a conteúdos HTML, por exemplo, e repassa as requisições a conteúdo Java dinâmico (JSPs) para o container *Web*.

A integração do Tomcat com o Apache é muito utilizada por provedores de conteúdo *Web* visto que estes provedores normalmente utilizam diversas tecnologias para provimento de conteúdo *Web* dinâmico e não apenas Java. É comum ver servidores que armazenam aplicações em PHP, Perl, Java, dentre outras.

APÊNDICE III – Conceito de *Frameworks* no Desenvolvimento de Softwares

O que é um *Framework*?

Um *framework* é um conjunto de classes e interfaces que agregam valor no desenvolvimento de software. Um *framework* tem as seguintes características:

- Um *framework* agrega diversas classes, onde cada uma delas é uma abstração de um conceito particular;
- O *framework* define como estas abstrações resolvem um problema;
- Os componentes do *framework* são reutilizáveis;
- Os *frameworks* implementam *patterns* em alto nível.
- Um bom *framework* tem comportamento genérico que pode ser utilizado em diferentes aplicações.

Framework x Biblioteca de funções

Bibliotecas de funções são utilizadas pela sua aplicação para realizar tarefas específicas;

Um *framework* provê componentes cooperativos que sua aplicação pode estender para obter determinadas funções. As partes do *framework* que podem ser estendidas são chamadas *extension points* (pontos de extensão).

APÊNDICE IV – *Design Patterns*

O que são *Design Patterns* e quais seus benefícios

por Mauro Roberto Rodrigues

A origem de design *patterns* encontra-se em um trabalho feito por um arquiteto chamado Christopher Alexander durante os anos 70 com seu livro "A *Pattern* Language" [Alex77]. Ele disse que cada "*pattern*" descreve um problema que ocorre várias vezes em nosso ambiente e então descreve a solução reutilizável para o problema de modo que você não precise procurar a solução mais de uma vez. O conceito foi sendo adaptado e desenvolvido para a área de informática. No entanto este conceito tornou-se popular pela primeira vez com a publicação do livro "*Design Patterns*: *Elements of Reusable Object-Oriented Software*" [Gamma95].

Desenvolver software orientado a objetos é difícil. Desenvolver software orientado a objetos reutilizável é muito mais difícil, pois o projeto deve atender um problema específico e também ser geral o suficiente para atender requisitos e problemas futuros. A habilidade de alterar classes de forma a se tornar útil em mais de um caso específico, é uma qualidade normalmente encontrada em projetistas com muita experiência em projeto de software orientado a objetos e em manutenção de sistemas.

Desenvolvedores experientes encontram moldes de classes e objetos em muitos sistemas em que participaram. O desenvolvedor utiliza sua experiência em projetos de sucesso para não ter que redescobrir a solução do problema novamente.

Design *patterns* são os registros destas experiências. Estes moldes resolvem problemas de projetos específicos e tornam estes projetos mais flexíveis, elegantes e reutilizáveis. Eles ajudam o desenvolvedor a fazer o projeto corretamente e de forma mais rápida.

Benefícios

Formam um vocabulário comum que permite uma melhor comunicação entre os desenvolvedores, uma documentação mais completa e uma melhor exploração das alternativas de projeto. Reduz a complexidade do sistema através da definição de abstrações que estão acima das classes e instâncias. Um bom conjunto de padrões aumenta muito a qualidade do programa;

Constituem uma base de experiências reutilizáveis para a construção de software. Funcionam como peças na construção de projetos de software mais complexos; podem ser considerados como micro-arquiteturas que contribuem para arquitetura geral do sistema;

Reduzem tempo de aprendizado de uma determinada biblioteca de classes. Isto é fundamental para o aprendizado dos desenvolvedores novatos;

Quanto mais cedo são usados, menor será o re-trabalho em etapas mais avançadas do projeto.

Referências bibliográficas:

[Alex77] ALEXANDER, Christopher. A pattern Language: towns, buildings, construction. Oxford: University Press, 1977.

[Gamma95] GAMMA, Erich. Design patterns: elements of reusable object-oriented software. Addison-Wesley, 1995.

[Heywo96] HEYWORTH, James. Introduction to design patterns in Delphi. Camberra PC Users Group Delphi, 1996.

APÊNDICE V – Diagramas Ilustrativos dos design *patterns* utilizados

1) Data Access Object design pattern

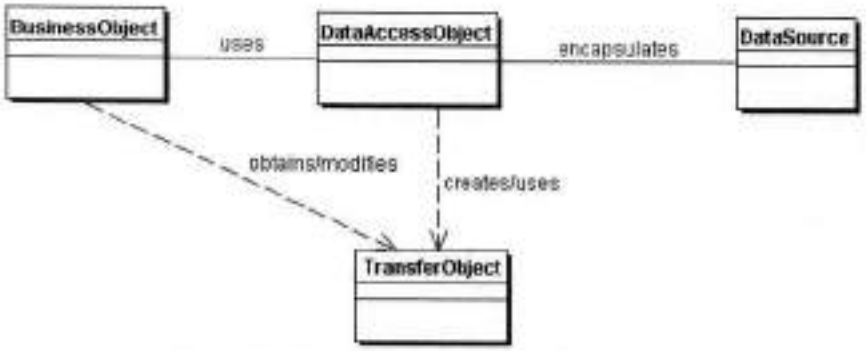


Figura 26 - Diagrama de classes do DAO *design pattern*

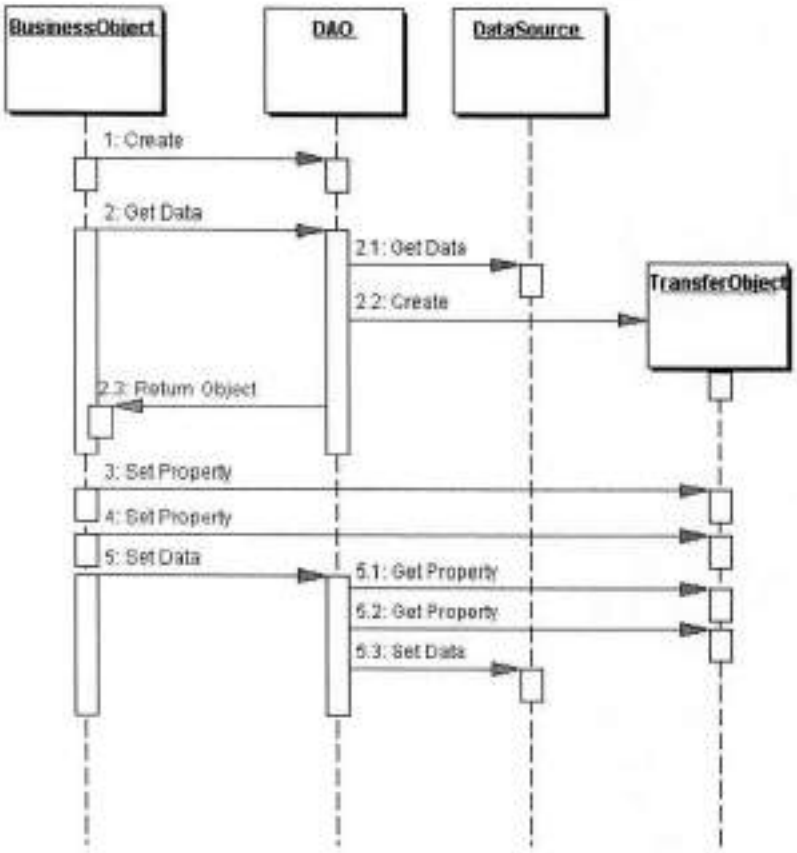


Figura 27 - Diagrama de sequência do DAO *design pattern*

2) Service Locator design pattern

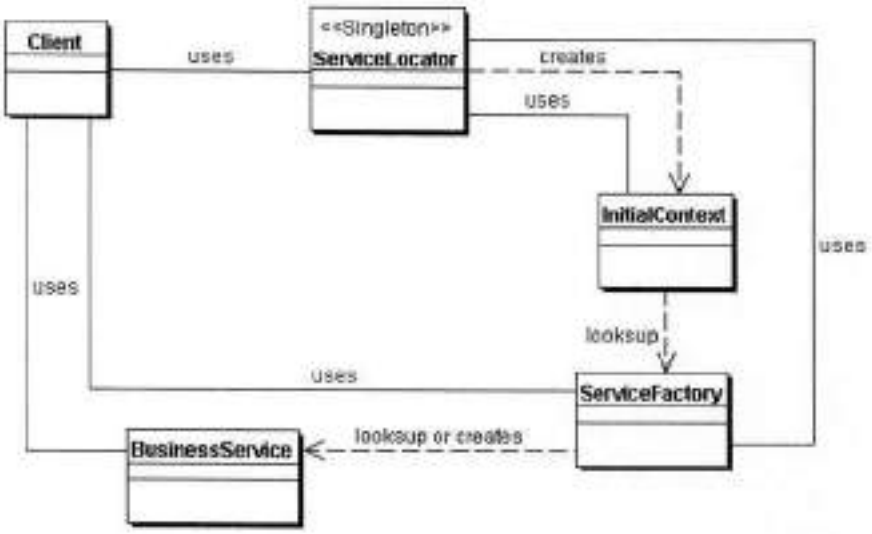


Figura 28 - Digrama de classes do *Service Locator design pattern*

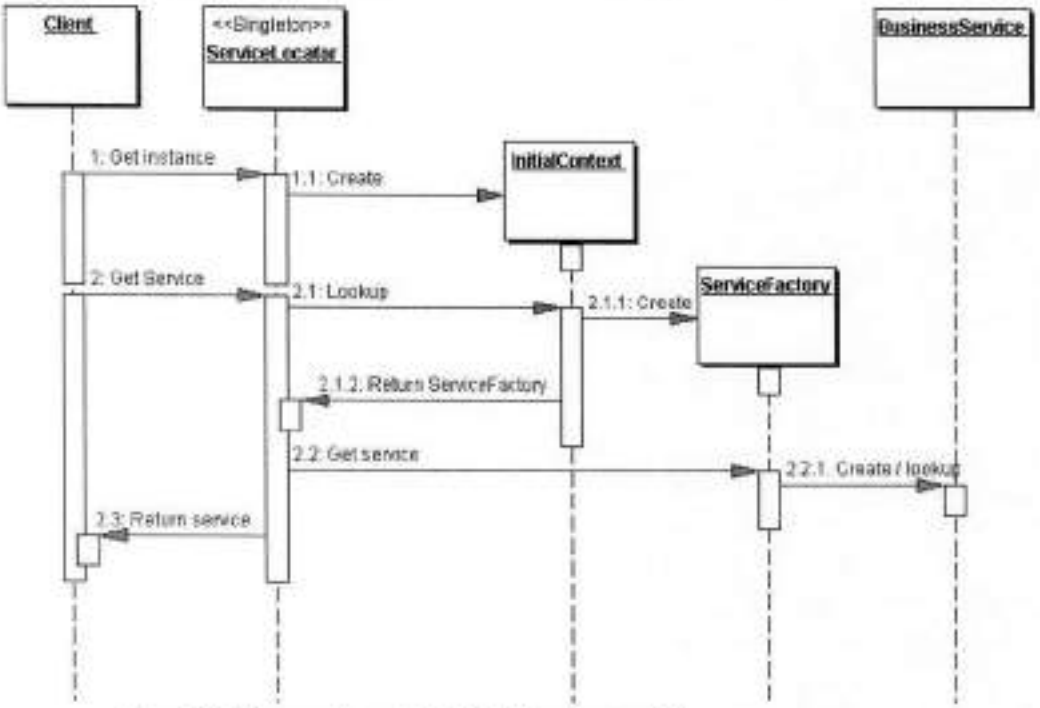


Figura 29 - Diagrama de sequência do Service Locator *design pattern*

3) Front Controller design pattern

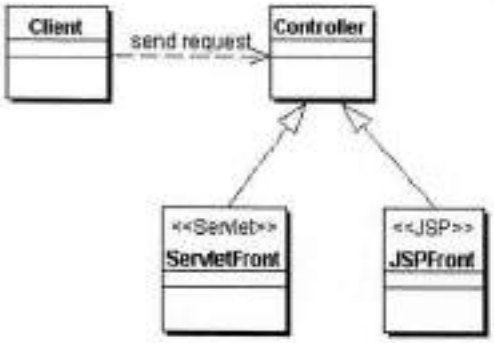


Figura 30 - Diagrama de classes do *Front Controller design pattern*

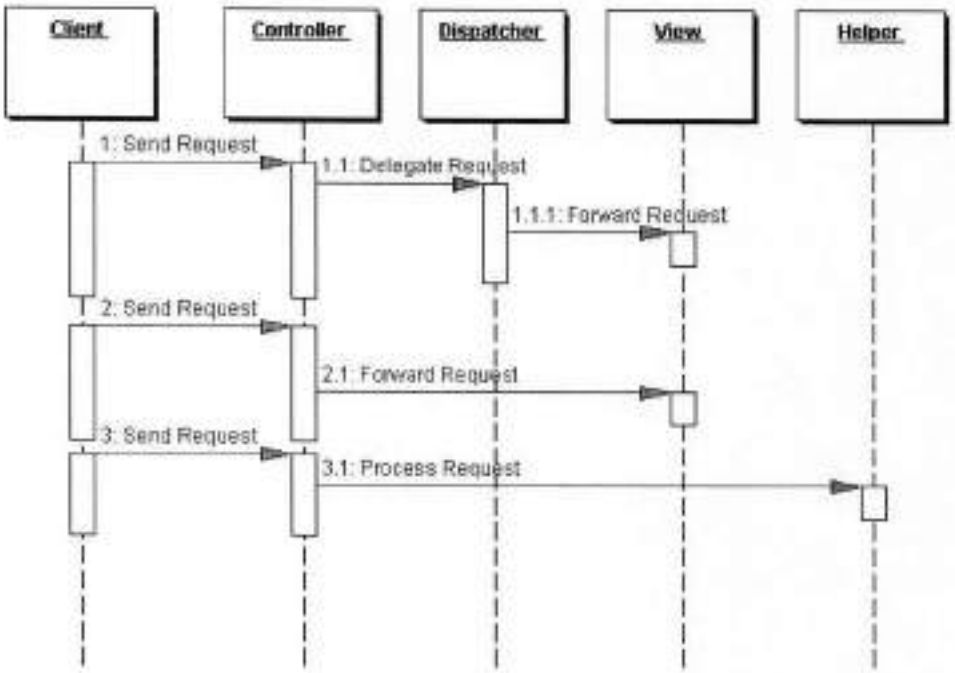


Figura 31 - Diagrama de Sequência do Front Controller *design pattern*

4) Intercepting Filter design pattern

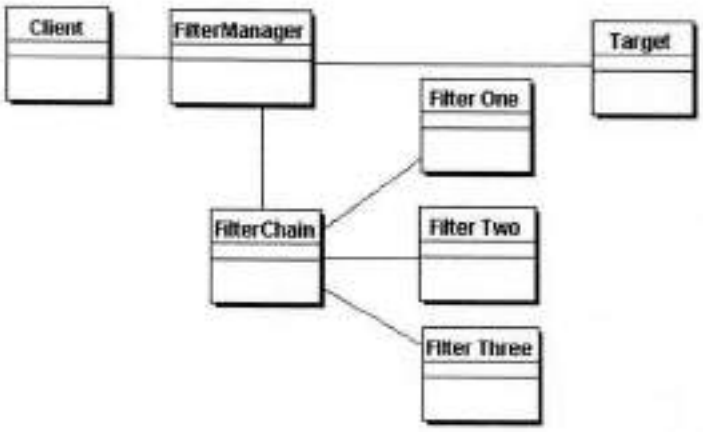


Figura 32 - Diagrama de classes de *Intercepting Filter design pattern*

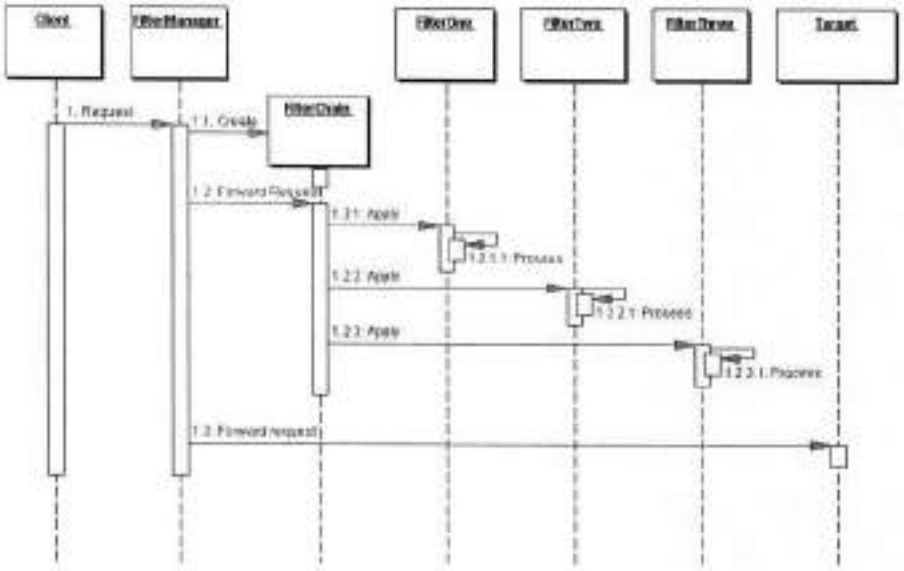


Figura 33 - Diagrama de sequência do Intercepting Filter *design pattern*

5) Command design pattern

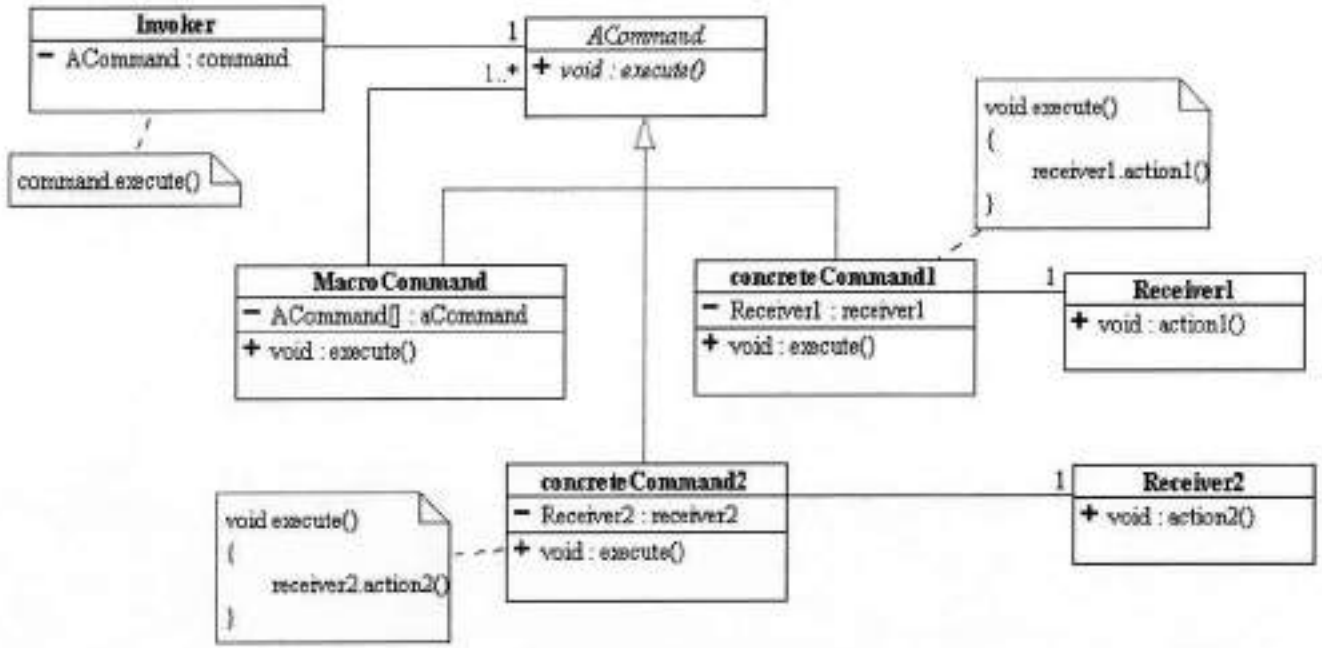


Figura 34 - Diagrama de classes do *Command design pattern*

6) *View Helper design pattern*

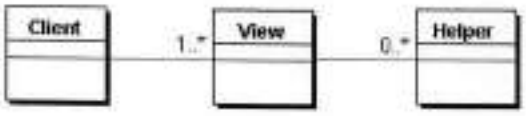


Figura 35 - Diagrama de classes do *View Helper design pattern*

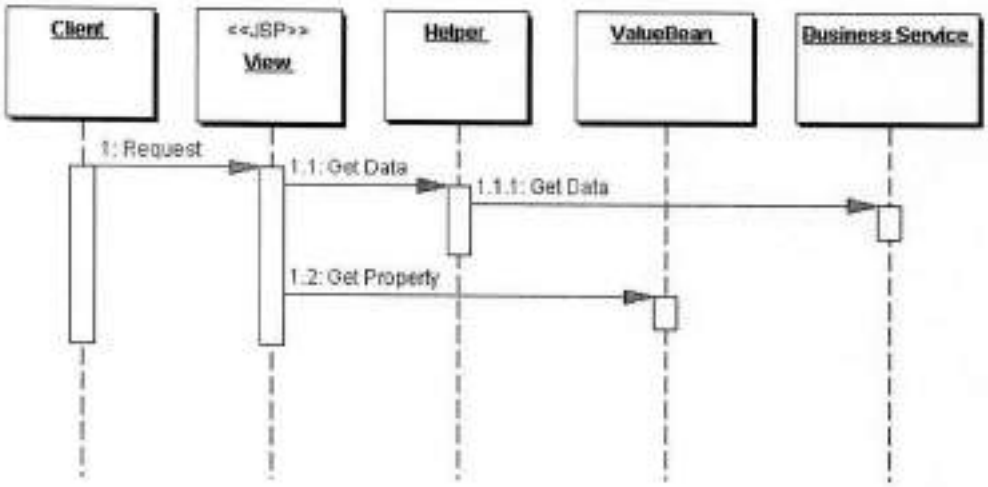


Figura 36 - Diagrama de seqüência do View Helper *design pattern*

7) Composite View design pattern

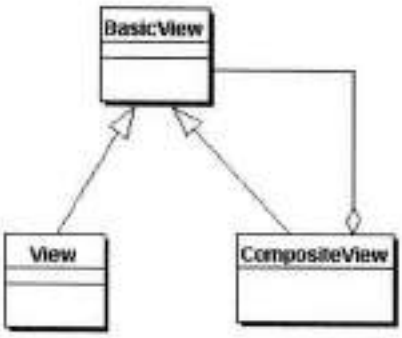


Figura 37 - Diagrama de classes do *Composite View design pattern*

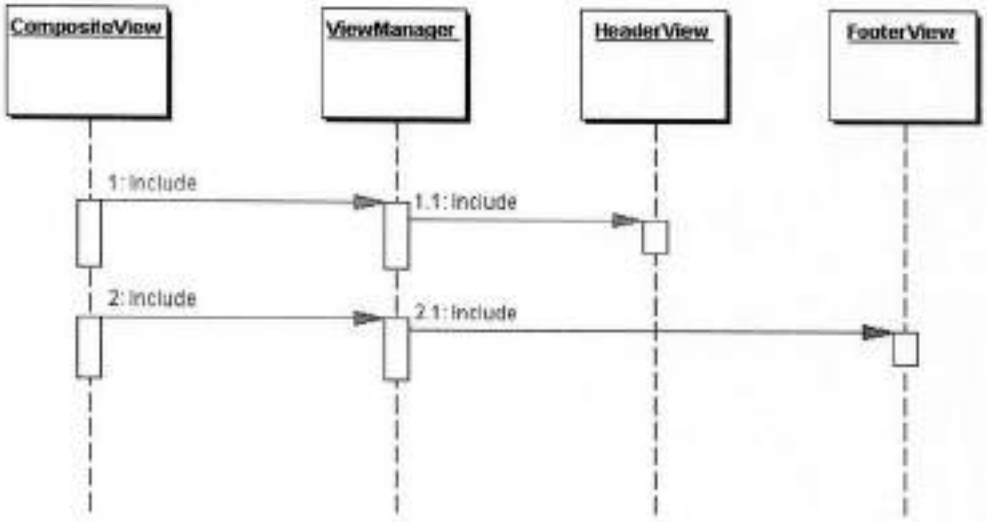


Figura 38 - Diagrama de sequência do *Composite View design pattern*

8) *Transfer Object design pattern*

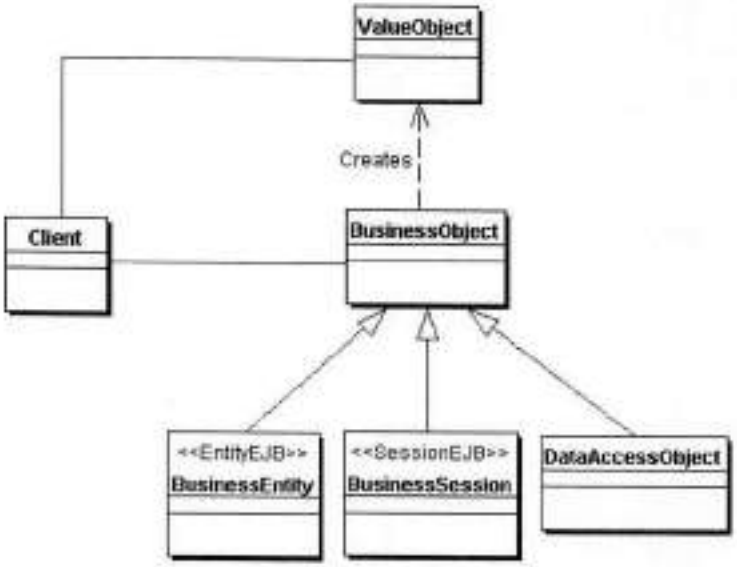


Figura 39 - Diagrama de classes do *Transfer Object design pattern*

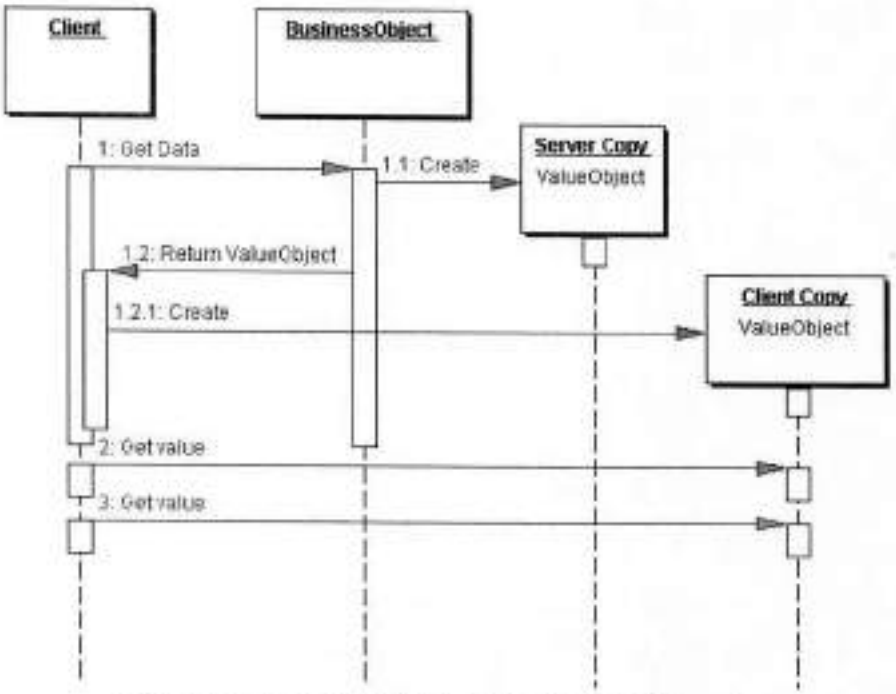


Figura 40 - Diagrama de sequência do *Transfer Object design pattern*

9) Façade design pattern

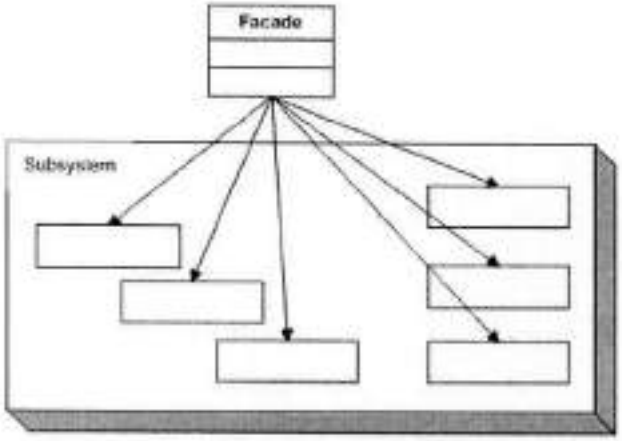


Figura 41 – Diagrama de classes do *Façade design pattern*

10) Template method design pattern

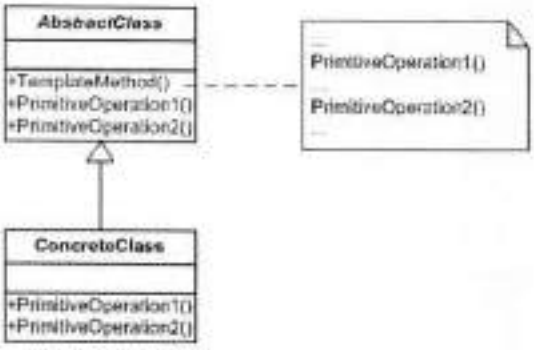


Figura 42 - Diagrama de classes do *Template Method design pattern*

11) Strategy design pattern

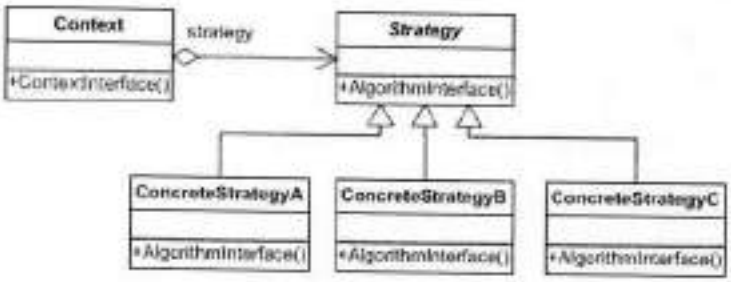


Figura 43 - Diagrama de classes do *Strategy design pattern*

APÊNDICE VI – Sites dos Fornecedores das Ferramentas Utilizadas

www.java.sun.com – J2SDK (kit para desenvolvimento de aplicativos Java);

www.eclipse.org – Eclipse (IDE) e *plug-ins*;

www.mysql.org - MySQL *Server* (RDBMS), MySQL Connector/J (*driver* JDBC) e MySQL *Control Center* (cliente gráfico para gerência do servidor de banco de dados).

www.apache.org – *Jakarta Tomcat* (servidor/*container Web*), *Jakarta Taglibs* (JSTL/*Standard*), Jakarta Commons (biblioteca de componentes), Struts (*framework* MVC), JMeter (ferramenta para stress test), Ant (ferramenta para execução de scripts *cross-platform*);

www.esm.jp/jude-web - JUDE/Community (modelador UML);

www.jfree.org/jfreechart - JFreeChart (API para geração de gráficos);

cewolf.sourceforge.net – Cewolf (custom tag para utilização do JFreeChart em aplicativos *Web*);

APÊNDICE VII – Conteúdo do CD

No CD entregue junto com a documentação estão inclusas todas as ferramentas utilizadas para desenvolvimento do projeto, além de toda a documentação gerada no decorrer do projeto.

As ferramentas inclusas e seus fornecedores são especificados no apêndice VI.

A documentação está dentro do diretório \docs, que inclui os seguintes arquivos:

- **sarda_model_1.0.jude** – arquivo gerado com a ferramenta JUDE/Community e contendo todos os diagramas UML elaborados;
- **documentaçãoFinal.doc / documentaçãoFinal.pdf** – versão digital da documentação final nos formatos .doc e .pdf.
- **apresentação_final.ppt** – apresentação final feita para banca de examinadores no dia 12/12/05.
- **sarda.sw.rs.1.0.doc** – documento de especificação de requisitos de software (ERS).
- **sarda.sw.uc.0.4.doc** – documento contendo especificação (cartões) dos casos de uso do módulo de infra-estrutura do sistema;
- **sarda.especificação.1.0.doc** – relatório do projeto de formatura solicitado pelo responsável da disciplina para fins de acompanhamento e controle do projeto.
- **projeto_sarda.zip** – arquivo compactado do projeto implementado no Eclipse. Contém fontes, binários, arquivos *Web* (JSP, CSS, JavaScript), arquivos de configuração, script para *build* da aplicação, dentre outros;
- **sarda.war** – arquivo executável da aplicação. Para executar este arquivo basta colocá-lo no diretório \webapps do servidor Tomcat fornecido no CD. Para levantar o servidor, pode-se utilizar o arquivo startup.bat, que fica no diretório *\bin* do servidor.
- **db_script** – este arquivo contém o *script* para geração das tabelas do banco de dados. Esse *script* foi gerado utilizando-se a ferramenta Mysql Administrator, também fornecida no CD.

LISTA DE REFERÊNCIAS

ALUR, D.; CRUPI, J.; MALKS, D. **Core J2EE Patterns: Best Practices and Design Strategies.** Prentice Hall / Sun Microsystems Press, 2001, 1st Edition. 496p.

CHICKERING, A. W., GAMSON, Z. F. (1987). **Seven Principles of Good Practice in Undergraduate Education**. AAHE Bulletin, 39, 3-7.

CD de referência do Projeto de Formatura: SARDA – Sistema de Apoio à realização de diagnósticos de aprendizagem. PCS2502-14-CD-2005. Escola Politécnica da USP, São Paulo, 2005.

DEITEL, H. M.; DEITEL, P. J. **Java How to Program.** Prentice Hall, 2002, 5th Edition. 1536p.

FARLEY, J. **Java Distributed Computing.** O'Reilly Publishers, 1998.

PRESSMAN, R. S. **Engenharia de Software**. McGraw Hill, 2002. 5ª edição. 872p.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, JOHN. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1995, 1st Edition. 395p.

MAIA, R. F. **Sistema multi-agentes para acompanhamento e auxilio de avaliação de alunos em ambientes de ensino à distância.** 2004. 167P. Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo. São Paulo.

ROBBINS, S. P. **Administração Mudanças e Perspectivas.** Saraiva, 2000. 1ª edição. 524p.

SINGH, I.; STEARNS, B.; JOHNSON, M.; THE ENTERPRISE TEAM. **Designing Enterprise Applications with the J2EE™ Plataform.** Addison-Wesley, 2002, 2nd Edition. 440p.

VYGOTSKY, L. S. **Mind in Society**. Cambridge, MA: Harvard University Press, 1978.