ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de Sistemas Computacionais

Sistema de Avaliação de Docentes e Banco de Créditos

Trabalho de Conclusão de Curso

Engenharia da Computação

Erik Mesel Ferreira Pires

Orientador: Prof. Tiago Massoni

Recife, novembro de 2007

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de Sistemas Computacionais

Sistema de Avaliação de Docentes e Banco de Créditos

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Erik Mesel Ferreira Pires

Orientador: Prof. Tiago Massoni

Recife, novembro de 2007

UNIVERSIDADE DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Erik Mesel Ferreira Pires

Sistema de Avaliação de Docentes e Banco de Créditos

Resumo

Com o crescimento do Departamento de Sistemas Computacionais da Escola Politécnica, está cada vez mais difícil de controlar as atividades dos docentes e de avaliá-los. Alguns docentes têm o desejo de se afastar durante um tempo do departamento para fazer pós-doutorado, mas para que algum docente se afaste é necessário que haja um planejamento prévio. Alguém precisa substituí-lo em suas atividades. O propósito deste trabalho é o desenvolvimento de dois sistemas de informação integrados. Um deles seria o responsável por gerenciar as atividades desenvolvidas pelos docentes e realizar avaliações dos mesmos. O outro sistema seria responsável por organizar uma fila de espera contendo os professores que desejam se afastar do departamento para fazer pós-doutorado.

.

Abstract

With the develop of the Department of Computing Systems of the Polytechnic School, is increasingly difficult to control the activities of teachers and evaluate them. Some teachers have the desire to move away during a time of the department to post-doctorate, but to a teacher move away it is necessary to have a prior planning. Someone needs to replace it in its activities. The purpose of this work is the development of two integrated information systems. One would be responsible for managing the activities developed by teachers and conduct assessments of the same. The other system would be responsible for organizing a queue of waiting with teachers who want to move away from the department to post-doctorate.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

URL *Uniform Resource Locator* (Localizador Uniforme de Recursos).

HTML *Hiper Text Mark-up Language* (Linguagem de Marcação de Hiper Texto).

MVC *Model-View-Controller* (Modelo-Visão-Controle).

WEB Sinônimo de WWW que por sua vez significa *Word Wide Web* (A Grande Rede Mundial).

JSP Java *Server Pages.*

JSF *Java Server Faces.*

CSS *Cascading Style Sheet* (Folha de Estilo em Cascata).

ASP *ActiveServer Pages.*

UML *Unified Modeling Language* (Linguagem de Modelagem Unificada).

MIS *Management Information System* (Sistema de Informações Gerenciais).

DSS *Decision Support System* (Sistema de Suporte à Decisão).

SGBD Sistema Gerenciador de Banco de Dados.

IDE *Integrated Development Enviroment*

RHDS *Red Hat Developer Studio.*

Agradecimentos

Primeiramente, gostaria de agradecer a Deus, pois é ele quem nos ilumina e conduz. Logo após agradeço aos meus pais (Moacyr e Dunya) e irmãos (Graco e Layla), são eles que sempre estão ao meu lado me ajudando quando preciso e também são os principais responsáveis pela pessoa que me tornei. Quero deixar um agradecimento especial também para minhas duas avós, que sempre torceram e oraram muito por mim e com certeza estão muito felizes por mais essa conquista.

Ao meu primo Haim Mesel e sua esposa Sarah, que estão sempre me ajudando, tanto no campo profissional quanto no pessoal. Sempre me aconselhando e dando dicas de que caminho ou atitudes tomar.

Agradeço também a todos os membros (professores e alunos) do DSC pela ótima formação. Em especial aos professores Tiago Massoni (que foi meu orientador nesse trabalho, me ajudou muito e teve bastante paciência), Cristine Gusmão e Carmelo Bastos. Aos amigos que fiz durante o curso de graduação, como Júlio, Marcos, Tiago e Ailton. Vocês são partes fundamentais dessa conquista. Obrigado!

Capítulo 1

Introdução

Hoje em dia, todo o corpo docente do Departamento de Sistemas Computacionais (DSC) [1] é composto por doutores e doutorandos. O que é uma prova do alto nível que o departamento atingiu, mesmo sendo relativamente novo, oficialmente foi criado em setembro de 2004. Porém é extremamente importante, tanto para os docentes quanto para o departamento, que eles façam pós-doutorados. Nos dias de hoje, os profissionais não podem parar de buscar novos conhecimentos, pois a concorrência entre profissionais e instituições é muito grande; principalmente na nossa área, Tecnologia da Informação (TI). Na área de TI, isto se torna mais importante devido à evolução contínua das tecnologias. O profissional que não for se especializando com o passar do tempo ficará desatualizado e poderá perder espaço no mercado de trabalho.

Para o departamento é interessante que seu corpo docente seja o mais qualificado possível, pois consequentemente o nível de ensino e pesquisa será maior. Também é interessante ter docentes com o maior nível de qualificação possível para obtenção de novas parcerias e até para a criação de novos cursos, sejam eles de graduação, especialização, mestrados ou doutorados. Outra vantagem é que nesses cursos fora do país, os professores estarão divulgando o nome do DSC para o mundo todo, estarão conhecendo vários outros pesquisadores da área e isso consequentemente pode render muitos frutos, através de parcerias de pesquisas e intercâmbios entre universidades. O que seria de muita valia para todos do DSC.

Outro problema encontrado no DSC é o controle das atividades dos docentes. Existe um conjunto de atividades que podem ser desenvolvidas pelos docentes e a cada atividade é atribuída uma pontuação. Então no final do semestre calcula-se a pontuação de cada professor. Todo professor tem uma meta a atingir, e através da avaliação ele saberá se o resultado foi satisfatório ou não. Hoje em dia, essa avaliação é feita através de uma planilha que cada professor deve preencher todo semestre, e quando necessário o coordenador pega a planilha e vai avaliar cada professor. Esse é um trabalho bastante tedioso, pois seus resultados não são obtidos facilmente. Além de ser uma avaliação bastante limitada, já que cada avaliação só pode ser feita para um docente em um determinado semestre.

[1] http://dsc.upe.br/

1.1 Objetivos

A melhor forma de se trabalhar com dados hoje em dia é através de sistemas de informação. Sistemas de Informação (SI) são conjuntos de elementos inter-relacionados que coletam (entrada), manipulam e armazenam (processo) e disseminam (saída) dados e informações [1] . A grande vantagem dos sistemas de informação é que eles provêem um fluxo rápido e confiável às informações, e informação nos dias de hoje é o que mais vale para o mundo dos negócios.

O objetivo desse trabalho é o desenvolvimento de dois sistemas de informação para sanar as dificuldades, citadas anteriormente, que o DSC vem passando. Estes dois sistemas deverão trabalhar de forma integrada, com compartilhamento de dados e ocorrendo a comunicação entre eles. Eles vão compartilhar um Banco de Dados e uma vez o usuário logado num sistema, ele terá acesso ao outro sem precisar fazer outro login.

Um sistema é o responsável pelo banco de créditos dos docentes. O banco de créditos funciona de forma análoga a um banco de horas existentes nas empresas. Sempre que um professor quiser passar um tempo afastado do departamento, ele terá que lecionar algumas disciplinas a mais que a sua carga normal. Então o banco de créditos será o sistema que organizará as informações necessárias para o afastamento temporário de professores para pós-doutorado.

O outro sistema é o de avaliação de docentes. É nele onde encontraremos todos os cadastros de docentes, atividades e quais atividades cada docente realizou. Através deste sistema o coordenador obterá informações para tomadas de decisões importantes dentro do DSC, como a divisão de atividades a serem exercidas, entre os membros do departamento. Ele tornará as avaliações muito mais úteis, e elas poderão ser feitas em qualquer momento de forma rápida e segura.

1.2 Estrutura do Trabalho

No capítulo 2 é passado um embasamento teórico sobre sistemas de informação, desenvolvimento iterativo, aplicações Cliente/Servidor e o que ocorre em cada uma das partes. Apresentamos também a arquitetura MVC (*Model View Controller*), o que ocorre em cada um de seus componentes e como surgiu o seu aprimoramento para aplicações web, que é conhecido como o Modelo 2 do MVC.

No capítulo 3 nós faremos uma descrição ampla dos sistemas e depois iremos destrinchá-los e apresentaremos os requisitos do nosso sistema e o porquê de eles serem essenciais em nossa aplicação.

No capítulo 4 iremos descrever como foi feita a implementação da solução. Falaremos sobre desenvolvimento iterativo, programação em camadas, sobre o *framework* Struts[2] e suas configurações, além de mostrarmos as visões diferentes que os professores e o coordenador terão dos sistemas.

No capítulo 5, são apresentadas as conclusões do trabalho e sugestões para trabalhos futuros.

[2] http://struts.apache.org/

1.3 Contribuições

As principais contribuições deste trabalho são:

- Organizar as informações de atividades exercidas pelos docentes;
- Tornar a avaliação de docentes uma tarefa mais agradável;
- Possibilitar ao DSC a liberação de docentes para fazer pós-doutorados de uma forma bem controlada.

Capítulo 2

Fundamentação Teórica

O uso dos Sistemas de Informação torna-se cada vez mais primordial para as instituições. No mundo dos negócios, as tomadas de decisões estão altamente conectadas com os SIs. Tanto é que as grandes empresas do hoje em dia possuem seu próprio departamento de TI, além de ter profissionais gabaritados a analisar as saídas dos sistemas de informação nas principais áreas das empresas.

Desenvolvimento iterativo é um processo de desenvolvimento baseado em entregas periódicas, constantes interações com os clientes e mudanças de requisitos. Estas mudanças são feitas baseadas nos *feedbacks* dos clientes, ao final de cada iteração.

Os sistemas de informação existentes hoje funcionam como aplicações Cliente/Servidor. Nessa arquitetura o cliente apenas faz requisições aos servidores que processam a resposta e a retorna para o cliente. Eles se comunicam através de uma rede de comunicações utilizando seus protocolos.

Outro paradigma de projeto bastante utilizado hoje em dia é o MVC. Este paradigma divide as responsabilidades do sistema em três grupos. O modelo, a visão e o controle.

2.1 Sistemas de Informação

Para se ter uma boa idéia do que é um SI, primeiro devemos saber qual a diferença entre dados e informações. Os dados são meramente fatos, não tem significado algum. A partir do momento em que combinamos os fatos de uma maneira que possamos fazer uso das mesmas com algum significado, transformamos os fatos em informação. Informação é um conjunto de fatos organizados de tal forma que adquirem valor adicional além do valor do fato em si[1] . Como exemplo, no nosso sistema, em algum momento a quantidade de orientações de trabalho de final de curso que um docente realizou não tenha nenhum valor além do fato. Porém o que o usuário do sistema pode estar querendo é a quantidade de todas as atividades de ensino realizadas por esse professor. Então a quantidade de orientações se junta aos dados das outras atividades e passam a ser uma informação. Essa informação certamente será usada para tomada de decisões para o DSC.

Para uma informação ser valiosa para gerentes e tomadores de decisões, ela deve ter as seguintes características:

- Precisa – Não pode ter erros.
- Completa – A informação precisa ter todos os fatos importantes.
- Econômica – A informação também deve ser de produção relativamente econômica.
- Flexível – A informação flexível pode ser usada para várias finalidades.
- Confiável – A confiabilidade da informação depende da confiabilidade do método de coleta dos dados.
- Relevante – A informação relevante é importante para o tomador de decisões.
- Simples – A informação também deve ser simples, não deve ser exageradamente complexa. A informação sofisticada e detalhada pode não ser necessária.
- Em tempo – A informação em tempo é enviada quando necessário.
- Verificável – A informação deve ser verificável. Isto significa que se pode checá-la para saber se está correta.

Estas características também tornam a informação mais valiosa para a organização. Se a informação não é precisa ou completa, decisões ruins podem ser tomadas, gerando um alto custo à organização. Se uma previsão imprecisa de demandas futuras indica que as vendas serão muito altas quando é o oposto que acontece, uma empresa pode investir alto em uma nova fábrica que não é necessária. Além disso, se a informação não é pertinente a situação, não é fornecida aos tomadores de decisão no tempo certo, ou é complexa demais para ser compreendida, ela pode ter pouco valor para a organização [1] .

Dentre os tipos de SI, encontramos os de processamentos de transações (TPS), que são aqueles que são projetados para lidar com volumes de transações de negócios comuns, e os sistemas de informações gerenciais (MIS), projetados para auxiliarem a administração no atingimento de suas metas.

Os sistemas de informações gerenciais produzem uma variedade de relatórios. Os sistemas de informação projetados para dar apoio na solução de problemas específicos são classificados como sistemas de suporte à decisão (DSS), esse é o caso dos sistemas de avaliação de docentes e banco de créditos.

2.2 Desenvolvimento Iterativo

Tempos atrás, os softwares eram em geral desenvolvidos através do modelo cascata (*waterfall* em inglês) de desenvolvimento. O modelo cascata é o modelo em que antes de começar a implementação do software, é realizado todo o projeto do mesmo. São feitos todos os levantamentos e detalhamentos de requisitos, é definida a arquitetura do sistema, e até o fim do projeto, os desenvolvedores devem seguir o que foi projetado no começo do ciclo de desenvolvimento. Antes mesmo de se iniciar a implementação. Porém o grau de desperdício de software era muito alto, assim como a taxa de erros em estimativas de tempo e custo em projetos. Mudanças em alguns requisitos de sistemas custavam muito para o projeto e algumas vezes eram impossíveis de serem realizadas.

Em pesquisa realizada pelo *Standish Group*[3], ficou constatado que 45% das funcionalidades dos sistemas não são usadas nunca, e que 19% são usadas raramente (ver Figura 1) [2],3]. Ou seja, frequentemente mais da metade do esforço de desenvolvimento é desperdiçado

[3] http://www.standishgroup.com/

em funcionalidades que raramente ou nunca são usadas. Para diminuir esses números passou a ser necessária a inclusão dos clientes no processo de desenvolvimento, além de passar a ter entregas periódicas para se ter um *feedback* dos clientes se as funcionalidades implementadas estão de acordo com suas necessidades.

Foi a partir desse contexto, que foi criado o modelo de desenvolvimento iterativo. O desenvolvimento iterativo – em contraste com o desenvolvimento em cascata – envolve programações iniciais e testes de sistemas parciais em ciclos repetitivos. Isto assume que o desenvolvimento inicia antes que todos os requisitos estejam detalhados; *feedback* é usado para esclarecer e melhorar as especificações envolvidas.

Nós desenvolvemos através de pequenos e rápidos passos, *feedbacks*, e adaptações para esclarecer os requisitos e *design*. Em contraste, desenvolvimento em cascata promove uma grande especulação de requisitos e passos de *design* antes de iniciar a programação. Constantemente, estudos mostram que o desenvolvimento em cascata está intimamente ligado com as maiores taxas de falhas para projetos de softwares e foi historicamente taxado como desenvolvimento baseado em crenças, ao invés de provas significativas. Pesquisas demonstram que métodos iterativos são associados a altas taxas de sucesso e produtividade, e baixos níveis de defeito [2] .

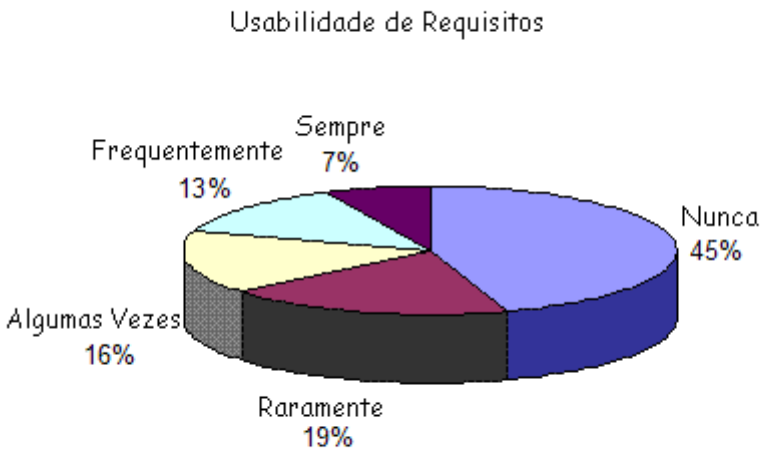


Figura 1. Usabilidade das funcionalidades implementadas em softwares [3] .

Um processo de desenvolvimento de software descreve um método de construir, entregar, e possivelmente de manter um software. O Processo Unificado (UP) surgiu como um popular processo de desenvolvimento iterativo para construir sistemas orientados a objetos [4] . O processo mais difundido é o RUP (*Rational Unified Process*), que é um processo comercial com ferramentas do UP [5] .

Uma iteração é um período de tempo definido dentro de um projeto em que você produz uma versão estável e executável do produto, junto com toda a documentação de apoio, scripts de instalação e similares, necessários para usar a liberação. É também conhecida como ciclo ou tempo definido [3] . Ao fim de cada iteração, os *stakeholders* (conjunto de pessoas que tenham algum interesse no sistema) se reúnem para avaliar o sistema parcial, fazer possíveis alterações nos requisitos e arquitetura, além de decidirem os passos a serem dados na próxima iteração

(quais casos de uso serão implementados total ou parcialmente). O sistema parcial que você obtém no fim das iterações não são protótipos a serem descartados, eles são partes prontas (ou não) do sistema final, que terão agregadas a elas as funcionalidades a serem desenvolvidas nas próximas iterações.

O UP divide o desenvolvimento em quatro fases:

- Concepção – Onde no fim dessa fase deve-se haver um consenso sobre o problema que se tenta resolver.
- Elaboração – Onde se deve chegar a um acordo quanto aos riscos do projeto, custos e prazos.
- Construção – Essa fase se encerra quando os *stakeholders* concordam que têm um sistema que atende as necessidades acordadas.
- Transição – Testes finais e implantação do sistema.

Em cada fase pode ocorrer uma ou mais iterações para se chegar ao resultado necessário. Este método em nada se parece com o modelo em cascata que primeiramente define todos os requisitos e depois se faz todos os *designs*. Concepção não é uma fase de requisitos, nesta fase se faz apenas investigações para decidir se vão dar continuidade ao projeto ou não. De mesmo modo, Elaboração não é fase de *design* ou requisitos. Nessa fase a arquitetura é iterativamente implementada e os principais riscos são atenuados.

O UP divide as suas tarefas em várias disciplinas (conjunto de atividades), as cinco disciplinas principais estão listadas abaixo:

- Modelagem de Negócio – Fazer o modelo de domínio para se visualizar conceitos no domínio da aplicação
- Requisitos – Fazer o modelo de caso de uso e especificações suplementares para identificação dos requisitos funcionais e não-funcionais.
- *Design* – Fazer o *design* dos objetos do software.
- Implementação – Programação e construção do software.
- Testes – Testes do software.

2.3 Cliente/Servidor

No modelo Cliente/Servidor, a comunicação costuma se dar através de uma requisição do cliente enviada para o servidor, pedindo para que alguma tarefa seja executada. Em seguida, o servidor executa a tarefa e envia a resposta. Geralmente, há muitos clientes usando um pequeno número de servidores [6] .

Neste trabalho, o cliente pode ser qualquer computador que possua um navegador e acesso a uma rede de comunicação que algum servidor esteja hospedado. Os usuários nos clientes poderão acessar os servidores através de suas URLs, que são os endereços de seus recursos. Os clientes então preencherão formulários HTML que serão enviados ao servidor, utilizando o protocolo TCP, para que o servidor possa processar a requisição de acordo com os dados enviados. Na Figura 2 encontra-se ilustrado como as máquinas clientes se conectam com o servidor através da web. Nota-se que é possível mais de um cliente acessar a aplicação servidora ao mesmo tempo. Essa é uma das características do modelo Cliente/Servidor.

Já o servidor é uma máquina onde se encontra toda a aplicação e seus dados. O nosso servidor precisará ser um computador que tenha um servidor web instalado nele. É através desse servidor que os clientes poderão se comunicar com a aplicação servidora. A máquina servidora também deverá conter um SGBD, que é onde estarão guardados os dados necessários para funcionamento dos Sistemas.

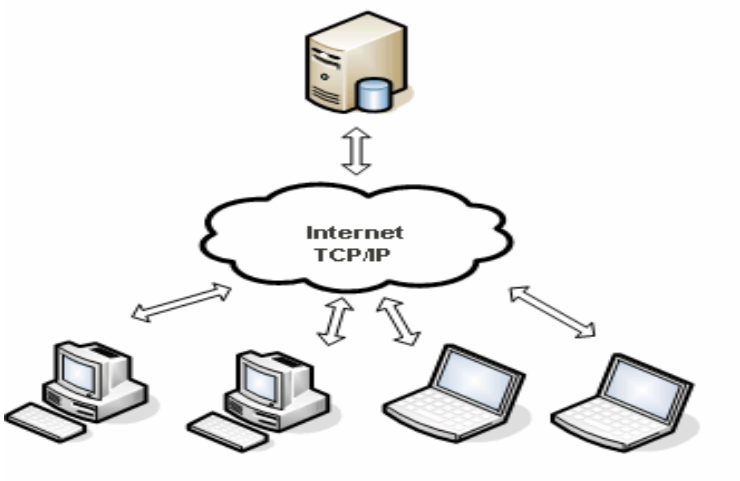


Figura 2. Computadores se comunicando com o servidor através da web.

2.4 Modelo 2 e o MVC

Como sabemos, para se construir um software é necessário a utilização de vários componentes, como as classes, as páginas de apresentação, banco de dados, *taglibs* (*Taglibs* são bibliotecas de "classes Java" que são utilizadas "na forma de *tags*" para auxiliar na geração de conteúdo dinâmico em uma página JSP)[7]. Para o funcionamento do software é necessário que esses componentes se comuniquem.

O mais natural de se imaginar é que todos os componentes devem se comunicar livremente, sem nenhum protocolo. É uma forma bastante eficiente de se trabalhar. Porém, os softwares de hoje em dia necessitam de interfaces gráficas que externem mudanças nos estados de seus dados instantaneamente. Isto tornou os códigos muito difíceis de serem mantidos, além de se tornarem muito repetitivos, já que cada componente da classe de apresentação seria o responsável pelas alterações nas suas regras de negócio.

Nos anos 70 surgiu uma nova solução chamada Modelo MVC (*model-view-controller*). No MVC os componentes são divididos em 3 grupos:

- Modelo (*model*) – É o grupo responsável pelo estado do sistema, são os componentes que acessam o banco de dados e realizam todas as alterações solicitadas pelos clientes em uma requisição.
- Visão (*view*) – É o grupo responsável pela interface com o usuário. Ela recebe as requisições e depois reconstrói a resposta e a passa para o cliente.
- Controle (*controller*) – Como o nome já diz, é o responsável pelo controle da aplicação. É quem repassa a requisição para a parte do modelo capaz de realizá-la.

O modelo MVC não é usado apenas para interface gráfica, qualquer tipo de comunicação entre componentes pode ser organizado por ele. Os dois princípios fundamentais no MVC é que o Controle, que é o componente que sabe qual operação deve ser invocada, despacha as solicitações ao Modelo e a Visão observa o Modelo para refletir as suas possíveis alterações de estado para o usuário. A Figura 3 representa como esses componentes são interligados.

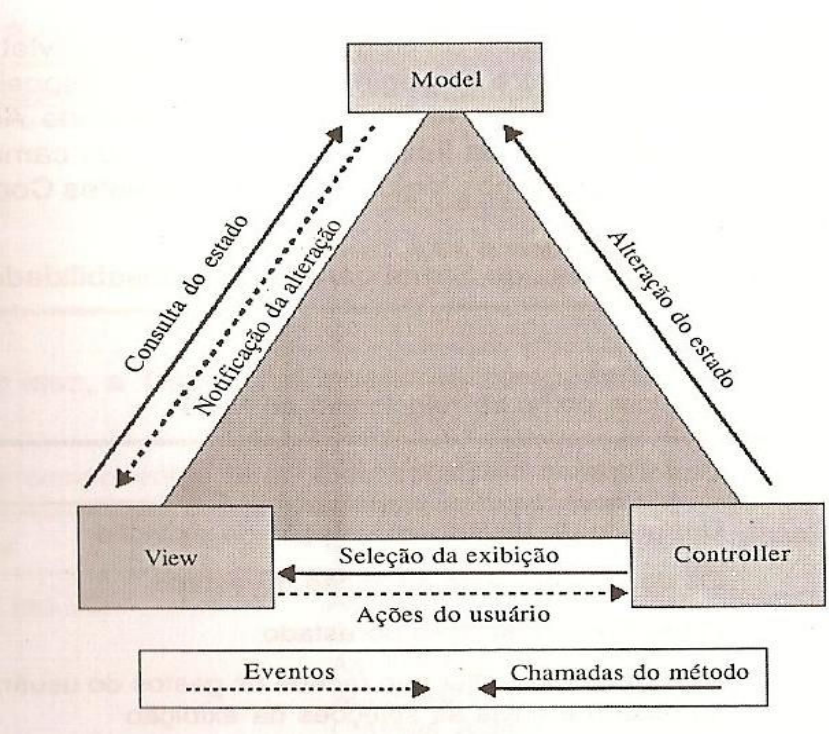


Figura 3. O MVC geralmente é representado como três componentes interligados [8] .

Nos dias de hoje, a área de TI é um dos principais pilares do mundo dos negócios e sempre que é solicitada a construção de um software, os prazos são muito curtos e se exige muito da equipe de desenvolvimento. É ai onde encontramos um dos principais problemas do Modelo MVC tradicional (citado acima). Muitas vezes ocorre de um mesmo arquivo (classes ou JSPs) ter parte integrante tanto do Controle quanto da visão, e consequentemente as partes não podem ser separadas entre equipes de desenvolvimento diferentes. Procurava-se uma solução onde a parte de interface gráfica (Visão) ficasse com os *designers*, enquanto uma parte dos programadores ficasse com as classes de controle e outra parte ficasse com as classes de negócio (Modelo). Foi ai que surgiu um modelo avançado do MVC, o Modelo 2 ou MVC 2.

O Modelo 2 é um avanço do paradigma MVC, e diferentemente do modelo inicial, é específico para aplicações web. Neste modelo as classes JSPs são apenas parte da Visão do projeto. A parte Controle é feita por *servlets* específicos com ajuda ou não de arquivos de configuração. A tecnologia de s*ervlet* hoje é projetada principalmente para utilização com o protocolo HTTP da *World Wide Web*, mas estão sendo desenvolvidos *servlets* para outras tecnologias. Os *servlets* são eficientes para desenvolver soluções baseadas na web que ajudam a fornecer acesso seguro a um *site*, interagir com bancos de dados em favor de um cliente, gerar dinamicamente documentos personalizados de HTML a serem exibidos pelos navegadores e manter informações para sessões exclusivas de cada cliente [9] .

A parte Modelo não sofre muita alteração, ela é a responsável pelas alterações e consultas nos bancos de dados. A única mudança sofrida nessa classe é que na hora da resposta ela passa a se comunicar com o Controle e não com a parte de Visão diretamente. A visão passa a ser um intermediador das duas outras partes do modelo.

Existem dois potentes *frameworks* (entenda por *framework* um conjunto de classes concretas e abstratas de colaboração para ser usada como *template* para solucionar uma relativa família de problemas. Essas classes geralmente são extendidas através de subclasses para aplicações com comportamentos específicos.) [8] que encorajam o uso do Modelo 2. Um deles é o Struts, um projeto *open-source* da fundação Apache[4], o outro é o JSF[5] (*Java Server Faces*), que é desenvolvido pela comunidade Java.

[4] http://www.apache.org/

[5] http://java.sun.com/javaee/javaserverfaces/

Capítulo 3

Requisitos do Sistema

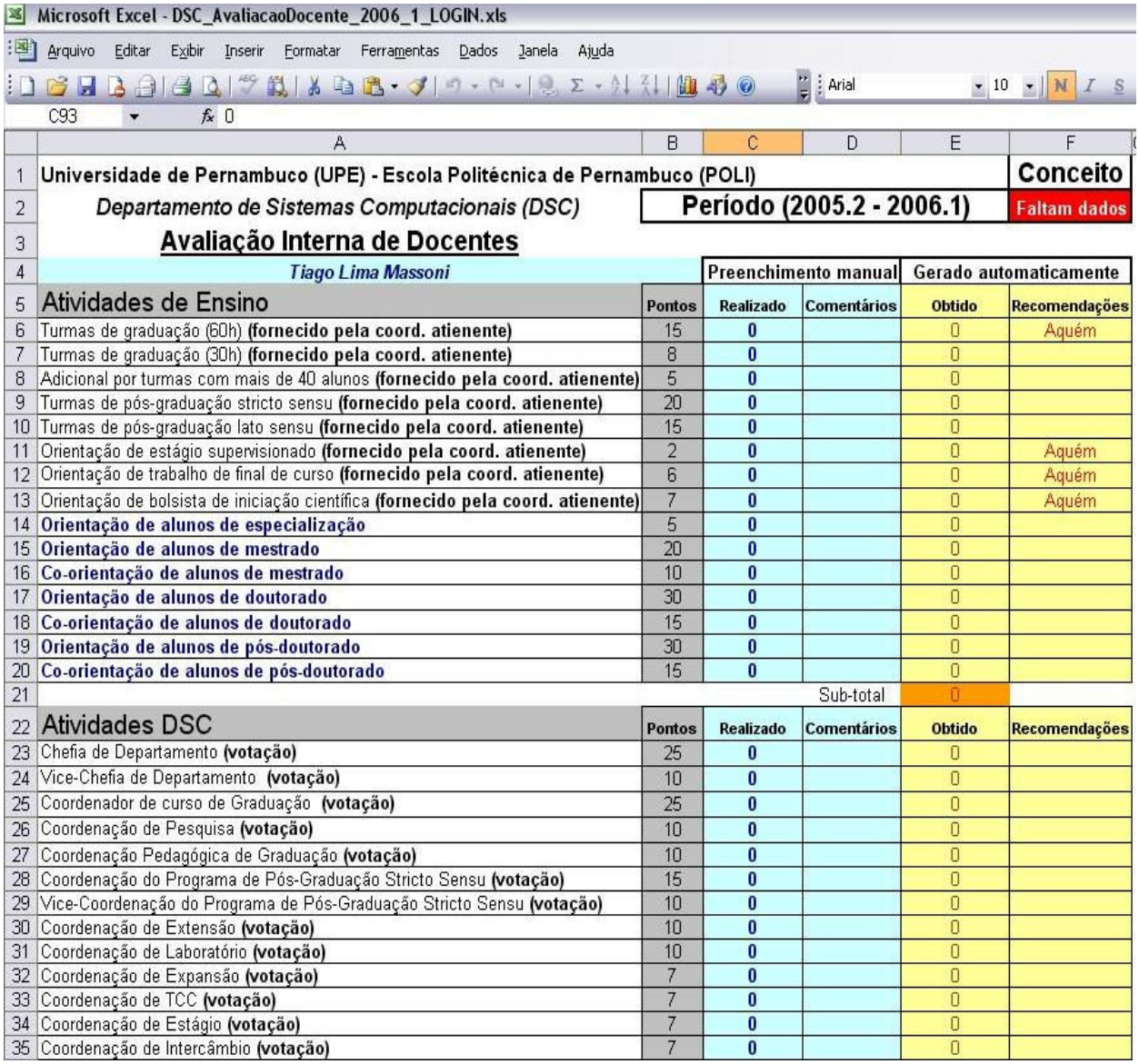
3.1 Estado Atual

Como descrito no Capítulo 1, o DSC tem algumas necessidades. Como se trata de um departamento novo, ele ainda busca criar uma estrutura melhor tanto para os docentes quanto para os discentes. Desde a criação até agora, o número de docentes vem aumentando e a expectativa é que aumente ainda mais, pois com a abertura do mestrado e a busca por outros cursos de pós-graduação como especializações e doutorado, será necessário um maior corpo docente.

Com o crescimento do departamento, e consequentemente do número de docentes, cresceu a dificuldade de se avaliar o desempenho e de controlar as atividades exercidas pelos mesmos. Para avaliação, cada docente do DSC tem uma pontuação mínima a ser atingida em cada semestre através da realização de atividades. Cada atividade tem associada a si um valor, e a soma dos pontos de cada atividade exercida por um docente no semestre é a sua pontuação. As atividades, hoje em dia, são noventa e cinco (95) e variam de ensino de disciplinas até publicações em revistas científicas. Hoje em dia o único modo que o departamento tem de avaliar seus docentes é através de uma planilha, que os docentes preenchem e depois o coordenador visualiza. O que torna a avaliação uma tarefa bastante tediosa. A Figura 4 ilustra uma parte dessa planilha, com as atividades e suas pontuações. Para o coordenador, fica bastante difícil tomar decisões a partir dessas planilhas, pois ele não tem uma visão macro das atividades, ele visualiza apenas um docente por vez.

Outra necessidade que o DSC possui é permitir que os seus docentes se afastem durante um período do departamento para fazer pós-doutorado no exterior. Como todos os seus professores são doutores ou doutorandos, este é o próximo passo para que o nível do departamento cresça ainda mais. O citado departamento precisa que seus docentes estejam em capacitação constante, para que seus acadêmicos ampliem sua formação e possam manter-se atualizados. O departamento, atualmente, não tem nenhuma forma de possibilitar tais saídas, pois não tem como fazer um controle seguro de alocação dos demais professores para reposições das disciplinas do docente que está se ausentando. Também não tem como possibilitar que os professores lecionem algumas disciplinas a mais antes de se ausentar para criar alguns créditos.

Além disso, o pleno do DSC ainda precisa decidir boa parte das regras para as saídas, pois esse é um projeto que está dando ainda seus primeiros passos.



Microsoft Excel - DSC_AvaliacaoDocente_2006_1_LOGIN.xls

Arquivo Editar Exibir Inserir Formatar Ferramentas Dados Janela Ajuda

C93 fx 0

	A	B	C	D	E	F
1	Universidade de Pernambuco (UPE) - Escola Politécnica de Pernambuco (POLI)					Conceito
2	*Departamento de Sistemas Computacionais (DSC)*		Período (2005.2 - 2006.1)			Faltam dados
3	**Avaliação Interna de Docentes**					
4	*Tiago Lima Massoni*		Preenchimento manual		Gerado automaticamente	
5	**Atividades de Ensino**	**Pontos**	**Realizado**	**Comentários**	**Obtido**	**Recomendações**
6	Turmas de graduação (60h) **(fornecido pela coord. atienente)**	15	0		0	Aquém
7	Turmas de graduação (30h) **(fornecido pela coord. atienente)**	8	0		0	
8	Adicional por turmas com mais de 40 alunos **(fornecido pela coord. atienente)**	5	0		0	
9	Turmas de pós-graduação stricto sensu **(fornecido pela coord. atienente)**	20	0		0	
10	Turmas de pós-graduação lato sensu **(fornecido pela coord. atienente)**	15	0		0	
11	Orientação de estágio supervisionado **(fornecido pela coord. atienente)**	2	0		0	Aquém
12	Orientação de trabalho de final de curso **(fornecido pela coord. atienente)**	6	0		0	Aquém
13	Orientação de bolsista de iniciação científica **(fornecido pela coord. atienente)**	7	0		0	Aquém
14	**Orientação de alunos de especialização**	5	0		0	
15	**Orientação de alunos de mestrado**	20	0		0	
16	**Co-orientação de alunos de mestrado**	10	0		0	
17	**Orientação de alunos de doutorado**	30	0		0	
18	**Co-orientação de alunos de doutorado**	15	0		0	
19	**Orientação de alunos de pós-doutorado**	30	0		0	
20	**Co-orientação de alunos de pós-doutorado**	15	0		0	
21				Sub-total	0	
22	**Atividades DSC**	**Pontos**	**Realizado**	**Comentários**	**Obtido**	**Recomendações**
23	Chefia de Departamento **(votação)**	25	0		0	
24	Vice-Chefia de Departamento **(votação)**	10	0		0	
25	Coordenador de curso de Graduação **(votação)**	25	0		0	
26	Coordenação de Pesquisa **(votação)**	10	0		0	
27	Coordenação Pedagógica de Graduação **(votação)**	10	0		0	
28	Coordenação do Programa de Pós-Graduação Stricto Sensu **(votação)**	15	0		0	
29	Vice-Coordenação do Programa de Pós-Graduação Stricto Sensu **(votação)**	10	0		0	
30	Coordenação de Extensão **(votação)**	10	0		0	
31	Coordenação de Laboratório **(votação)**	10	0		0	
32	Coordenação de Expansão **(votação)**	7	0		0	
33	Coordenação de TCC **(votação)**	7	0		0	
34	Coordenação de Estágio **(votação)**	7	0		0	
35	Coordenação de Intercâmbio **(votação)**	7	0		0	

Figura 4. Planilha atual para Avaliação de Professores.

3.2 Objetivos do Sistema

Como dito anteriormente, esse trabalho tem o objetivo de desenvolver dois sistemas de informação integrados em uma única aplicação. Um sistema será o de avaliação de docentes e o outro será o de Banco de Créditos. Ambos os sistemas serão usados para tomada de decisões pelo pleno do DSC. Os sistemas serão desenvolvidos para uso interno do DSC e só terão acesso a eles os docentes, o coordenador, e os funcionários que o coordenador permitir.

O sistema de avaliação permitirá que o coordenador do DSC possa visualizar o comprometimento que cada docente está tendo com o Departamento. Os docentes serão os responsáveis por cadastrar suas atividades, e o coordenador terá a responsabilidade de gerar as avaliações dos docentes de acordo com suas necessidades. Ele terá as opções de escolher o período em que ele quer avaliar os professores e se quer avaliar por um determinado tipo de atividade, ou se quer uma avaliação completa (com todos os tipos de atividades). Através desse sistema o coordenador poderá visualizar tanto se os professores estão atingindo a meta de pontos como se eles estão distribuindo seus pontos entre atividades diversas. Para o departamento é interessante que todas as atividades sejam distribuídas da melhor forma possível entre os docentes. Por exemplo, não é justo que algum docente obtenha a maioria de seus pontos através de participações em bancas de defesas de teses e de Trabalhos de Conclusão de Curso, enquanto ele poderia estar rendendo produção tecnológica e a escrita de artigos, o que divulgaria ainda mais o nome do departamento.

O sistema de banco de créditos será o responsável por organizar a saída dos docentes para pós-doutorado. Ele implementará uma fila dos docentes que estarão aguardando para o afastamento e conterá as informações importantes tanto das saídas já realizadas, como as que ainda acontecerão. Inicialmente o sistema só permitirá que um docente se ausente a cada semestre letivo. Fica a cargo do pleno do DSC a decisão para que mais de um docente possa se afastar do departamento em um mesmo semestre. Os créditos estarão associados à quantidade de disciplinas extras lecionadas pelos docentes, pois como eles vão continuar recebendo enquanto estiverem fora, eles devem lecionar certa quantidade de disciplinas a mais do que sua carga normal antes de se afastarem e outra parte depois que voltar do pós-doutorado. Quando algum docente estiver afastado outros docentes deverão assumir provisoriamente as suas disciplinas. Isto fará com que esses docentes já ganhem créditos para uma possível saída, ou contem como pagamento de créditos caso o docente já tenha se afastado e ainda esteja devendo alguns.

3.3 Requisitos

Os sistemas foram desenvolvidos para gerenciar e facilitar a manutenção de informações dos docentes do DSC, essas informações são baseadas em cinco itens, que formam a base dos sistemas.

3.3.1 Docentes

Os docentes definem o elemento central para a existência do sistema. Todas as funcionalidades que o sistema possui não fariam sentido se eles não existissem. São eles que devem ser avaliados, e que querem se afastar um tempo do departamento para fazer um curso de pós-doutorado. Eles deverão uma vez por semestre cadastrar as atividades exercidas no sistema. Os docentes terão seus cadastros administrados pelo coordenador.

3.3.2 Coordenador

O coordenador também é um docente, porém ele tem responsabilidades a mais. É o coordenador quem fará as solicitações (extra sistema) para que os demais docentes cadastrem

suas atividades durante certo período de tempo, uma vez por semestre. É ele quem poderá cadastrar docentes, atividades, alterar pontuação de atividades e gerenciar a fila de espera para os afastamentos. Também é ele quem fará mais uso do sistema, para geração de avaliações para possíveis tomadas de decisão.

3.3.3 Atividades

Existirá um cadastro de possíveis atividades a serem executadas pelos docentes. Esse cadastro deverá ser gerenciado pelo coordenador. Cada atividade tem a ela associada um valor, e é a soma dos valores das atividades exercidas por um docente em um semestre que dará sua pontuação neste semestre quando o coordenador for avaliá-lo. A pontuação de cada atividade é configurável e poderá ter seu valor mudado de um semestre para o outro nas avaliações.

As atividades inicialmente cadastradas são noventa e cinco divididas em oito tipos de atividades. Atividades de ensino são atividades como ensino de disciplinas e orientação de alunos, seja em trabalho de final de curso, especialização, mestrado etc. As atividades DSC são atividades relacionadas aos cargos exercidos pelos docentes dentro do departamento, como chefia de departamento ou coordenação de extensão. As atividades de pesquisa e extensão dizem respeito às coordenações e participações em projetos de pesquisa e extensão. Em Produção Tecnológica estão atividades como construção de protótipos e produção de software. As atividades de Qualificação são: participação como aluno de pós-graduação *stricto sensu* e estágio de pós-doutoramento. Em Outras Atividades encontramos atividades como participação em bancas de exame. As atividades de produção intelectual e cientifica são aquelas relacionadas a participações em eventos e publicações científicas. Já as atividades administrativas, são atividades como Diretoria e Vice-diretoria. A Tabela 1 mostra como as atividades estão divididas.

Tabela 1. Quantidade de atividades por tipos de atividades.

Tipos de Atividades	**Quantidade**
Atividades de Ensino	15
Atividades DSC	24
Atividades de Pesquisa e Extensão	10
Produção Tecnológica	3
Qualificação	2
Outras Atividades	8
Produção Intelectual e cientifica (p/coordenador de pesquisa)	23
Atividades Administrativas	10

3.3.4 Créditos

Os créditos são a base para o banco de créditos. Os créditos no sistema de banco de créditos funcionam de forma análoga às horas trabalhadas em banco de horas de instituições que adotam essa solução.

As atividades de ensino de turmas de graduação e pós-graduação valerão como créditos para os docentes. Cada docente tem que lecionar certa quantidade de turmas por semestre; as turmas que ele lecionar a mais do que essa quantidade contarão como créditos no banco para registro de algum afastamento seu.

3.3.5 Fila de Espera

O sistema terá um fila de docentes que estão com seus afastamentos agendados. O coordenador será o responsável pelo preenchimento da fila. Para entrar na fila o docente deve apresentar sua proposta ao coordenador, com um possível cenário definido para suas disciplinas antes e durante o seu pós-doutorado. O desempate nessa fila será definido pela avaliação dos docentes. Após a visualização dos resultados obtidos pelos professores em avaliações, o coordenador terá a tarefa de indicar qual docente tem prioridade de se afastar em um determinado semestre, caso nesse semestre tenha mais de um docente querendo afastamento.

3.4 Casos de Uso

Nesta seção falaremos um pouco sobre os casos de uso do sistema e iremos descrever detalhadamente alguns deles. Alguns autores dizem que um caso de uso define um contrato de como o sistema vai se comportar [10].

Para levantamento dos requisitos funcionais, inicialmente foi realizada uma reunião com os professores Tiago Massoni, Márcio Lopes e com o coordenador do departamento Abel Guilhermino. Durante as iterações (ver capítulo 4) com o professor Tiago, ocorreram algumas alterações, até que foram levantados todos os requisitos funcionais do sistema. Através desse levantamento criamos o diagrama de casos de uso. O diagrama de casos de uso é um dos diagramas descritos na linguagem UML[6]. UML é uma linguagem para especificação, documentação e visualização em desenvolvimento de sistemas orientados a objetos.

Um ator de um sistema é alguém ou alguma coisa (ex: máquina, outro sistema), externa à aplicação, que interage de alguma forma com o sistema [11] . Nossos sistemas possuem três atores; Professor, Coordenador e Avaliador. O Coordenador é o único ator que tem o privilégio de utilizar todos os casos de uso, e como indicado no diagrama, o Coordenador também é um professor. Isso nos diz que todos os casos de uso destinados aos professores também poderão ser utilizados pelo Coordenador.

A Figura 5 ilustra os atores do sistema e seus casos de uso. Para exemplo do funcionamento dos sistemas, vamos detalhar alguns dos casos de uso que estão ilustrados no diagrama.

[6] http://www.uml.org/

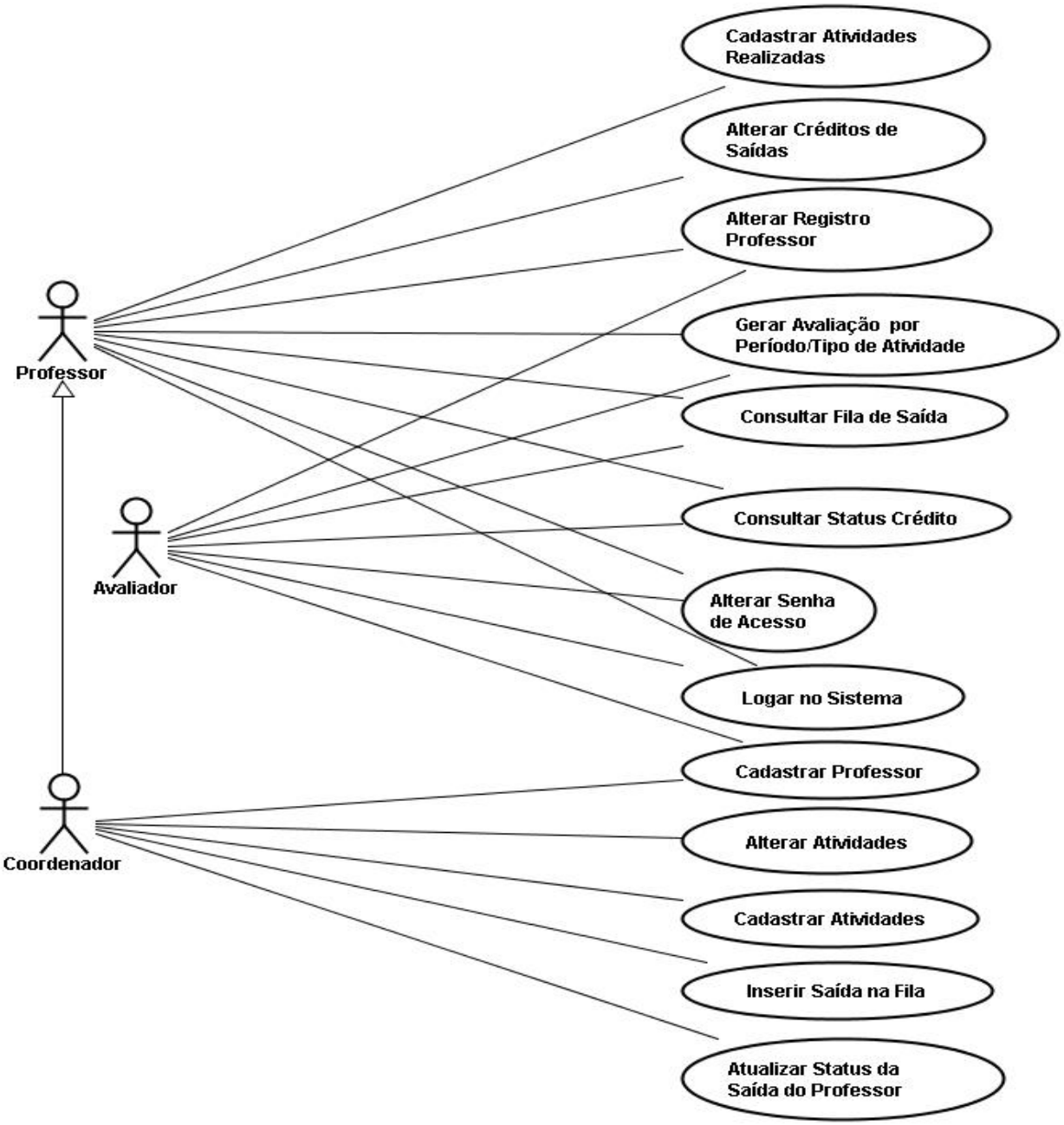


Figura 5. Diagrama de Casos de Uso.

Caso de Uso: Gerar Avaliação Por Tipo de Atividade

Descrição: Caso de uso responsável pela geração de uma avaliação de docentes em um tipo de atividade durante um determinado período.

Ator Primário: Docente, Coordenador.

Pré-condições: Docente está identificado e autenticado, e as atividades e seus tipos estarem devidamente cadastrados.

Pós-condições: Nenhuma.

Fluxo Básico (Cenário de Sucesso):

1. O docente inicia a geração da avaliação.

2. Uma tela é mostrada ao docente para que ele escolha o semestre inicial e o final de sua consulta, além de que para qual tipo de atividade ele quer gerar a avaliação.

3. Depois da escolha dos parâmetros, o docente solicita a consulta.

4. A tela com o resultado da avaliação é exibida para o docente com as atividades do tipo escolhido sendo exibidas, com suas pontuações e o total da pontuação para cada semestre escolhido na consulta.

Fluxos Alternativos:

1. O docente ainda não cadastrou nenhuma atividade sua no período que ele escolheu na consulta.
    1. É exibida para o docente uma tela indicando o erro e com opções para o docente fazer uma nova consulta ou para voltar à página principal.
        1.1 O docente opta por fazer uma nova consulta.
            1. O docente volta ao passo dois do fluxo básico.
        1.2 O docente opta por cancelar a operação.
            1. A página principal é exibida para o docente.

Caso de Uso: Cadastrar Atividades

Descrição: Caso de uso responsável por inserir novas atividades no bando de dados. Atividade essa que poderá ser realizada pelos docentes.

Ator Primário: Coordenador

Pré-condições: Coordenador está identificado e autenticado e os tipos de atividades estarem devidamente cadastrados.

Pós-condições: O registro da atividade é devidamente salvo no banco de dados.

Fluxo Básico (Cenário de Sucesso):

1. O coordenador inicia a inserção da atividade.

2. Uma tela é mostrada ao coordenador para que ele escolha o tipo de atividade a que a nova atividade estará vinculada. O coordenador deverá também dar um nome à nova atividade, um valor para ela e opcionalmente uma descrição da mesma.

3. Depois do preenchimento do formulário, o coordenador solicita a inserção da atividade.

4. A tela de confirmação da inserção é exibida para o coordenador.

Caso de Uso: Inserir Saída na Fila

Descrição: Caso de uso responsável por inserir um docente na fila de saída para o pós-doutorado.

Ator Primário: Coordenador

Pré-condições: Coordenador está identificado e autenticado. O docente ter apresentado ao coordenador sua proposta com um possível cenário definido para suas disciplinas antes e durante o seu afastamento.

Pós-condições: O registro da saída é devidamente salvo no banco de dados. O docente passa a ter créditos relacionados com essa saída.

Fluxo Básico (Cenário de Sucesso):

1. O coordenador inicia a inserção da saída.

2. Uma tela é mostrada ao coordenador para que ele escolha o docente que estará vinculado à nova saída. O coordenador deverá também selecionar o semestre em que o docente ficará ausente. Se o docente já tiver alguns créditos adquiridos, eles podem ser registrados agora.

3. Depois do preenchimento do formulário, o coordenador solicita a inserção da saída no banco de dados.

4. A tela de confirmação da saída é exibida para o coordenador.

3.5 Modelo de Domínio

Um modelo de domínio é a representação visual de conceitos e situações analisadas em um domínio de interesse. Esta representação é feita através de classes conceituais que são uma visão de mais alto nível do pensamento em objetos.

No UP, o termo “Modelo de Domínio” significa a representação de uma situação real através de classes conceituais, não de objetos de software. O termo não significa um conjunto de diagramas descrevendo as classes do software, nem a camada de domínio de uma arquitetura, ou objetos de softwares com responsabilidades [2].

Usando a notação UML, o modelo de domínio é ilustrado com um conjunto de classes sem assinaturas de métodos estarem definidas. Ele demonstra uma perspectiva conceitual, e deve mostrar:

- Objetos de domínios ou classes conceituais;
- Associações entre as classes conceituais e
- Atributos das classes conceituais.

A Figura 6 ilustra um modelo de domínio dos Sistemas de Banco de Créditos e Avaliação de Docentes. Onde podemos ver que cada avaliação é feita para um docente e esse docente realiza várias atividades. Cada atividade realizada pelo docente tem a quantidade de pontos associada a ela e a quantidade de vezes que o docente realizou. Cada avaliação de docente possui o semestre avaliado e a quantidade de pontos obtida. A figura também mostra que na fila de espera se tem vários docentes, e para cada docente existe associado o semestre em que ele vai sair e a quantidade de créditos já pagas por ele.

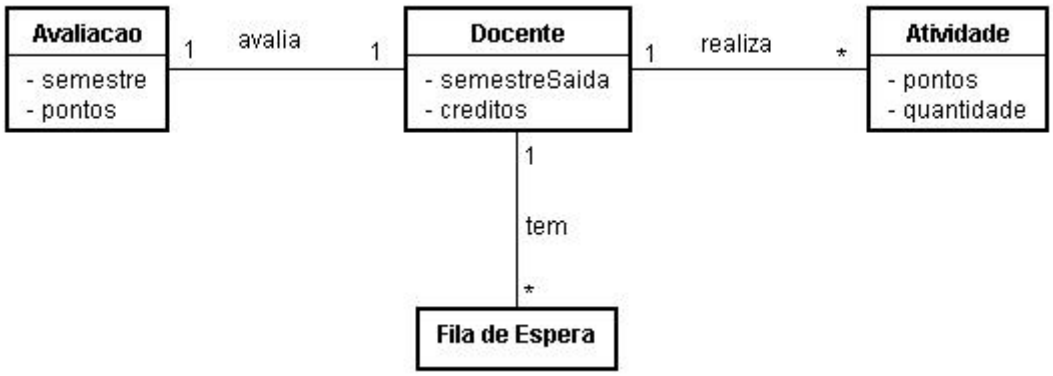


Figura 6. Modelo de domínio para os Sistemas de Banco de Créditos e Avaliação de Docentes.

Capítulo 4

Projeto e Implementação

Neste capítulo descrevemos detalhes de como foram projetados os sistemas e da implementação dos mesmos. Como os nossos sistemas são aplicações web, decidimos por usar o Modelo MVC 2, através do *framework* Struts. Outra particularidade da arquitetura dos sistemas é que eles foram desenvolvidos em camadas.

Os sistemas foram implementados através de um desenvolvimento iterativo usando a linguagem Java[7]. Ela foi escolhida por ser uma linguagem orientada a objetos, que hoje em dia é o paradigma mais usado em desenvolvimento de softwares, além de ser altamente portável, ou seja, funciona em diversos sistemas operacionais. Além das classes (arquivos) Java, fizemos uso de artefatos comuns para qualquer aplicação web, como a linguagem Javascript e CSS.

O uso de Javascript é bastante útil em aplicações web, pois seu código é todo executado no navegador (lado cliente da aplicação). Através dele podemos fazer várias validações em formulários antes de submetê-los via rede. Outra vantagem é interação com a página, tornando o conteúdo mais dinâmico. A sua sintaxe é bastante semelhante a Java, porém o seu uso é totalmente diferente.

O uso de CSS é bastante encorajado também pelos desenvolvedores web, é através dele que formatamos nossas páginas (HTML, JSP, ASP) dando a ela cores nos seus componentes, mudanças de fundos, alterações nos tamanhos e tipos de fontes utilizadas, etc.

4.1 Struts e o MVC

Os desenvolvedores de hoje precisam construir aplicações com diversos recursos e que podem ser mantidas com o tempo. Os *frameworks* de aplicação web, como o Struts, resolvem problemas comuns, portanto os desenvolvedores podem se concentrar na funcionalidade exclusiva de sua aplicação. Os *frameworks* são especialmente importantes ao construir aplicações web, uma vez que o modo como o HTTP e a HTML funcionam torna mais difícil criar aplicações dinâmicas, pois o HTTP transfere basicamente os dados usando campos de textos simples. Transferir dados binários requer o uso de uma extensão complicada para o protocolo. Já a

[7] http://java.sun.com/

linguagem HTML não pode construir muitos elementos de interface encontrados nos ambientes da área de trabalho [8] .

Uma das coisas que os desenvolvedores do Struts defendem sobre ele é que o *framework "...encoraja as arquiteturas de aplicação baseadas na abordagem Modelo 2, uma variação do clássico paradigma de projeto Model-View-Controller (MVC)*" [8]

Nas aplicações web em camadas, a construção difere um pouco do MVC convencional. O controlador passa a funcionar como um elo de ligação entre as camadas de apresentação (*View*) e de negócios (*Model*). Cada componente continua com suas mesmas responsabilidades, porém o fluxo muda um pouco. Qualquer consulta de estado ou notificação de mudança deve passar pelo controlador. E quando a camada de apresentação mostra o conteúdo dinâmico, usa os dados transmitidos pelo controle em vez dos retornados diretamente da camada de negócios.

O Struts implementa o Modelo 2 através de um *servlet* controlador que pode gerenciar o fluxo entre as páginas JSP e os outros componentes da *View*. Já o MVC é implementado através do uso de ActionForwards e ActionMappings (indicados na Tabela 2) que mantém o controle do fluxo fora da *View*. Nas páginas JSPs podemos passar os caminhos lógicos que o controlador se encarregar de mapear nas URIs reais. A Tabela 2 faz um relacionamento entre as classes básicas do Struts e as funcionalidades dos componentes numa arquitetura MVC. A Figura 7 ilustra o fluxo básico de uma requisição no *framework* Struts. Façamos uma descrição da solicitação-resposta. Os números entre parênteses referem-se à Figura 7.

- Um cliente solicita um caminho (1).
- Se o mapeamento especificar um componente do formulário, o ActionServlet verá se já existe um , ou criará um (1.1).
- Se o mapeamento tiver a propriedade *validate* definida para *true*; chamará *validate* no componente do formulário (1.2).
- Se o mapeamento especificar um tipo Action, será reutilizado se já existir um ou será instanciado. (1.3).
- O Action pode pega as propriedades do formulário (1.3.1), chama os métodos de negócio (1.3.2) e preenche os componentes de classes auxiliares (1.3.3) e do formulário (1.3.4).
- O Action retorna um ActionForward para oActionServlet (1.3.5).
- Se o ActionForward for para outro Action, começaremos tudo de novo; do contrário, será enviado para uma página de exibição ou algum outro recurso (2,3) .

Tabela 2. As classes Struts básicas como se relacionam com o MVC [8] .

Classes	Descrição
ActionForward	Uma ação do usuário ou seleção da exibição.
ActionForm	Os dados para alteração de um estado.
ActionMapping	O evento de alteração de estado.
ActionServlet	A parte do controle que recebe as ações do usuário, as alterações de estado e envia as seleções de exibição.
Classes Action	A parte do controle que interage com o modelo para executar uma alteração do estado ou consulta e avisa o ActionServlet sobre a próxima exibição a selecionar.

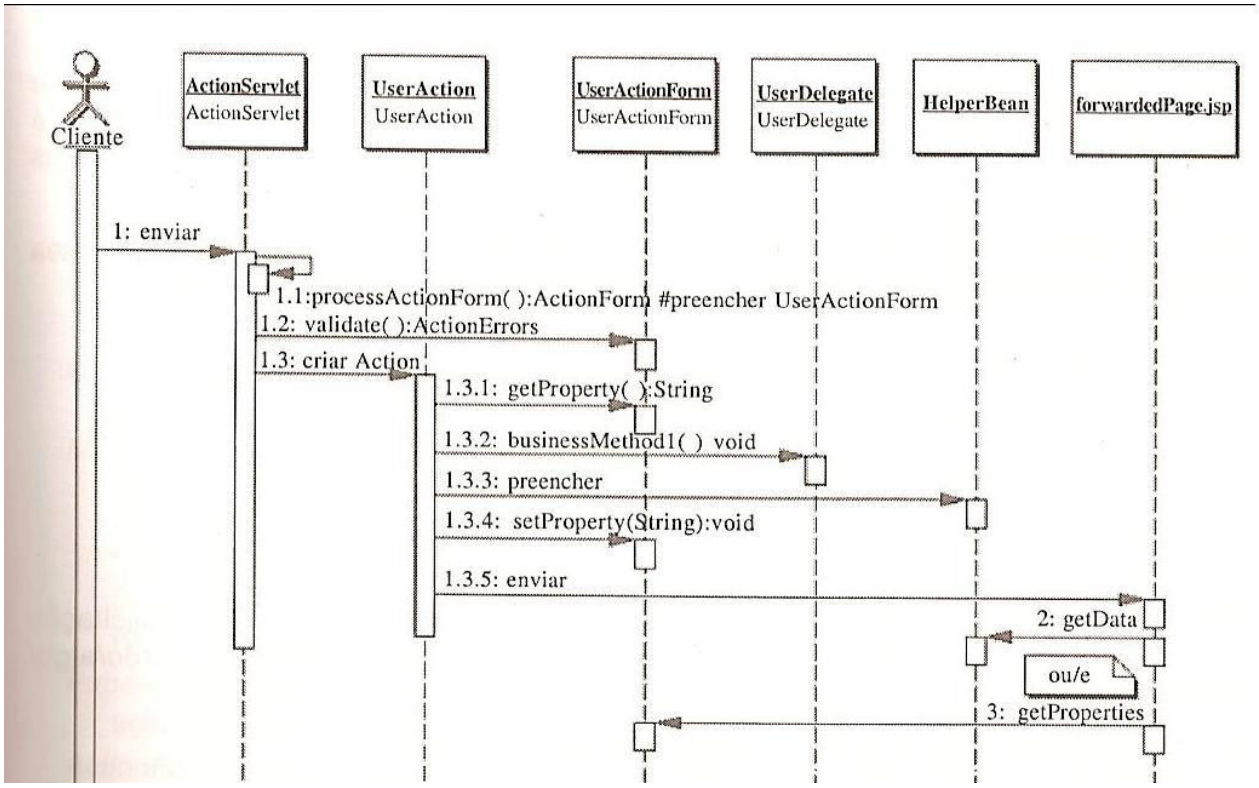


Figura 7. O processo de solicitação-resposta do Struts. UML de Jean-Michael Garnier [8] .

Além das classes básicas, o Struts possui arquivos de configuração. Entre eles um de muita importância é o *struts-config.xml*. É nesse arquivo onde declaramos os nossas classes Actions e ActionsForms e fazemos mapeamentos entre as camadas de controle e a de visão. Este arquivo armazena a configuração padrão para os objetos do controlador, que inclui as ações do usuário, as alterações de estado e as consultas de estados suportadas por esse modelo.

Uma parte do *struts-config.xml* da nossa aplicação está ilustrada na Figura 8. A seção *<form-beans></form-beans>* é onde declaramos os nossos formulários (ActionForm) que usaremos nas telas, através da *tag <form-bean/>*. Na seção *<action-mappings><action-mappings/>* é onde declaramos nossas ações (Classes Action) e seus caminhos, através da *tag <action/>*. Dentro da *tag* <action/>, podemos encontrar a *tag* <forward/> que indica os possíveis encaminhamentos (ActionForward) dessa ação. Onde também podemos encontrar encaminhamentos, é dentro da seção *<global-forwards><global-forwards/>*, porém esses podem ser usados por todas as ações do sistema.

4.2 Projeto em Camadas

Uma das principais vantagens do uso do paradigma de programação orientada a objetos é a abstração. Cada classe só deve saber quais os métodos que as outras classes provêem. Elas não devem saber quais computações são feitas no interior desses métodos, nem devem acessar livremente as propriedades de outras classes.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config (View Source for full doctype...)>
- <struts-config>
    <data-sources />
  - <form-beans>
      <form-bean name="LoginForm" type="LoginForm" />
      <form-bean name="AvaliarForm" type="avaliacao.AvaliarForm" />
      <form-bean name="IncluirAtividadeDocenteForm" type="atividade.IncluirAtividadeDocenteForm" />
      <form-bean name="IncluirAtividadeForm" type="atividade.IncluirAtividadeForm" />
      <form-bean name="AlterarAtividadeForm" type="atividade.AlterarAtividadeForm" />
      <form-bean name="IncluirDocenteForm" type="recursosHumanos.IncluirDocenteForm" />
      <form-bean name="AlterarDocenteForm" type="recursosHumanos.AlterarDocenteForm" />
      <form-bean name="AlterarCreditosDocenteForm" type="saida.AlterarCreditosDocenteForm" />
      <form-bean name="InserirSaidaForm" type="saida.InserirSaidaForm" />
      <form-bean name="AlterarSaidaForm" type="saida.AlterarSaidaForm" />
    </form-beans>
    <global-exceptions />
  + <global-forwards>
  - <action-mappings>
      <action name="LoginForm" path="/login" scope="request" type="LoginAction" />
    - <action name="AvaliarForm" path="/avaliar" scope="request" type="avaliacao.AvaliarAction">
        <forward name="sucessoCompleta" path="/jsp/avaliacao/sucessoAvaliacaoCompleta.jsp" />
        <forward name="sucessoPorTipoAtividade" path="/jsp/avaliacao/suscessoAvaliacaoPorTipoAtividade.jsp" />
        <forward name="erro" path="/jsp/avaliacao/erroAvaliacao.jsp" />
      </action>
    + <action name="IncluirAtividadeDocenteForm" path="/incluirAtividadeDocente" scope="request" type="atividade.IncluirAtividadeDocenteAction">
      <action name="IncluirAtividadeForm" path="/incluirAtividade" scope="request" type="atividade.IncluirAtividadeAction" />
      <action name="AlterarAtividadeForm" path="/alterarAtividade" scope="request" type="atividade.AlterarAtividadeAction" />
      <action name="IncluirDocenteForm" path="/incluirDocente" scope="request" type="recursosHumanos.IncluirDocenteAction" />
      <action name="AlterarDocenteForm" path="/alterarDocente" scope="request" type="recursosHumanos.AlterarDocenteAction" />
      <action name="AlterarCreditosDocenteForm" path="/alterarCreditosDocente" scope="request" type="saida.AlterarCreditosDocenteAction" />
      <action name="InserirSaidaForm" path="/inserirSaida" scope="request" type="saida.InserirSaidaAction" />
      <action name="AlterarSaidaForm" path="/alterarSaida" scope="request" type="saida.AlterarSaidaAction" />
    </action-mappings>
    <controller />
</struts-config>
```

Figura 8. Exemplo de *struts-config.xml*.

Nos nossos sistemas de Avaliação de Docentes e Banco de Créditos, decidimos por usar a programação em camadas na parte de negócios (ou modelo no MVC). Esse tipo de programação se utiliza da idéia de abstração. Os principais benefícios que tal arquitetura traz são modularidade, maior coesão, reusabilidade e extensibilidade. Além disso, a programação em camadas facilita muito a manutenção dos softwares, pois o sistema fica mais bem estruturado. O que o torna de mais fácil entendimento pelas pessoas que vão mantê-lo. Hoje em dia a maior parte dos custos de um software é com a manutenção. As mudanças em uma camada não afetam em nada as outras, desde que as interfaces (contratos) sejam mantidas.

O propósito geral de cada camada da arquitetura em camadas é fornecer serviços para a camada superior, abstraindo a mesma dos detalhes específicos sobre os serviços que estão sendo fornecidos. Então, cada camada deve se preocupar com os detalhes importantes para o seu domínio e que devem ser escondidos da camada superior. Cada camada deve implementar mecanismos que permitam que a camada superior não precise saber dos detalhes de implementação da camada inferior.

Os sistemas também se utilizam do padrão de projeto *Singleton*[8] na classe Fachada (é através dessa classe o único modo de interagir com os sistemas), que indica que apenas uma instância dessa classe existe. Ou seja, antes de se criar uma instancia verifica-se se já existe uma criada.

[8] http://www.javaworld.com/javaworld/jw-04-2003/jw-0425-designpatterns.html

As camadas de negócios dos nossos sistemas são divididas em Fachada, Controlador, Cadastro e Repositório (ver Figura 9). A fachada é a classe *singleton* que apenas inicializa os controladores. Os controladores é que funcionam como interface com o mundo externo. Para cada pacote da aplicação existe um controlador (o sistema é dividido em subsistemas, chamados pacotes, para melhor organização e tornar mais fácil o entendimento do sistema para manutenções). É o controlador quem diz quais as ações que podem ser tomadas com os dados contidos nesse pacote. O controlador chama os métodos dos cadastros. Os cadastros possuem um repositório de dados. O repositório é uma interface, ela possui apenas a assinatura dos métodos que acessarão a base de dados, e de acordo com como você terá o acesso aos dados você deve criar uma classe que implemente o repositório e acesse os dados. No nosso caso, como todos os dados são guardados em banco de dados, criamos as classes RepositorioBDR (BDR faz alusão a Banco De Dados Relacional). Caso o acesso a alguns dados do sistema venha a ser feito de outro modo, como arquivos XML, arquivos TXT, ou até através da rede, basta apenas criar um novo tipo de repositório que implemente a interface Repositório e nada mais. Das classes de cadastro para cima, não é preciso modificar nada. A Figura 9 ilustra como as camadas de negócio dos sistemas estão divididas. As setas entre as classes indica que as classes das camadas superiores possuem uma instancia das classes imediatamente abaixo delas.

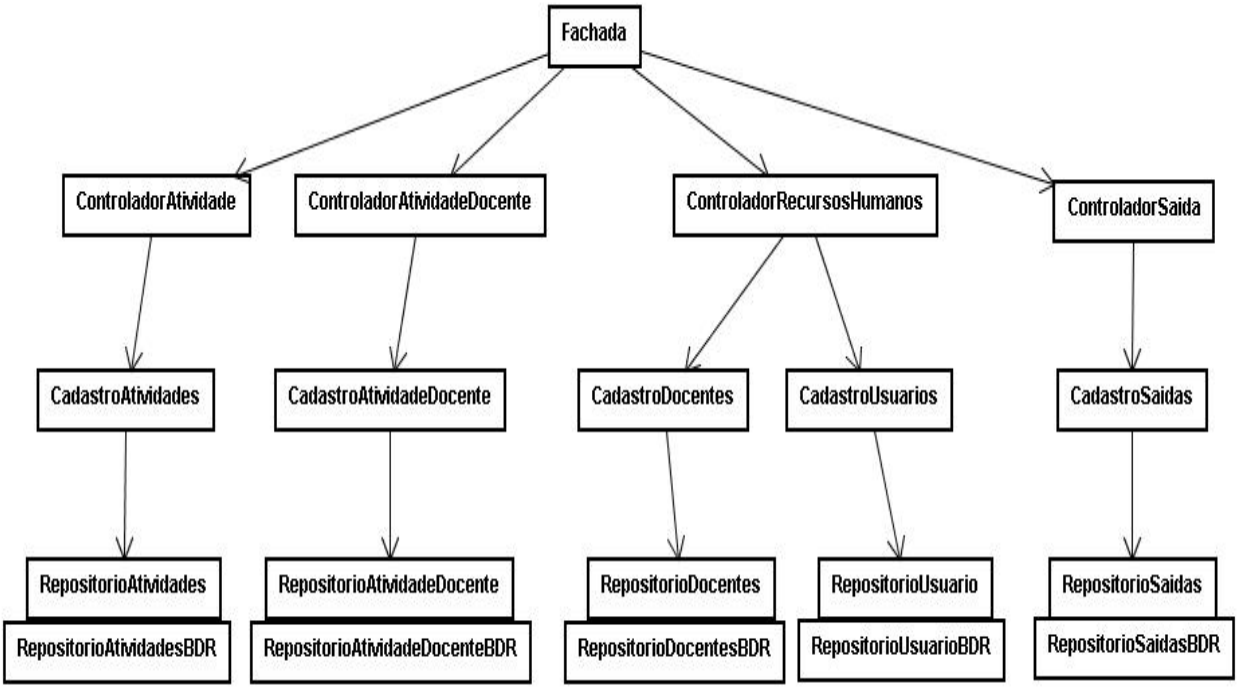


Figura 9. Representação da camada de negócio do sistema e suas subcamadas.

4.3 Acesso aos Dados

Os dados dos sistemas são guardados em um banco de dados MySQL [9]. O MySQL é um Sistema Gerenciado de Banco de Dados (SGBD), que utiliza a linguagem SQL (*Structured Query*

[9] http://www.mysqlbrasil.com.br/

Language – Linguagem de consulta estruturada). SQL é a linguagem utilizada pelos principais SGBDs. Ela se diferencia de outras linguagens de consulta a banco de dados no sentido em que uma consulta SQL especifica a forma do resultado e não o caminho para chegar a ele. Ela é uma linguagem declarativa em oposição a outras linguagens procedurais.

Um banco de dados é uma coleção integrada de dados. Um sistema de banco de dados envolve os próprios dados, o hardware em que os dados residem, o software que controla o armazenamento e a recuperação deles (SGBD) e os próprios usuários [9] . Dentre as várias vantagens de se usar um sistema de banco de dados, podemos citar como as mais importantes:

- A redundância pode ser reduzida – Em sistemas sem banco de dados, cada aplicativo distinto mantém seus próprios arquivos, frequentemente com redundância considerável e uma variedade de formatos físicos. Em sistemas de banco de dados, a redundância é reduzida integrando arquivos separados.
- Os dados podem ser compartilhados – Aplicativos separados podem acessar a mesma base de dados.
- Restrições de segurança podem ser aplicadas – Os sistemas de banco de dados devem ser projetados com controles de acesso sofisticado.

Java possui uma API (*Application Programming Interface* – Interface de Programação de Aplicativos), para acesso a bancos de dados relacionais, chamada JDBC (*Java Database Connectivity*). [10] APIs são conjuntos de classes que provêem serviços que são requisitados a ele por alguns sistemas computacionais.

JDBC provê interfaces, de acesso a banco de dados, mas cada SGBD tem sua própria linguagem interna. Então, cada banco tem sua própria implementação dessas interfaces. E são os próprios fabricantes dos bancos de dados que fornecem essas implementações nos seus *sites* na internet. Ao conjunto de classes nativas de um banco que implementam as interfaces e dão suporte a JDBC chamamos de *Driver*. Toda aplicação Java que precise acessar um banco de dados deverá utilizar um *Driver* daquele banco para realizar a comunicação [11] .

Nos nossos sistemas as classes que acessam os dados são as classes RepositórioBDR (implementações dos Repositórios para acesso ao banco de dados MySQL) e para cada conexão com o banco essas classes devem seguir uns passos, que estão listados abaixo.

1. Carregar e Registrar o *Driver* JDBC.
2. Estabelecer uma conexão com o banco de dados.
3. Submeter a execução de comandos SQL.
4. Ler os resultados.
5. Fechar a conexão.

A Figura 11 ilustra o modelo do banco de dados dos Sistemas de Avaliação de Docentes e Banco de Créditos. As principais tabelas de nosso modelo são: atividade, docente e saída. A notação usada na Figura 10 indica que a tabela atividade se comunica à tabela atividade_docente através do código da atividade. Ou seja, cada registro de atividade_docente estará ligado a uma atividade.

[10] http://java.sun.com/javase/technologies/database/

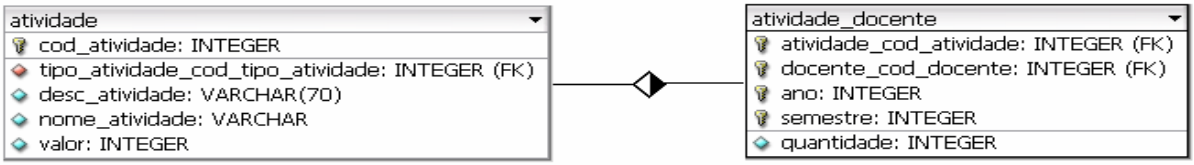


Figura 10. Relação entre as tabelas atividade e atividade_docente.

Como podemos perceber, a tabela "atividade" tem um tipo de atividade que vem de sua ligação com a tabela "tipo_atividade", e a tabela "saida" tem um docente. O ponto principal do modelo é a tabela "atividade_docente". Ela foi criada para guardar registros de atividades realizadas por docentes. Ela se fez necessário porque cada docente pode realizar várias atividades, e cada atividade pode ser realizada por vários docentes. Então, essa tabela tem o registro de cada atividade realizada por cada docente.

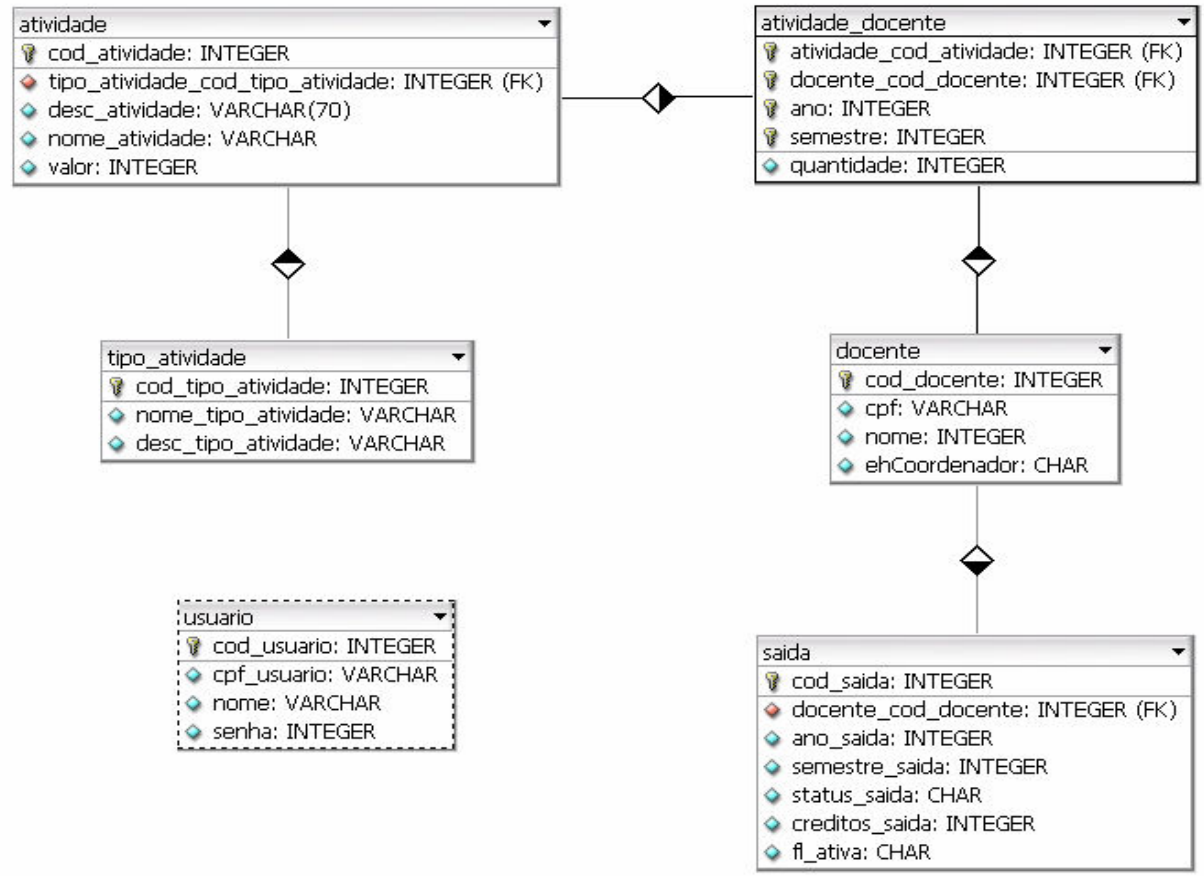


Figura 11. Modelo do banco de dados do Sistema.

4.4 Iterações

Os nossos sistemas foram desenvolvidos utilizando o modelo de desenvolvimento iterativo, que foi descrito na seção 2.4. Uma decisão de projeto que tomamos foi que teríamos 3 iterações ao longo do desenvolvimento. Fizemos uma divisão dos requisitos identificados inicialmente entre as iterações, para entregas de sistemas parciais. Como de praxe nesse modelo de desenvolvimento, alguns dos requisitos identificados inicialmente perderam seu sentido e apareceram outros na medida em que o desenvolvimento foi passando, além de alguns outros que sofrerem algumas mudanças. Constatamos que se tivéssemos detalhados os requisitos iniciais além de prepararmos toda a arquitetura e modelagem em cima do que foi identificado no inicio, teríamos desperdiçado muito tempo em vão.

Certamente teríamos perdido muito tempo com o caso de uso "Sincronização com fontes externas", que seria o de sincronizar os nossos sistemas com a base de dados do SIGA[11], da plataforma lattes[12] e com a WebQualis[13]. Esse caso de uso teve que ser esquecido pelo menos por enquanto devido a problemas burocráticos. Já foi feita a solicitação junto à direção da POLI para que entrassem em contato com as pessoas responsáveis por estas bases de dados. Porém a direção ainda não conseguiu realizar os procedimentos necessários para que fossem disponibilizados esses dados. Esses problemas foram identificados na fase de elaboração como o principal risco do projeto. Tal caso de uso deverá ser implementado no futuro e melhorará bastante os sistemas. Muitas das atividades não precisarão mais ser inseridas pelos docentes. Eles deverão apenas confirmar os dados que virão de fontes externas.

A Tabela 3 ilustra como o desenvolvimento dos casos de uso foi dividido pelas iterações do projeto. Lembrando que alguns casos de uso sofreram algumas alterações nas iterações que sucederam as de seu início.

Tabela 3. Casos de Uso e iterações.

Iteração	Caso de Uso
1ª Iteração	Cadastrar Atividades Realizadas
	Gerar Avaliação por Período/ Tipo de Atividade
2ª Iteração	Cadastrar Professor
	Alterar Registro do Professor
	Cadastrar Atividades
	Alterar Atividades
3ª Iteração	Consultar Fila de Saída
	Consultar Status Crédito
	Alterar Senha de Acesso
	Logar No Sistema
	Inserir Saída na Fila
	Atualizar Status da Saída do Professor
	Alterar Créditos de saídas

[11] http://www.siga.poli.br/poli/principal.jsp

[12] http://lattes.cnpq.br/

[13] http://qualis.capes.gov.br/webqualis/

4.5 Diferentes visões

Nos nossos sistemas teremos três tipos de usuários, os professores (docentes), o coordenador e os avaliadores, que deverão ter seu tipo identificado no momento em que eles se logarem. Sempre lembrando que um coordenador também é um professor, apenas com alguns privilégios a mais. Consequentemente, nem todos os usuários terão a visão ampla do sistema.

O único usuário que terá visão de todo o sistema é o coordenador do departamento. No sistema de avaliação de docentes, os docentes terão visão ampla sobre os seus dados. Podendo modificar o seu registro e senha. O docente é quem deverá cadastrar as suas atividades realizadas. Ele poderá também gerar avaliações pessoais através dos parâmetros passados por ele.

A Figura 12 demonstra como funciona o fluxo interno durante a requisição de inserção de atividades de um docente. O Action-Servlet recebe a requisição e encaminha para o IncluirAtividadeAction (Action), o Action em seguida pega as propriedades da inserção no IncluirAtividadeForm (ActionForm) e depois acessa a camada de negócios passando um objeto atividadeDocente como parâmetro. Um objeto atividadeDocente possui um docente, uma atividade, um semestre e a quantidade de vezes em que o docente realizou essa atividade. A requisição vai "descendo" pela camada de negócio até chegar ao repositorioAtividadeDocenteBDR que é a classe responsável por inserir a atividadeDocente passada como parâmetro no banco de dados. A Figura 13 ilustra uma tela de entrada de dados para esse mesmo caso de uso.

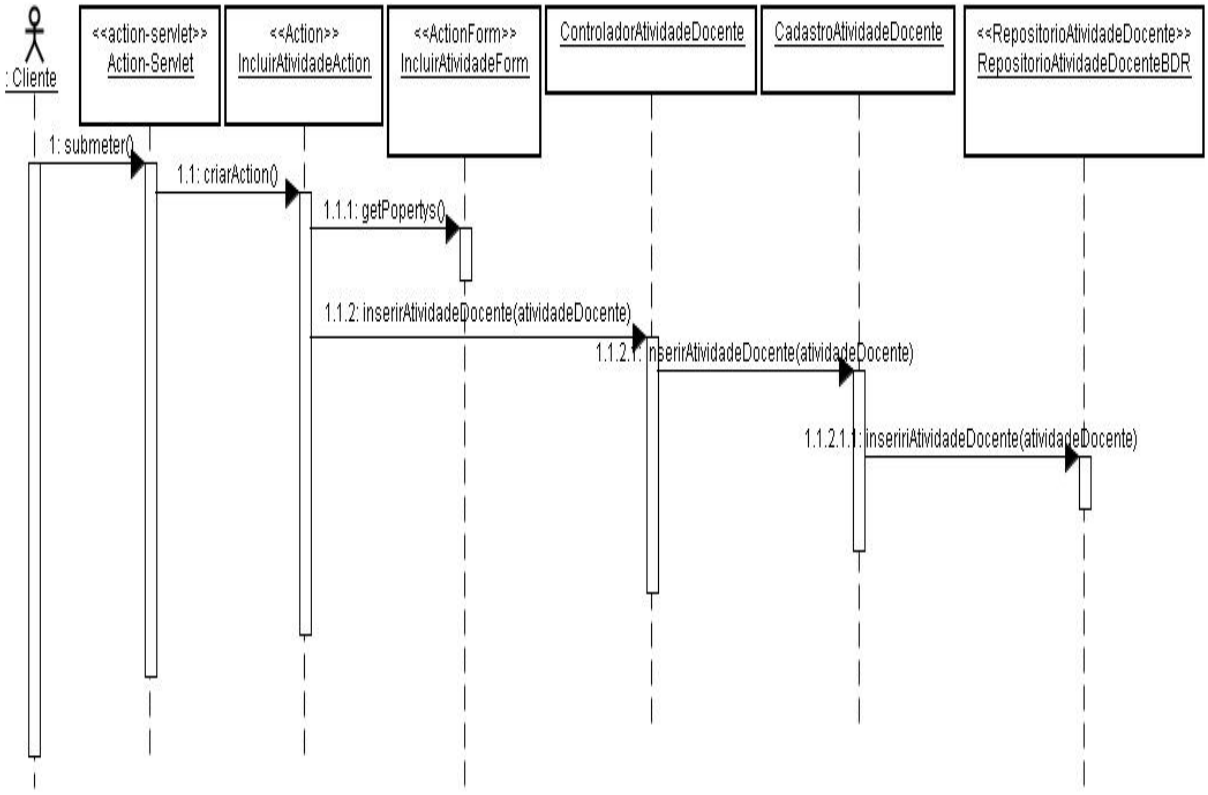


Figura 12. Diagrama de Seqüência do caso de uso de Cadastrar Atividades Realizadas.

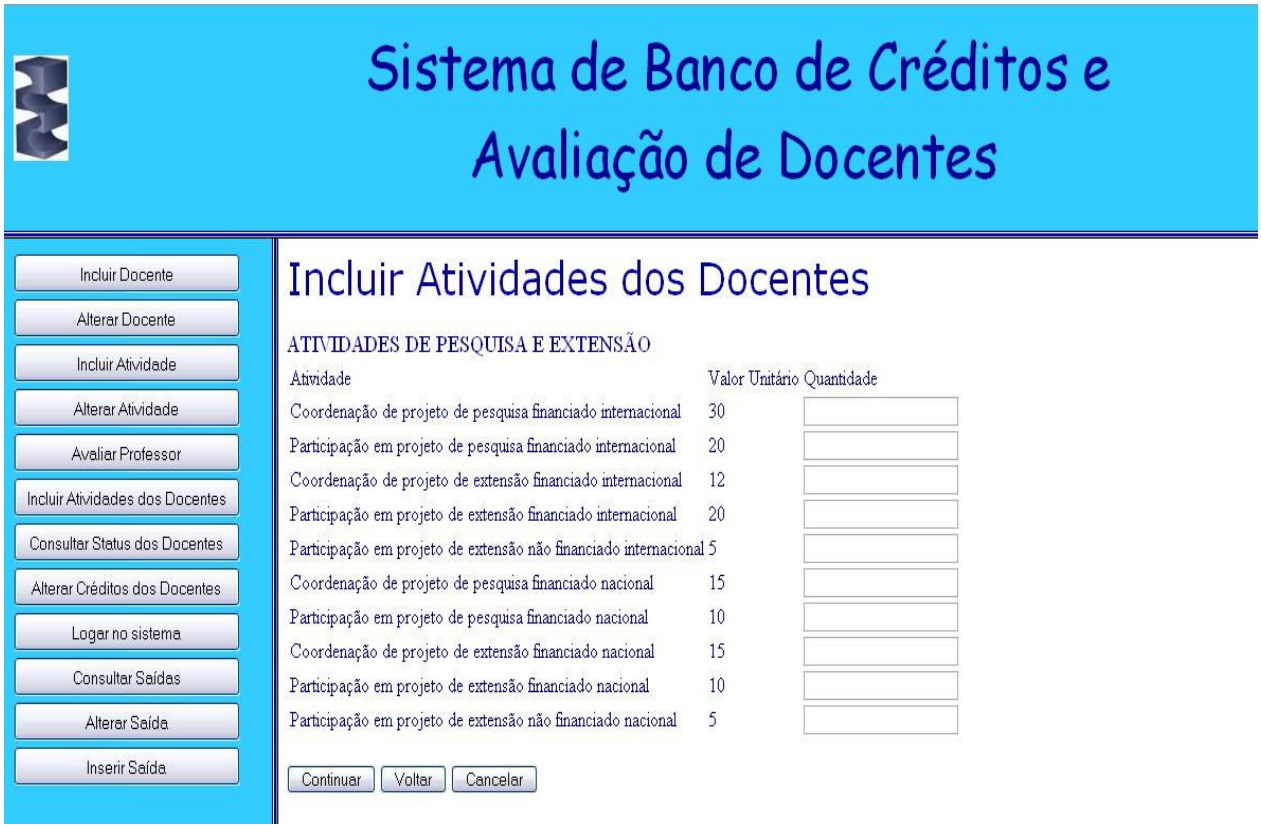


Figura 13. Uma das telas de inserção de atividades dos docentes.

O avaliador poderá cadastrar e alterar registros de professores, além de gerar avaliações para todos os professores. O coordenador pode fazer tudo que os professores e avaliadores fazem, além de cadastrar e alterar atividades a serem realizadas pelos docentes. O coordenador pode consultar e alterar dados e gerar avaliações não só suas como dos demais docentes também. A única coisa que não é permitida ao coordenador é de cadastrar as atividades realizadas pelos outros docentes.

A Figura 14 demonstra o fluxo que ocorre em uma avaliação de docente. O Action-Servlet recebe a requisição e encaminha para o AvaliarAction (Action), o Action em seguida pega as propriedades da consulta no AvaliarForm (ActionForm) e depois acessa camada de negócios passando um docente e um semestre como parâmetros. A requisição vai "descendo" pela camada de negócio até chegar ao repositorioAtividadeDocenteBDR que é o responsável por consultar no banco as atividadeDocente desse docente e semestre, passados como parâmetros. O Action recebe a consulta, seta as propriedades do ActionForm e em seguida chama a página de exibição da resposta (sucessoAvaliacao.jsp). A página de resposta em seguida acessa o ActionForm para pegar as propriedades setadas pelo Action e exibe sua resposta. A Figura 15 ilustra um resultado obtido.

A outra principal funcionalidade do sistema é a implementação da fila de saída dos docentes para pós-doutorados. A tela que lista as saídas cadastradas no banco de dados (Figura 20., no Anexo 1.) é visível para todos os usuários, inclusive docentes. No sistema de banco de créditos, os docentes só têm permissão para consultar a fila de saída, consultar o status e créditos de alguma saída sua e alterar os créditos dessa saída. Os avaliadores só têm permissão para fazer consultas nas saídas, e o coordenador tem o poder de inserir as saídas na fila e atualizar o status das saídas, além das permissões atribuídas a docentes e avaliadores também. Para visualizar mais algumas telas importantes do sistema, podemos verificar o Anexo 1. Este Anexo ilustra telas de avaliação e a tela de consultas a saídas.

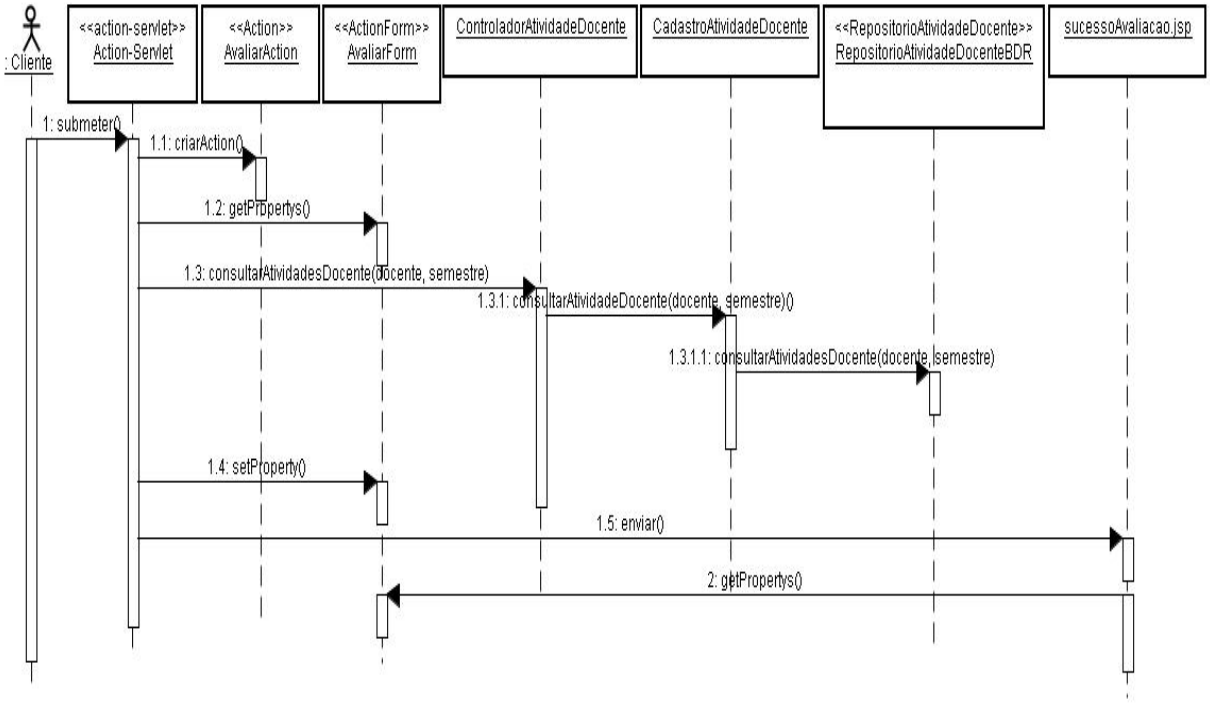


Figura 14. Diagrama de Seqüência do caso de uso de Gerar Avaliação dos docentes.

Figura 15. Exemplo do resultado de uma avaliação.

4.6 Softwares Utilizados

Para o desenvolvimento deste sistema foi utilizado um conjunto de softwares que tornaram a implementação mais fácil e menos tediosa. Para fazermos programação Java, utilizamos a IDE (*Integrated Development Enviroment* – Ambiente Integrado de Desenvolvimento) Red Hat Developer Studio (RHDS) [12] . Essa IDE, que ainda está na versão *Beta*, surgiu assim que a Red Hat incorporou o Exadel Studio ao seus produtos. O Exadel Studio era um *pluging* para o Eclipse [13] que facilitava o desenvolvimento de aplicações web.

O JBoss é um servidor de aplicação de código aberto [14] , que já vem integrado ao RHDS juntamente com o Eclipse. O RHDS, além dessas vantagens, vem preparado para o desenvolvimento de projetos Struts ou JSF. Ao criar um desses tipos de projeto, a IDE já configura todo seu ambiente de desenvolvimento, bastando a você se preocupar apenas com suas classes.

Para gerenciar nosso banco de dados, usamos o SGBD MySQL [15] . Como já foi mencionado, o MySQL é um sistema gerenciador de banco de dados que faz uso da linguagem SQL para manipular seus dados. É um banco de dados que é código aberto e é atualmente um dos mais populares. Para interagir com o SGBD, usamos um software chamado HeidiSQL [16] . O HeidiSQL é um software que gasta poucos recursos da máquina, tem um visual bom de trabalhar (como podemos observar na Figura 16) e que ajuda bastante os desenvolvedores a usar o MySQL. Ele permite que você gerencie e navegue por seus bancos de dados e tabelas através de uma interface semelhante ao Windows.

Para criarmos arquivos de modelagem UML, usamos a versão gratuita do software Jude [17] que assim como o HeidiSQL, também possui uma interface visual simples.

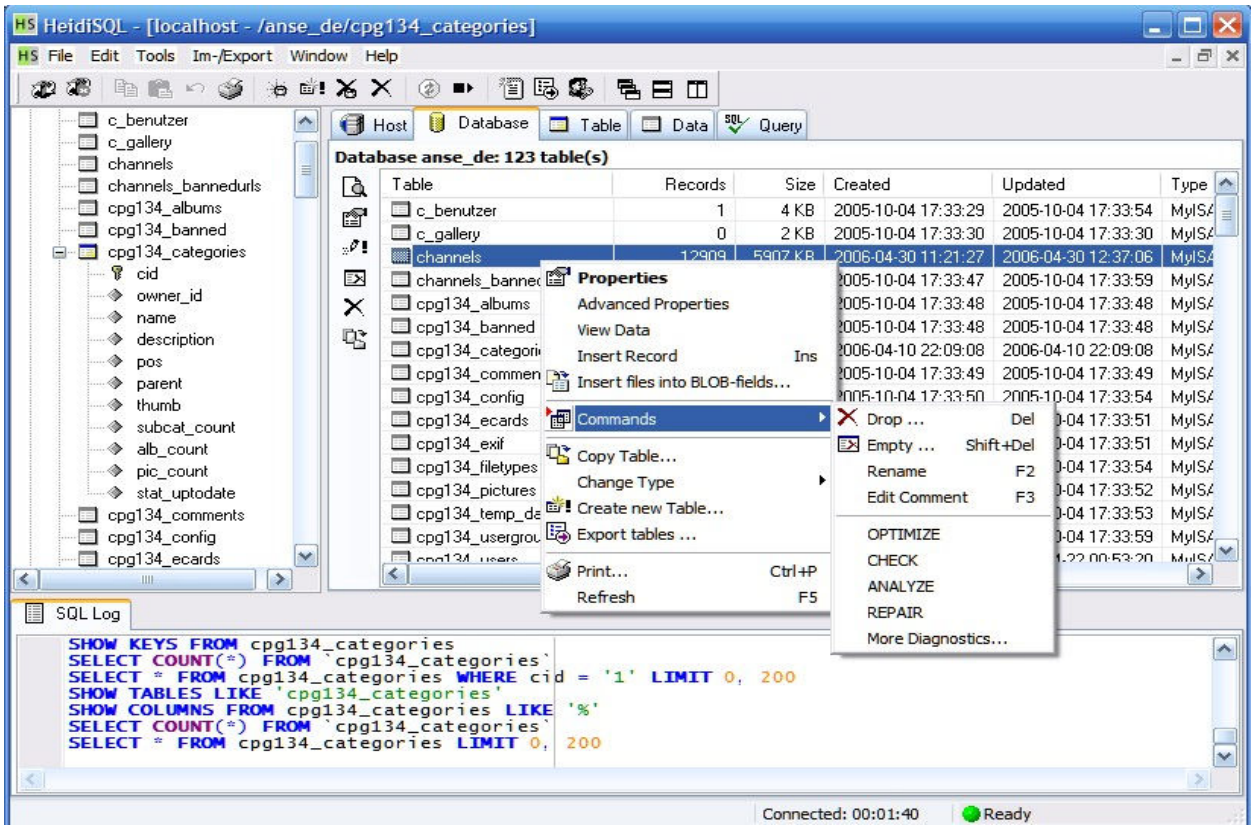


Figura 16. *Screenshot* do HeidiSQL [16] .

4.7 Dificuldades

Durante o desenvolvimento dos sistemas, surgiram algumas dificuldades. A primeira dificuldade foi para se fazer o levantamento dos requisitos iniciais. Para fazer o levantamento dos requisitos era necessária uma reunião com alguns professores e o coordenador do curso, e não foi fácil conseguir um horário em que todos poderiam estar disponíveis. Após essa reunião e já com os primeiros requisitos identificados, as principais dificuldades passaram a ser na tentativa da configuração do ambiente para o desenvolvimento e também na corrida para tentar fazer com que os nossos sistemas interagissem com alguns sistemas externos.

Como foi a primeira vez que fomos responsáveis pela configuração de um projeto, encontramos muitas dificuldades. Tentamos alguns SGBDs antes de utilizar o MySQL [15] , mas não obtivemos sucesso algum em conseguir a criação de um banco de dados. Antes mesmo de conseguir com o MySQL tentamos outras versões do mesmo e não fomos felizes, pois a interação com o SGBD não era nada fácil. Até que após muitas procuras na internet, conseguimos achar o HeidiSQL [16] .

Depois veio a tentativa de configurar uma aplicação web. Tentamos primeiro o desenvolvimento com o Exadel Studio, que é uma versão anterior do RHDS, porém não fomos felizes. Não conseguimos integrar nenhum servidor web com o nosso projeto, gastamos bastante tempo nessa fase. Só com a descoberta do RHDS é que finalmente conseguimos "fazer rodar" uma aplicação web.

Enquanto isso, tentávamos entrar em contato com a plataforma lattes e com a WebQualis para que disponibilizassem dados para o nosso sistema, sem obter êxito. Apenas no projeto SIGA conseguimos que eles disponibilizassem dados para nosso sistema, mas devido a problemas burocráticos, que não temos como interferir, os dados ainda não são disponibilizados e os sistemas ainda não se comunicam.

Durante a fase de implementação, encontramos algumas dificuldades em utilizar o *framework* Struts, que é muito poderoso, porém não tínhamos muito conhecimento de suas técnicas. Também encontramos dificuldades por ser um projeto novo, onde estávamos sozinhos na implementação, foi a primeira vez que isso ocorreu. Estamos acostumados a usar rotinas antigas e apenas ajustá-las a nossas necessidades. Mas neste caso, tudo estava no início, não tinha como obter exemplos ou rotinas parecidas.

Outra dificuldade que tivemos foram os *designs* das páginas. Não temos muito conhecimento em CSS e seu funcionamento. Também foi a primeira vez que tivemos que nos preocupar com as interfaces com o usuário, pois em todos os projetos anteriores, tínhamos uma equipe responsável por isto.

4.8 Implantação

Para implantar o sistema em alguma máquina servidora você só precisa ter um contêiner web "rodando" na sua máquina, uma máquina virtual Java, um banco de dados MySQL e o arquivo ".war" da nossa aplicação.

Primeiramente, você deve ir em http://java.sun.com/j2se/1.4.2/download.html e fazer o download da JRE (Java Runtime Enviroment – Ambiente de execução Java) que é onde se

encontra a máquina virtual Java. Em seguida você vai em http://tomcat.apache.org/ e faz o download da versão 5.5 do Tomcat. O Tomcat é o contêiner onde sua aplicação ficará hospedada. Depois vá em http://dev.mysql.com/downloads/ e faça o download do MySQL.

Instale primeiro o JRE na sua máquina, depois instale o Tomcat, e em seguida o MySQL. Estando com os aplicativos instalados e tendo em mãos o arquivo compactado da nossa aplicação (avaliação.war), coloque esse arquivo dentro da classe *webapps* do Tomcat. Essa pasta fica no diretório padrão de instalação do Tomcat. Por exemplo, C:\Program Files\ Tomcat 5.5\webapps. Depois selecione o nosso banco de dados e coloque na pasta "data" do diretório de instalação do MySQL (por exemplo C:\mysql\data).

O próximo passo é iniciar o Tomcat. É só executar o programa tomcat.exe na pasta "bin" do diretório de instalação do Tomcat. Para acessá-la, vá ao endereço http://IP_DA_MAQUINA:8080/avaliacao. Onde IP_DA_MAQUINA é o endereço IP da máquina onde você hospedou o sistema. Para acessar da mesma máquina basta digitar http://localhost:8080/avaliacao.

Capítulo 5

Conclusões e Trabalhos Futuros

O intuito deste trabalho foi o desenvolvimento de dois sistemas de informação integrados para o DSC. O primeiro teria a função de gerenciar atividades exercidas pelos docentes e de avaliá-los. O outro seria responsável por implementar uma fila de espera para docentes que desejam se afastar do DSC por um tempo para fazer pós-doutorado.

5.1 Contribuições

O desenvolvimento destes sistemas, como trabalho de conclusão de curso, possibilitará ao DSC um maior controle e organização sobre suas atividades. As avaliações de docentes se tornarão mais informativas, proporcionando uma maior facilidade para tomadas de decisões internas, como as divisões de atividades entre seus docentes.

Este trabalho possibilitará também que o DSC passe a permitir que seus docentes se afastem do departamento, para fazer pós-doutorado, durante um período de forma bastante organizada.

5.2 Considerações

Os sistemas desenvolvidos terão no escopo apenas a satisfação de duas necessidades atuais do DSC, que são o controle de disciplinas para os professores poderem se afastar para um pós-doutorado e otimizar a avaliação de professores. Nenhum outro controle com relação à ementa de disciplinas, quantidade de alunos por sala, horários de laboratório, ou qualquer outra particularidade da parte acadêmica não entra no escopo deste trabalho.

Outra limitação é que os sistemas não têm comunicação com alguns sistemas externos como o SIGA, a plataforma lattes ou com a webqualis da CAPES. Isto tornaria os sistemas bem mais interessantes, pois diminuiria a quantidade de informação a ser passada pelos professores e faríamos melhor uso de uma das maiores qualidades dos sistemas de informação, que é evitar ao máximo a entrada de dados iguais em locais diferentes.

5.3 Trabalhos Futuros

Nesta seção vamos citar algumas formas de dar continuidade a este trabalho. A principal delas seria integrar nosso sistema com a base de dados do SIGA, isto diminuiria a quantidade de dados a serem passados diretamente pelos docentes ao sistema e automatizaria o processo. Para informatizar mais ainda, seria interessante também integrarmos nossos dados com as bases da plataforma lattes e da CAPES.

Outro trabalho futuro que pode ser realizado é o desenvolvimento de uma interface visual mais agradável para os usuários, através de páginas JSP e HTML Outra forma de progredirmos com esse trabalho seria fazendo mais uso dos recursos que o *framework* Struts nos traz. Isto poderia diminuir ainda mais a quantidade de código Java em nossas páginas JSP.

Bibliografia

[1] STAIR, Ralph M. Princípios de Sistemas de Informação. 2ª ed. Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora S.A., 1998, 451p.

[2] LARMAN, Craig. *Applying* UML *and patterns: an introduction to object-oriented analysis and design and interative development.* 3 ed. Massachussetts: Prentice Hall, 2004. 702p.

[3] Conceito – *Iteration*. Disponível na URL: http://www.epfwiki.net/wikis/openuppt/openup_basic/guidances/concepts/iteration,_lam4ADkBEduxovfWMDsntw.html , Acesso em: 23/11/07

[4] JACOBSON, I., BOOCH, G. e RUMBAUGH, J. *The Unified software development process*. Addison – Wesley, 1999.

[5] KRUCHTEN, P. *The Rational Unified Process – An Introduction,* 2ª ed. Addison – Wesley, 2000.

[6] TENENBAUN, Andrew S. Redes de Computadores. 3ª ed. Rio de Janeiro: Campus, 1997, 923p

[7] DevMedia – JavaMagazine – Introdução a Taglib, disponível na URL: http://www.devmedia.com.br/articles/viewcomp.asp?comp=3317&hl=, Acesso em: 13/11/2007

[8] HUSTED, Ted; DUMOULIN Cedric; FRANCISCUS, George; WINTERFELDT David. Struts em ação. Rio de Janeiro: Editora Ciência Moderna, 2004. 604p.

[9] DEITEL, H. M. e DEITEL P.J. Java Como Programar. 3ª ed. Porto Alegre: BookMan, 2001, 1201p.

[10] COCKBURN, A. *Wtiting effective Use Cases.* Reading, MA.: Addison-Wesley, 2001.

[11] TAVEIRA, Julio César. *Help Desk Framework*. Recife: DSC-UPE, 2007. Monografia de Trabalho de Conclusão de Curso.

[12] *Red Hat Developer Studio*, disponível na URL: http://www.redhat.com/developers/rhds/ , Acesso: 20/11/2007.

[13] *Eclipse, an open development platform*, disponível na URL: http://www.eclipse.org/ . Acesso em: 20/11/2007

[14] JBoss.org *community Driven*, disponível na URL: http://labs.jboss.com/ , Acesso: 20/11/2007.

[15] MySQL *The World's most popular open source database*, disponível na URL: http://www.mysql.com/, Acesso: 20/11/2007.

[16] HeidiSQL – MySQL *made easy*, disponível na URL: http://www.heidisql.com/ , Acesso: 20/11/2007.

[17] JUDE – *System Design Tool*, disponível na URL: http://jude.change-vision.com/jude-web/index.html/ , Acesso: 20/11/2007.

Anexo 1

Telas dos Sistemas

O propósito deste documento é demonstrar algumas telas dos Sistemas de Avaliação de Docentes e Banco de Créditos. Vamos mostrar algumas telas de resultados de Avaliação e um consulta à Fila de Saída de Docentes.

Quando vamos gerar alguma avaliação, devemos antes passar alguns parâmetros que serão utilizados na pesquisa. A Figura 17. ilustra a tela onde os parâmetros são setados.

Figura 17. Tela Inicial de uma avaliação.

Como podemos observar, esta tela permite que seja selecionado apenas um ou todos os professores, quais os semestres a serem avaliados e se o desejo é fazer uma avaliação completa (com todos os tipos de atividades) ou avaliação por um determinado tipo de atividade.

Caso seja selecionado um professor e todos os tipos de atividade, o resultado obtido será semelhante à Figura 18. A Figura 18. foi obtida ao selecionar o professor Tiago Massoni nos semestres 2007.1 e 2007.2 com todas as atividades. É exibida a pontuação parcial por cada tipo de atividade, e após é exibido o total do semestre. Que corresponde à soma das pontuações parciais.

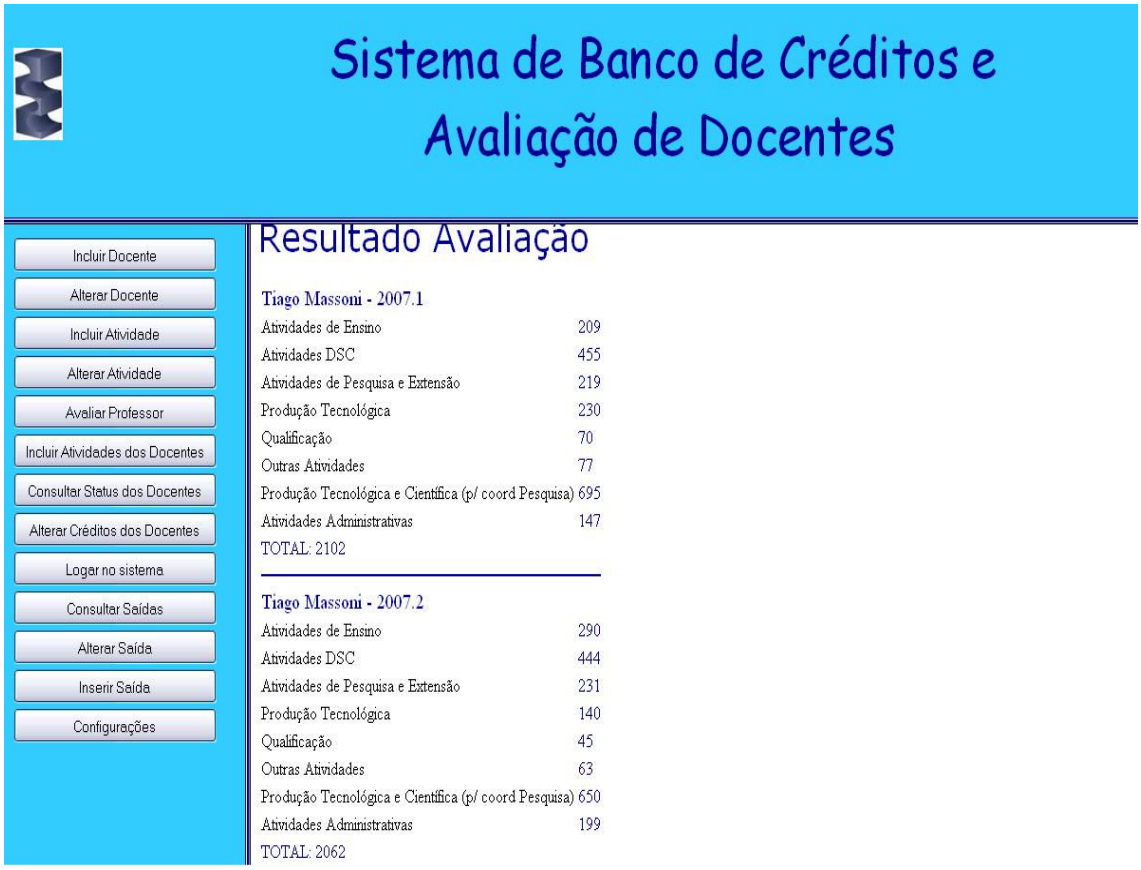


Figura 18. Resultado de Avaliação (1).

Caso nós selecionarmos todos os professores em um determinado tipo de atividade, nós obteremos um resultado semelhante à Figura 19. Este exemplo foi obtido ao selecionarmos o tipo de atividade "Qualificação".

Podemos observar que na tela primeiro ocorre uma separação por professor e depois por semestre. Na tela podemos ver que primeiro foram exibidas as pontuações obtidas pelo professor Tiago Massoni no tipo de atividade "Qualificação". A tela nos mostra a pontuação de cada atividade deste tipo, a quantidade de vezes que essa atividade foi realizada e a pontuação total desta atividade. Em seguida é exibida a pontuação total obtida pelo professor nesse tipo de atividade no semestre 2007.1. Em seguida, é feito o mesmo com semestre 2007.2.

Após a exibição das pontuações do professor Tiago, estão exibidas as pontuações do professor Abel Guilhermino. Como não estão sendo exibidos os dados sobre as atividades dele no semestre 2007.1., é um sinal que ele não cadastrou suas atividades realizadas nesse semestre. Então as únicas informações referentes ao professor Abel são do semestre 2007.2.

Figura 19. Resultado de Avaliação (2).

Já a figura 20. ilustra um consulta a fila de espera. Como podemos observar, a tela traz as saídas ordenadas por semestre. A consulta traz além do semestre da saída, o nome do professor a se ausentar, a situação da saída (em espera, em andamento, realizada ou cancelada), e a quantidade de créditos pagos e a pagar pelo professor. Podemos ver que na fila de saída estão os professores Abel Guilhermino (está ausente no semestre 2007.2), Tiago Massoni (se ausentará no semestre 2008.1) e Carmelo Bastos (se ausentará em 2009.1).

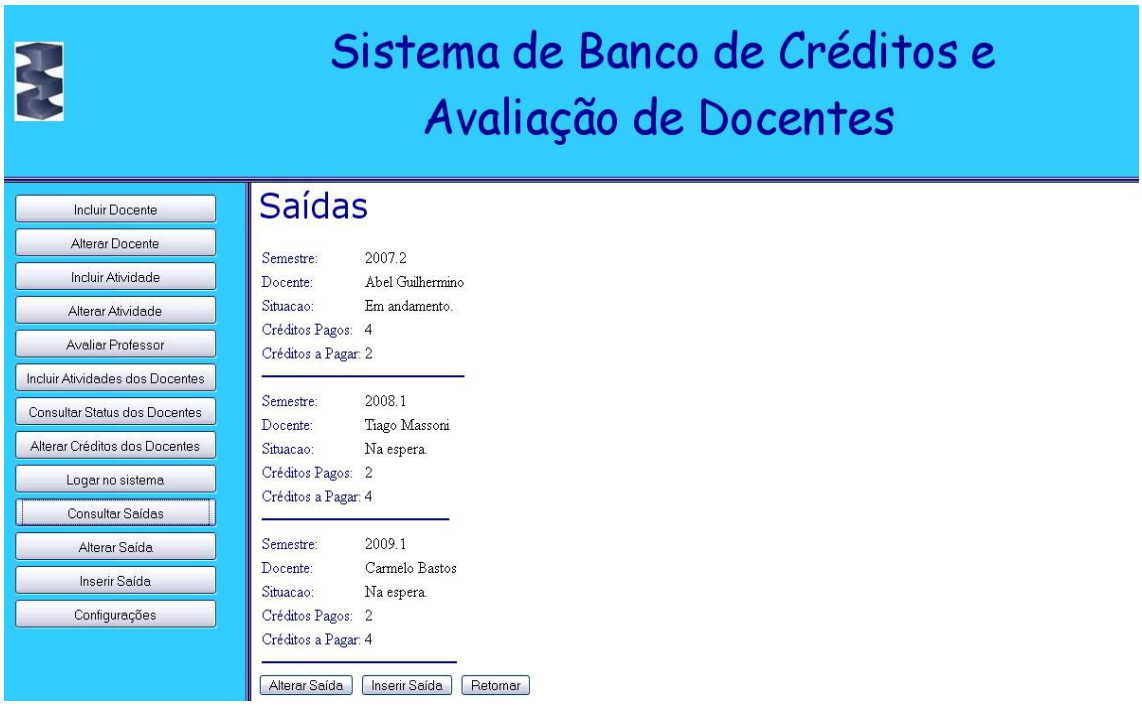


Figura 20. Tela de Consulta à fila de Espera.