ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

DESENVOLVIMENTO DE SISTEMA DE INFORMAÇÃO WEB PARA O CONTROLE INTERNO DE PROTOCOLOS DA ESCOLA POLITÉCNICA DE PERNAMBUCO

Trabalho de Conclusão de Curso

Engenharia da Computação

Nome do Aluno: Paulo André Ferreira
Orientador: Prof. Dr. Adriano Lorena Inácio de Oliveira

Recife, julho de 2006

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de Sistemas Computacionais

DESENVOLVIMENTO DE SISTEMA DE INFORMAÇÃO WEB PARA O CONTROLE INTERNO DE PROTOCOLOS DA ESCOLA POLITÉCNICA DE PERNAMBUCO

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.

Nome do Aluno: Paulo André Ferreira
Orientador: Prof. Dr. Adriano Lorena Inácio de Oliveira

Recife, julho de 2006

UNIVERSIDADE
DE PERNAMBUCO

Paulo André Ferreira

DESENVOLVIMENTO DE SISTEMA DE INFORMAÇÃO WEB PARA O CONTROLE INTERNO DE PROTOCOLOS DA ESCOLA POLITÉCNICA DE PERNAMBUCO

Resumo

Os sistemas de informação (SI) podem ser empregados com diversos propósitos por uma instituição, dependendo de suas necessidades específicas. Sua elaboração depende das opções de telecomunicações e redes adotadas.

Este trabalho consistiu do desenvolvimento de um sistema de informação voltado para o controle de protocolos da área administrativa da escola Politécnica de Pernambuco – Poli, utilizando os recursos da rede local para gerenciamento e controle de envio e recebimento de documentos entre seus departamentos, informando aos usuários a localização atual do documento em trânsito, o destino final do mesmo, bem como seu possível estado: em andamento, suspenso, deferido, indeferido ou arquivado.

A elaboração do sistema seguiu as boas práticas da engenharia de software, tendo sido adotado o “modelo espiral” como forma de organização do processo.

O trabalho teve início pela análise de requisitos, etapa na qual foram coletadas as informações inerentes ao sistema, partindo-se para a prototipagem em busca de validação. Em seguida passou-se à construção propriamente dita, seguindo-se os testes de caso de uso e a implantação.

Abstract

The information systems (IS) can be used with many different purposes by an institution, according to its particular purposes. The elaborating of an information system depends on the alternatives of telecommunications and adopted nets.

This work consists in the development of an information system directed toward the control of protocols of the administrative area of the Escola Politécnica de Pernambuco, Poli. It uses the resources of the local net for management and control of sending and receiving documents among its departments, informing the user the current l location of the document in transit, the final destination and as much as possible its condition: in progress, suspended, accepted, rejected or filed.

The system elaboration followed the software engineer good practices and adopted the model spiral for the process of organization.

The work began with the analysis of requirements, stage in which all the inherent information of the system was collected, following to the development a prototype for validation. Then began the built and finally tests of use case and the implantation.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

API – Application Programming Interface.
GUI – Graphical User Interface (Interface Gráfica com o Usuário).
HQL – Hibernate Query Language.
HTML – Hypertext Markup Language.
HTTP – Hypertext Transfer Protocol.
JAR – Java archive.
J2EE – Java 2 Enterprise Edition.
J2SE – Java ™ 2 Platform, Standard Edition.
JDBC – Java Database Connectivity.
JSP – Java Server Page.
JSTL – JSP Standard Tag Library
MVC – Model View Controller.
ORM– Mapeamento Objeto/Relacional.
S@DPOLI – Sistema de Acompanhamento de Documentos da Poli.
SGDB – Sistema Gerenciador de Banco de Dados.
SI – Sistema de Informação.
SQL – Structured Query Language.
UC – User Case (Casos de Uso).
UML – Unified Modelling Language.
URL – Uniform Resource Location.
VO – Value Object.
XML – Extensible Markup Language.

Agradecimentos

Agradeço primeiramente a Deus por todas as oportunidades vivenciadas e por sempre ter me guiado a bons caminhos.

À minha família, em especial à minha mãe Vitória Ferreira e irmã Rosilene Ferreira que sempre me deram o apoio necessário para alcançar meus objetivos e por todo o investimento depositado na minha formação como ser humano e na vida estudantil.

Agradeço ao meu orientador, Prof. Dr. Adriano Lorena Inácio de Oliveira, pelo apoio, incentivo e compreensão durante esta jornada.

Aos meus amigos universitários e ex-universitários que de alguma forma contribuíram para o meu bom desempenho acadêmico, especialmente à turma de Engenharia de Computação 2001.1 a qual transformou-se numa equipe homogênea de estudos, apoios e gargalhadas.

Capítulo 1

Introdução

Com o advento da tecnologia proveniente do século XX, a sociedade vem vivenciado tempos de praticidade. A busca pela economia de tempo, racionalização das atividades e maior eficiência na execução de tarefas é cada vez mais freqüente por parte de indivíduos, empresas e instituições. Nesse sentido, o computador tem se apresentado como um instrumento facilitador, capaz de oferecer recursos para os mais diferentes tipos de necessidades. Dentre os vários serviços oferecidos pelo computador, destacam-se os Sistemas de Informação (SIs), os quais permitem o gerenciamento e controle de vários processos [2]. Um bom sistema de informação ajuda a empresa a atingir as suas metas.

A Escola Politécnica de Pernambuco - Poli é uma instituição de ensino na qual o controle dos documentos que transitam entre seus departamentos é realizado de forma manual, pela utilização de "cadernos de protocolos". Cada departamento possui seu próprio caderno, o qual é encaminhado para diferentes setores da área administrativa na tentativa de obter seu próprio controle, contudo o gerenciamento dos documentos é deficiente, pois o controle descentralizado e subdividido gera diversos problemas, como por exemplo registro do mesmo documento em diferentes cadernos, levando à dificuldade de rastreamento do mesmo. A Poli possui uma boa estrutura de rede Intranet que viabiliza a comunicação entre seus departamentos, bem como um servidor Web e um banco de dados, justificando assim a criação de um sistema de informação computadorizado que torne mais prático e eficiente o seu controle de documentos.

Um sistema é um conjunto de elementos ou componentes que interagem para atingir objetivos. São as relações entre esses elementos ou componentes que determinam o seu fluxo de operação [2][3]. Os sistemas apresentam entradas, mecanismos de processamento e saídas com possibilidades de "*feedback*". Como todo produto, um sistema pode ser simples, tal como a lavagem de carros ou mais complexos como a construção dos próprios carros. As metas de um sistema em um contexto corporativo normalmente estão relacionados a lucros e satisfação do cliente.

Sistemas de informação funcionam pela coleta de dados, processamento dos dados coletados e produção de informação. Eles também apresentam entradas, processamento da informação e saídas. Sua elaboração depende de software como ferramenta, hardware, pessoal para interagir com o software e hardware, banco de dados para armazenar dados e informação como também procedimentos que definem a política de manuseio. Um sistema deve ser visualizado num contexto amplo, o "mini-mundo" ao qual representa ou opera, para depois descer em níveis de detalhes [2][3].

O SIs vão além de um mero produto de software, pois apresentam um contexto lucrativo, no processamento de dados e geração de informação, a lucratividade pode ser de custos, de eficiência, gerência, dentre outros. Os *feedbacks* provenientes dos resultados normalmente estão atrelados a mais de uma pessoa, quer envolvida diretamente ou não, e podem estar relacionados com a qualidade e produtividade de uma organização.

São muitos os benefícios oferecidos pelos SIs, destacando-se: maior segurança, melhor serviço prestado aos usuários, aperfeiçoamento das comunicações, menor número de erros, maior precisão, administração mais eficiente e carga de trabalho reduzida [2][3][4]. Tais sistemas podem ser desenvolvidos para atender a vários propósitos, mas a sua elaboração e construção estratégica depende das operações de telecomunicações e redes adotadas. Alguns sistemas atendem a necessidade de apenas um usuário final, outros atendem vários usuários finais que podem estar no mesmo espaço geográfico ou distribuídos pelo mundo [2][4]. A complexidade no desenvolvimento dos SIs depende naturalmente do tipo de problema a ser solucionado como também dos recursos oferecidos e cabíveis para sua construção.

O uso do computador torna-se ainda mais interessante e valioso quando se pensa na Web [5], responsável pela tecnologia Internet, que se originou com o intuito de disponibilizar informações. Sua estrutura, inicialmente definida, baseava-se em armazenagem e visualização de documentos eletrônicos com fins de estudo e pesquisa.

O número de usuários da Web cresce a cada ano gradativamente, fator fundamental para atrair também o interesse das empresas e organizações. A visão de lucros e novas oportunidades no mercado fizeram surgir outras abordagens no uso da Web. Os interessados no canal de comunicação permitiram que a tecnologia Web fosse incorporando novos recursos e outras roupagens. Uma mudança bastante significativa foi o fato dos usuários poderem interagir com o ambiente Web, submetendo formulários aos servidores e recebendo informações dinâmicas e processadas.

Desde a sua origem os repositórios de informação ficavam espalhados em servidores Web [6]. A estrutura cliente/servidor [6][7] continuou a evoluir fortemente no lado do servidor. Os processamentos de informações e armazenagem exigem servidores com altos desempenhos. Os terminais dos clientes também precisavam evoluir, mas em proporções bem menores ao ponto de não encarecer o produto (computador). Esta estrutura permitiu popularizar e disponibilizar cada vez mais computadores na sociedade.

O desenvolvimento de sistemas de informação direcionado para Web é uma conseqüência das mudanças e da iteratividade vigente. Entre os vários produtos disponíveis na rede Web os SIs têm a vantagem de oferecer aos seus usuários a possibilidade de realizar negócios e trabalhos. A metodologia de construção desses sistemas apresenta uma nova representação focada para o contexto Web. As interfaces, a estrutura do conteúdo, a navegabilidade e o estilo de apresentação podem ser adaptáveis ao perfil do usuário [9][16].

O ganho de produção na construção de sistemas com redução de custos e prazos está relacionado com os recursos de software reutilizáveis. Projetos web normalmente são extensos com muitas linhas de código e lógica de programação peculiar ao produto web. A adoção de componentes populares e de fácil manuseio oferecem tanto um arcabouço de software inicial para um projeto em construção quanto a segurança de que aquela estrutura fora validada e testada em projetos existentes no mercado [9].

1.1 Objetivos

O objetivo prático é apresentar o desenvolvimento de um sistema de informação de apoio voltado para o controle de protocolos da área administrativa da Escola Politécnica de Pernambuco – Poli, utilizando os recursos da rede local para gerenciamento e controle de envio e recebimento de documentos entre seus departamentos, deixando claro aos usuários a localização atual do documento que se encontra em trânsito, o destino final dos mesmos, bem como seu possível estado: em andamento, suspenso, deferido, indeferido ou arquivado; melhorando as condições de rastreamento e eliminando os problemas causados pela atual forma de controle. Como objetivo teórico teve-se o estudo e emprego das regras e recomendações da Engenharia de Software na elaboração do projeto e, deste modo, a vivência dos desafios de planejar, especificar, implementar e testar o sistema em acordo com a expectativa do cliente.

1.2 Estrutura do Trabalho

Esta monografia segue a seguinte estrutura: o Capítulo 2 descreve a abordagem da análise de requisitos, as dificuldades e estratégias adotadas na coleta de informação para o sistema, as regras de negócio, a descrição do processo de desenvolvimento, como também uma ilustração geral dos casos de uso implementados.

O Capítulo 3 trata recursos tecnológicos adotados na construção do sistema: plataforma J2EE, frameworks Struts e Hibernate, Servlets e JSP.

O Capítulo 4 aborda o desenvolvimento sistema: arquitetura, testes e representação gráfica do sistema S@ADPOLI.

O Capítulo 5 apresenta as lições aprendidas na construção do sistema, os riscos e dificuldades encontrados ao longo da elaboração do projeto e os caminhos alternativos para prosseguir com o desafio.

Finalmente no Capítulo 6 é apresentada a conclusão deste trabalho, e os possíveis trabalhos futuros em prol da extensão do sistema.

Capítulo 2

Técnicas, Métodos e Processo

Neste capítulo são descritos conceitos da análise de requisitos, a problemática da coleta de informação para o sistema de controle de protocolos, os requisitos funcionais e não funcionais do sistema, uma visão do sistema por meio de um diagrama UML [11] de casos de uso.

2.1 Visão Geral da Análise de Requisitos

A análise de requisitos é uma das fases mais complexas na elaboração de qualquer produto de software. Cabe a esta fase compreender o problema para o qual se busca uma solução computacional [9][10]. Dentre as metas da Engenharia de Requisitos (ER) pode-se citar: descrever o quê o sistema deve fazer, especificado junto com o cliente; delimitar o escopo da problemática; gerenciar mudanças de requisitos; definir a interface com o usuário ou protocolo de comunicação.

Levando em consideração que o desenvolvimento de software tem como finalidade prover valor ao cliente e aos usuários finais, tem-se a necessidade em que os requisitos devem ser definidos na direção correta [9]. O requisito representa uma funcionalidade do sistema, isto é, uma atividade computável para o futuro produto e de interesse do cliente e usuário. Requisitos ambíguos ou não levantados no momento certo podem prover má interpretação ou até mesmo conflitos entre os "***stakeholders***", que neste contexto são os indivíduos interessados pelo produto. Problemas desse gênero, se não forem tratados a tempo, no período estabelecido para a fase de elicitação, serão elementos com potencial a futuros retrabalhos.

O sucesso ou fracasso desta fase está diretamente relacionado com a capacidade do analista de sistema(s) em conduzir a especificação do produto de forma incremental e fiel ao idealizado pelo cliente. A grande questão é saber como conseguir gerenciar o que está sendo coletado, filtrar o que é necessário para cada momento do projeto, definir o escopo do problema e fazer com que as reuniões continuem a render informações úteis.

A ER oferece boas abordagens a esta fase de elaboração de software [9][10], no entanto, não há regras definidas, cada caso é um caso, por mais experiente que seja um analista de sistema, o mesmo poderá vir a ter facilidades ou dificuldades de acordo com a complexidade do problema, do nível de envolvimento do cliente dentre outros fatores.

O desenvolvimento de software é um processo bastante instigante e minucioso. Infelizmente não é freqüente que o comprador do produto de software já tenha incorporado uma

consciência de que nada será construído de forma estimada caso não haja sua participação direta ou ao menos um canal de comunicação com seus representantes durante todo o desenvolvimento.

2.2 Artefatos e Atividades para Analise de Requisitos

Como dito na Seção 2.1 não há regras estabelecidas para o fluxo de requisitos, apenas boas recomendações para esta fase, portanto, os requisitos podem estar escritos de várias maneiras de acordo com: necessidade dos usuários, características do sistema, casos de uso ou mesmo solicitação de mudança e com requisitos não funcionais, que normalmente representam uma exigência imposta pelo cliente e estão relacionados com atributos ou qualidades do produto [8][9][10].

As atividades tradicionais para esta etapa consistem de prototipagem, coleta de informações, detalhamento e especificação de casos de uso, estruturação do modelo, revisão e homologação de requisitos. A figura 1 ilustra o fluxo citado.

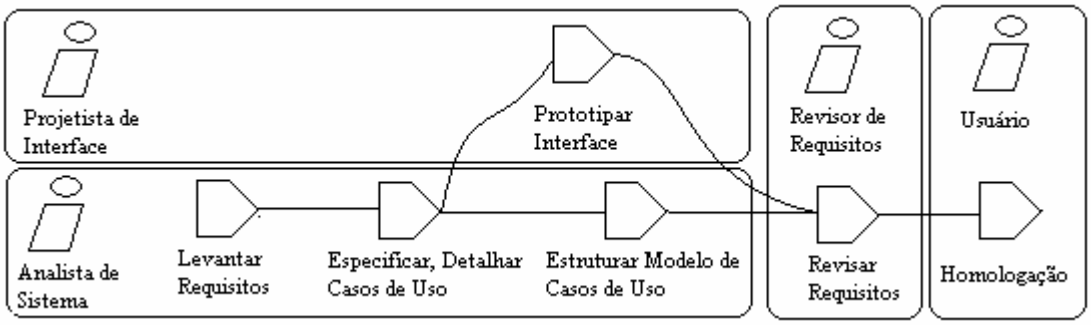


Figura 1. Fluxo de atividades básicas da análise de requisitos. Figura pertencente à escola Qualiti Software Processes.

No levantamento de requisitos, o papel do analista de sistemas é desenvolver documentos que descrevam o modelo a ser representado. Os artefatos de entrada consistem do resultado de, por exemplo, reuniões com todos *stakeholders* envolvidos na elaboração e uso do sistema. Os artefatos de saída normalmente são: uma descrição da necessidade e da viabilidade, descrição do ambiente técnico do sistema, especificação e diagramas de casos de uso, dentre outros itens cabíveis [8][9]. A especificação de casos de uso refere-se à documentação dos requisitos de forma a definir os fluxos de atividades a serem executados num sistema enquanto a estruturação do modelo de casos de uso diz respeito à definição de esboços e diagramas que modelam os requisitos na perspectiva do “mini-mundo” do sistema a ser construído. Na prototipagem de interface o analista define os dados e informação que a interface deve gerenciar e o projetista de interface define a estrutura visual de acordo com tais especificações. A revisão de requisitos consiste de uma releitura dos requisitos com a utilização da prototipagem da interface com o intuito de confirmação e finalmente a homologação de requisitos corresponde a aceitação do cliente/usuário do cenário até então definido.

2.3 Requisitos do Sistema de Controle de Protocolos

A proposta da elaboração do sistema de controle de protocolos surgiu da necessidade da Escola Politécnica de Pernambuco - Poli em melhorar e agilizar o seu fluxo de protocolos entre departamentos. O processo atual, manual, não atende à nova realidade vivenciada. Falhas no gerenciamento, perdas de documentos e muitas vezes ausência de provas que identifiquem os envolvidos na tramitação do documento foram os indícios básicos de alerta na busca de uma nova alternativa para tal atividade, considerada corriqueira na Escola.

O interesse pelo sistema originou-se por parte da Diretoria da Poli, mas para a fase de levantamento de requisitos a mesma não se interessou em fazer parte do processo, embora tenha indicado uma representante para esta finalidade. A partir de então, foram observados os seguintes problemas iniciais:

- Apenas uma profissional, de um único departamento, a contribuir com os requisitos: a chefe do departamento DESAF.
- Datas de reuniões com um espaçamento médio de 15 dias e com freqüentes remarcações.
- Resistência ao envolvimento na coleta e validação de requisitos.
- Requisitos limitados ao contexto vivenciado pelo departamento da funcionária anteriormente citada.

Para viabilizar a obtenção de informações a respeito do fluxo de protocolos foi necessário desenvolver um protótipo gráfico inicial do sistema que possibilitou uma nova aproximação com o departamento DESAF. O protótipo foi implantado numa máquina do DSC - Departamento de Sistemas Computacionais e foi encaminhado a URL de acesso tanto para a Diretoria quanto para o departamento DESAF. A necessidade de implantação do protótipo em um servidor se deu por dois fatores:

1. O protótipo, por ser inicial, tinha grandes chances de mudanças e isto implicava em novas instalações nas máquinas dos interessados.
2. Novos envolvidos poderiam participar da elaboração do sistema, bastando a Diretoria e o DESAF redirecionar a URL para outros funcionários da Poli, sem grandes incômodos.

A tabela 1 descreve os meses envolvidos nesta atividade:

Tabela 1. Período relacionado às atividades da análise de requisitos

Atividade	Mês
Reuniões iniciais com a chefe do departamento DESAF	Setembro/2005
Prototipagem Inicial	Outubro/2005
Documento de dúvidas	Novembro/2005
Disponibilização da 1ª versão do protótipo	Novembro/2005
1ª verificação e validação do protótipo	Dezembro/2005
2ª Rodada de validações com alteração no protótipo	Dezembro/2005
3ª e última rodada de validações com alteração no protótipo	Janeiro/2006

Percebe-se pelo cronograma acima que foram disponibilizados 5 meses para a Poli contribuir com informações relevantes e corretas ao projeto a ser construído. Em seguida partiu-se para a fatoração dos casos de uso descritos na seção 2.6.

Inicialmente a visão de uso do sistema era de que haveria um controle personalizado nas operações feitas pelos usuários, isto é, cada funcionário autenticado teria acesso às mesmas

funcionalidades, porém só visualizaria as solicitações ou documentos pertencentes a ele, quer fosse encaminhado ou recebido. Apenas o responsável pelo departamento poderia visualizar todas as tramitações de documentos referentes ao seu departamento, independentemente do funcionário envolvido. Contudo esta abordagem foi rejeitada pelos seguintes motivos:

1. Nem todo funcionário dispõe de um computador próprio, conseqüentemente não estaria visualizando o sistema em tempos periódicos.
2. Existe a possibilidade de um estagiário realizar o fluxo de operações, substituindo um funcionário do próprio departamento.

Sendo assim, o sistema apresenta um controle menos rígido, no qual todos os funcionários autenticados terão acesso aos protocolos encaminhados ao seu departamento. Contudo, é possível, em qualquer tempo, adaptar o sistema para o que foi originalmente pensado, desde que se faça uma manutenção adaptativa do seu fluxo interno.

2.4 Abordagem das Regras de Negócio

As regras de negócio são elementos lógicos, específicos a cada sistema, por sua vez, a cada visão do cliente. Um mesmo produto produzido em duas instituições distintas podem facilmente apresentar regras em comuns. As regras podem até mesmo ser visualizadas de forma diferenciada pelos próprios "stackholders" envolvidos com a elaboração do projeto [9][10].

Mapear uma atividade do mundo real para um contexto computacional não é uma tarefa simples ou trivial, exige um bom entendimento do negócio e por isso deve ser bem explicado pelo cliente e usuário. As flexibilidades e restrições que um sistema possa ter estão relacionadas com as regras definidas ao software. As regras normalmente são externadas aos poucos pelo cliente e capturadas por completo em fases finais do levantamento de requisitos. As fases iniciais estão voltadas aos fluxos ideais das operações, com poucas restrições ou mesmo sem altos níveis de detalhes [9][10].

O presente processo de controle de protocolos, no contexto prático, é uma atividade simples, pois trata-se de um registro formal e protocolado do fluxo de documentos entre departamentos da Poli. Nesta atividade registra-se o que está sendo encaminhado e o que está sendo recebido. As vantagens referentes a ter um controle de protocolos computadorizado é bem perceptível no momento de localizar quem está manipulando uma solicitação, qual é a situação atual do documento, se já foi arquivado e por qual departamento; tudo isso, sem precisar folhear um caderno de protocolo.

A definição do controle de documentos, para o contexto eletrônico, foi construída a partir de uma abordagem de ciclo de estados para a movimentação que um protocolo, por ventura, possa ter entre departamentos. A característica dinâmica desta atividade tenta ser representada por meio da Figura 2. Na Tabela 2, é descrita a semântica de cada um destes estados.

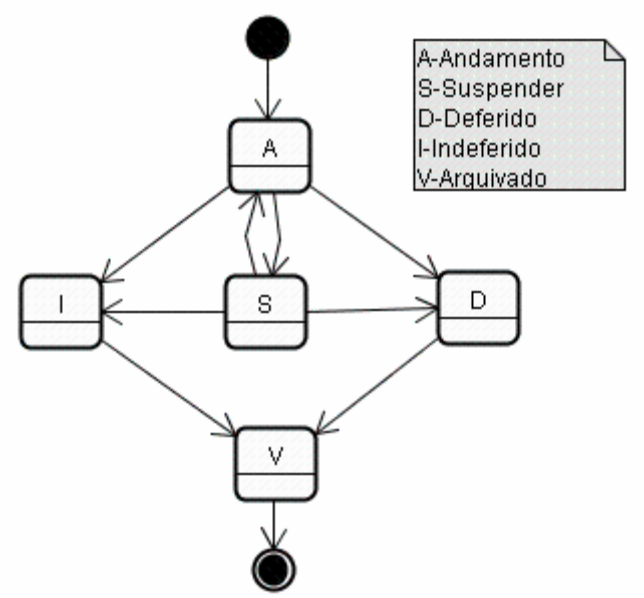


Figura 2. Ciclo de estados de apreciação de documentos.

Tabela 2. Definição dos estados de um documento.

CODIGO	TERMO	DESCRIÇÃO
A	ANDAMENTO	Situação do protocolo encaminhado pela primeira vez através do sistema para um departamento de destino.
S	SUSPENSO	Situação que representa um protocolo que, por motivos diversos, não pode ser atendido dentro de uma margem de tempo estimado.
D	DEFERIDO	Situação que representa a aprovação do documento encaminhado. A mudança de estado para deferido é realizada pelo departamento que está avaliando o protocolo.
I	INDEFERIDO	Situação que representa a não aprovação do documento encaminhado. A mudança de estado para indeferido é realizada pelo departamento que está avaliado o protocolo.
V	ARQUIVADO	Situação que representa o encerramento da avaliação do protocolo.

Política de responsabilidades:

- Responsabilidade dos departamentos remetentes:

    1. encaminhar um novo registro de protocolo no sistema com o estado inicial "Andamento".
    2. monitorar a sua solicitação visualizando o histórico do protocolo encaminhado.
    3. receber o re-encaminhamento dos protocolos, originalmente produzidos pelo próprio departamento, para fins de arquivamento.

- Responsabilidade dos departamentos destinatários:

    1. receber o encaminhamento e, caso o protocolo tenha sido enviado pela primeira vez, com o estado "Andamento", definir um novo estado para: "Suspenso", "Deferido" ou "Indeferido".
    2. receber o encaminhamento e não modificar o estado de imediato, permanecendo no estado "Andamento".
    3. receber o encaminhamento e re-encaminhar o protocolo sem modificar o estado.
    4. Receber o encaminhamento modificar o estado e re-encaminhar.
    5. Receber o encaminhamento e, caso o protocolo venha de um prévio re-encaminhamento, com o estado "Deferido" ou "Indeferido", modificar o estado para "Arquivado".

Esta abordagem de estados foi baseada no sistema de código aberto (open source) Mantis [45], que é um sistema de controle de mudanças, isto é, permite controlar, por exemplo, o desenvolvimento de software por meio de um controle de requisições protocoladas. A ferramenta trabalha com uma abordagem de ciclo de estados com envio automático de e-mails para os envolvidos com a requisição de mudança. O sistema de controle de protocolos seguiu a mesma abordagem de ciclos de estados com envio de e-mails tanto para o remetente quanto para o destinatário de um documento trafegado no sistema. O Mantis é originalmente desenvolvido na linguagem PHP [46] e com a persistência em banco de dados MySQL [39][40].

2.5 Definição do Processo de Desenvolvimento

A elaboração do sistema de controle de protocolos seguiu a abordagem incremental oferecida pelo modelo prescritivo espiral [9][10]. O modelo em espiral é um dos mais populares, onde seu comportamento evolucionário combina aspectos da natureza iterativa da prototipagem com modelo em cascata, conhecido também como ciclo de vida clássico, sistemático e seqüencial – passando unilateralmente por: comunicação → planejamento → modelagem → construção → implantação [9].

O modelo em espiral oferece a possibilidade de passar por todas as etapas disponíveis no modelo em cascata de forma a repetir os ciclos ao longo de todo o desenvolvimento do produto de software, possibilitando coletar, rastrear, corrigir e validar em vários ciclos incrementais [9][10].

O modelo foi escolhido para ser seguido pelo fato de ser iterativo com uma abordagem de análise de riscos, inerente ao processo de desenvolvimento do sistema, e permitindo construir versões do produto cada vez mais completas. As versões foram necessárias para validar e corrigir

falhas referentes a má interpretação por parte do analista ou mesmo ausência de especificação por parte do cliente. Para o sistema de controle de protocolos ocorreram 3 (três) ciclos iterativos com implantação no servidor do DSC - Departamento de Sistemas Computacionais e com posterior avaliação do cliente.

De acordo com o proposto na Figura 3 o processo em espiral segue:

- *Comunicação com o cliente*: considerado o marco inicial de cada ciclo e está relacionado com a definição e estruturação do problema.
- *Planejamento*: definição e/ou revisão de cronograma de atividades necessário para os ciclos de iteração.
- *Análise de riscos*: identificação dos possíveis riscos como também uma definição de uma gestão de controle dos riscos.
- *Engenharia*: montagem estrutural do projeto a partir da organização dos requisitos.
- *Construção*: codificação e liberação da versão do produto.
- *Avaliação*: realizada pelo cliente e/ou usuário oferecendo um "*feedback*" à equipe de projeto no tocante ao produto.

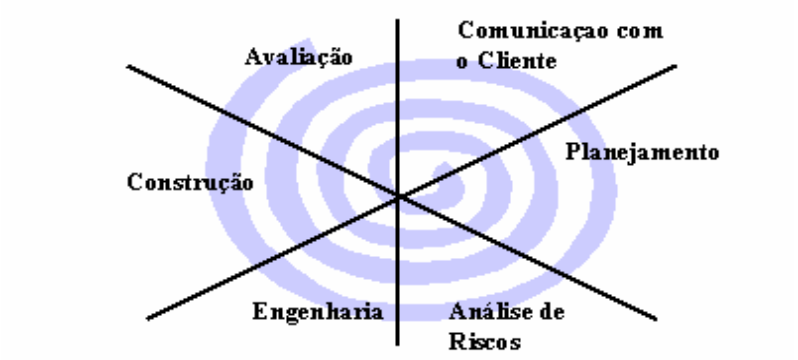


Figura 3. Modelo em Espiral.

A análise de risco é um item que sustenta a escolha do modelo prescritivo espiral. Esta atividade está basicamente composta de duas abordagens [9]:

- identificação dos riscos, tendo em vista que cada projeto tem seu próprio domínio de riscos;
- gestão dos riscos, relacionados à escolha de medidas e alternativas que viabilizem a continuidade do desenvolvimento de um projeto.

Os riscos na definição do sistema de controle de protocolos estão relacionados tanto a um escopo geral, participar da elaboração de um sistema constituído por uma única pessoa a desenvolver o produto, quanto específicos, referente às rodadas iterativas do modelo em busca de produzir uma versão estável do produto. Os riscos gerais na elaboração do sistema estão definidos na Tabela 3 e os riscos incrementais estão na Tabela 4.

Tabela 3. Riscos gerais para elaboração do sistema.

Riscos		
Tipo	**Descrição**	**Ação de prevenção**
Gerencial	Baixa experiência em gerência de projeto no que diz respeito à organização de cronogramas, reuniões e coordenação.	Estudo da disciplina de engenharia de software em busca de experiências apresentadas na literatura.
Análise	Pouca vivencia na atividade de um analista de sistema.	Capacitação por meio de curso de análise de requisitos, com o intuito de coletar e documentar de forma adequada.
Construção	Tecnologia adotada em pleno aprendizado durante o desenvolvimento.	Antecipação do aprendizado, utilizando fases anteriores a construção à obtenção de domínio nas tecnologias.

Quanto aos riscos incrementas ao modelo seguem na Tabela 4:

Tabela 4. Riscos incrementais na elaboração do produto.

Riscos		
Fase	**Descrição**	**Ação de prevenção**
Análise	• Reuniões com o cliente com espaçamento de 15 dias e freqüente remarcação. • Resistência por parte do cliente em ler e argumentar sobre as especificações geradas principalmente referentes à mudança de versão de documentos. • Perda do apoio da direção do departamento.	• A adoção da prototipagem para fornecer versões rápidas do produto em busca de maior aproximação com o cliente como também de críticas e maiores participações do mesmo.
Projeto	• Desvio de construção referente a má interpretação dos requisitos provenientes do cliente. • Utilização de soluções não homologadas. • Retrabalhos por falta de validação. • Atrasos na exibição das versões do produto. • Falta de acesso direto ao ambiente de implantação para fins de configuração.	• Intensificação do canal de comunicação com o cliente. • Agilização das respostas aos questionamentos. • Negociação novas datas para o processo de validação.
Técnico	• Definição da interface gráfica apropriada para diferentes usuários. • Possibilidade de não garantir a manutenção do produto em tempo estimado.	• Identificação do domínio da aplicação e dos objetivos dos utilizadores finais. • Documentação do produto beneficiando o engenheiro de software a identificar o(s) ponto(s) de alteração.

2.6 Casos de Uso do Sistema de Controle de Protocolos

Os casos de uso (*Use Cases -UC*) têm por finalidade relatar o fluxo de operação do sistema de acordo com o ponto de vista dos usuários finais, ou seja, representam uma história formal de como navegar e operar em um produto funcional [9][13].

Os modelos mais utilizados e desenvolvidos para o projeto consistem de uma representação em texto narrativo e de uma representação diagramática. Os atores envolvidos no sistema da Poli são o responsável pelo departamento e funcionários em geral.

Os casos de uso (UC) foram definidos a partir das seguintes funcionalidades :

- Encaminhar documento, permitindo que um funcionário remetente envie uma solicitação ou documento.
- Receber uma solicitação ou documento, permitindo ao funcionário destinatário comprovar o recebimento.
- Reencaminhamento de documento, permitindo ao funcionário destinatário de uma solicitação, previamente realizada, repassá-la a outro destinatário ou retornar para quem inicialmente a originou, ou seja, para o remetente.
- Modificar a situação de um documento, permitindo ao funcionário que encontra-se com a solicitação, no caso o destinatário, definir um novo status de atendimento, podendo ser em *Andamento, Suspenso, Deferido, Indeferido, Arquivado.*
- Cancelar um encaminhamento, permitindo ao funcionário remetente de uma solicitação desistir da atividade.
- Arquivar um documento, representando o status final de um documento circulado entre departamentos.
- Manter funcionários, relacionado com o cadastro e consulta de funcionários no sistema
- Manter departamentos, relacionado com o cadastro e consulta de departamentos no sistema.
- Visualizar histórico de documentos, permitindo aos funcionários verificar o andamento do documento e sua eventual mudança de situação.
- Guia do usuário, representando um tutorial da política de uso do sistema.

Dentre os requisitos não-funcionais, foram levantados:

- Usabilidade, o sistema deve ser de fácil manuseio pelos usuários devido ao nível variado dos funcionários da Escola quanto ao domínio de informática.
- Distribuição, o acesso ao sistema deve ser geral e livre a toda rede Intranet.
- Segurança, o acesso ao sistema deve ser restrito a funcionários previamente cadastrados.

Um documento de caso de uso é elaborado na fase de análise e é estruturado pelo analista de sistema. O documento tem o intuito de relatar como um ou mais requisitos deve (em) se comportar no ambiente funcional. Dentre os objetivos deste tipo de documento tem-se a preocupação em relatar como executar uma ação ou atividade e não em como descrever a forma de implementação das mesmas. Não há um modelo pré-estabelecido para este tipo de documentação, no entanto, há itens que são mais comuns de serem utilizados e encontrados,

como, por exemplo, a descrição de fluxo primário, secundário, interfaces associadas, dente outros.

O modelo adotado para a elaboração do documento de caso de uso para o sistema de controle de protocolos está constituído dos itens tradicionais para este tipo de artefato. Normalmente a elaboração do documento ocorre após a definição do escopo do sistema, ou seja, após restringir o domínio do problema. A Figura 4 exibe o diagrama de casos de uso relacionados com o fluxo básico de controle de protocolos, o qual representa um "mini-mundo" do sistema desenvolvido exibindo todas as suas possíveis atividades de execução. Nesta figura não estão sendo contemplados o processo de autenticação e nem o guia de ajuda, mas ilustra as principais funcionalidades coletadas e separadas disponíveis para entrar em nível de detalhes por meio de uma documentação. A partir desta representação diagramática é possível identificar os atores, como também, o relacionamento entre cada ator e os casos de uso do sistema.

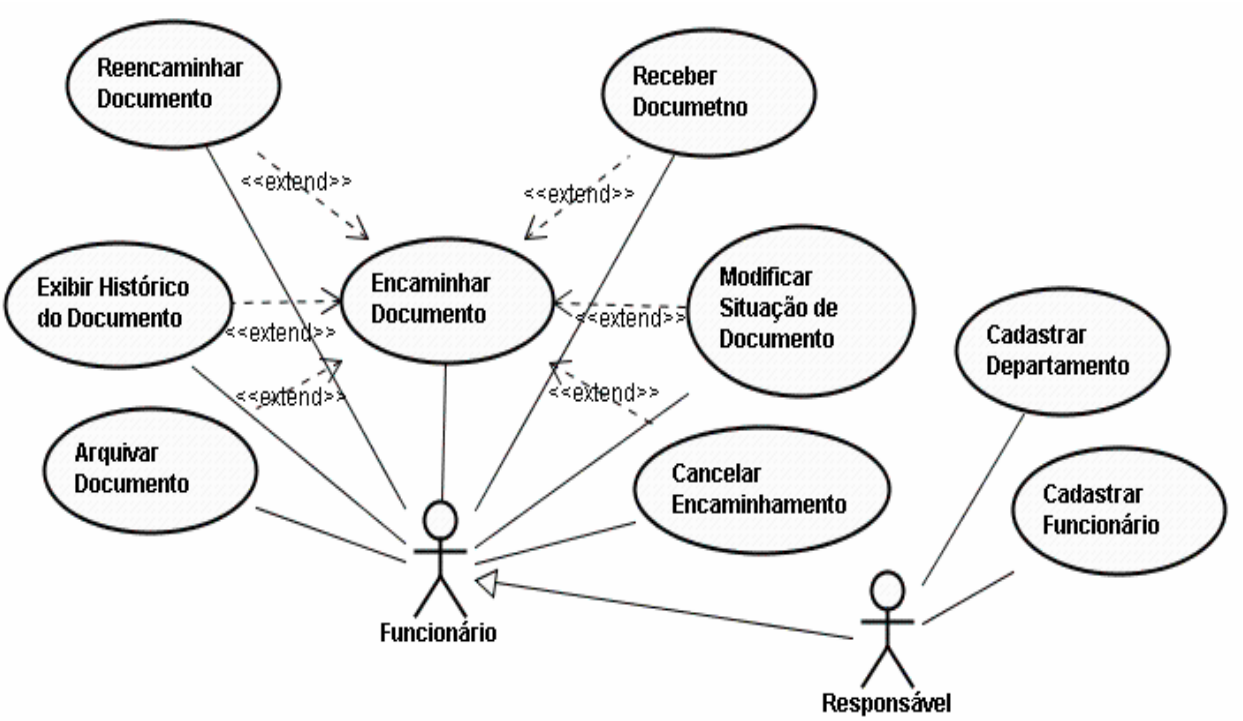


Figura 4 . Casos de uso básicos do controle de protocolos.

Os documentos de casos de uso foram elaborados a partir desta representação diagramática, Figura 4, e estão constituídos basicamente das seguintes seções:

1. *Descrição*: tem o intuito de relatar a finalidade do caso de uso em questão.
2. *Pré*-Condições: define o que deve ser verdade antes de um ator iniciar a execução do caso de uso.
3. *Fluxo Básico ou Principal:* concentra-se inicialmente na funcionalidade básica e central do caso de uso.
4. *Subfluxos:* representando um dos possíveis caminhos provenientes do fluxo básico ou principal.

5. *Exceções:* representando as possíveis violações ao fluxo do caso de uso.
6. *Pós-Condições:* define o que deve ser verdade após o ator realizar o caso de uso.
7. *Regras de Negócios*: define as regras adicionadas para a correta execução do caso de uso.
8. *Detalhamento das interfaces gráficas*: define o layout gráfico e os tipos de dados a serem coletados ou exibidos.
9. *Comandos*: descrevem as ações proveniente do evento de acionar botões e links.

No Apêndice A encontra-se um documento de caso de uso referente ao *Exibir Histórico do Documento*, este caso de uso permite ao usuário visualizar o fluxo de encaminhamento do documento, como por exemplo, possibilidade de rastreamento das últimas operações realizadas, permitindo verificar se o documento foi recebido pelo departamento destinatário, se houve modificação na situação ou mesmo se o documento fora reencaminhado.

Capítulo 3

Tecnologia

Neste capítulo serão apresentados os recursos tecnológicos utilizados no desenvolvimento do sistema: a plataforma J2EE, os frameworks Struts e Hibernate, a API Servlets e JSP.

As aplicações de negócio que são baseadas no ambiente Web possuem grandes possibilidades de apresentarem características em comum, principalmente no que diz respeito à arquitetura e infra-estrutura. Tais características permitem que o desenvolvimento para Web torne-se, de certa forma, padronizado e como boa conseqüência deste fato, têm-se produtos com grandes capacidades de reusabilidade e flexibilidade [9][18]. A plataforma Java, que adota orientação a objetos (OO), é bem aceita pelos engenheiros de software que utilizam Java como ferramenta de trabalho para seguir uma construção com base nos padrões. Os padrões de projetos para o ambiente Web são vastos nas literaturas de desenvolvimento de software e sistemas[18][20]. Os padrões de projetos recomendados pela plataforma J2EE [18][22] são aqueles que fizeram e fazem parte da solução de vários projetos de sucesso. Os padrões têm o intuito de prover um poderoso mecanismo de reusabilidade e ajuda aos desenvolvedores e arquitetos de software a fim de evitar “reinventar a roda” [9][20].

3.1 A plataforma J2EE

A plataforma J2EE (*Java 2 Enterprise Edition*) é constituída de um conjunto de especificações e práticas que provêem serviços às aplicações empresariais (*enterprise applications*). A plataforma é propícia a uma abordagem arquitetural em multicamada, que consiste de uma separação estruturada de responsabilidades que propicia a reutilização de código fonte como também a legibilidade, detecção e correção de erros [19][22][24], e disponibiliza acesso a diversos serviços de infra-estrutura como: segurança, controle de transação, comunicação entre camadas, dentre outros.

A plataforma J2EE estende a linguagem Java por meio de padrões, de forma a atender as aplicações distribuídas, e dispõe de um ampla API que independe dos fabricantes das tecnologias atreladas com a especificação.

Um cliente web, normalmente um browser, inicia uma iteração de solicitação/resposta com o servidor Web via protocolo HTTP. O servidor, por sua vez, invoca o container Web para atender requisições provenientes deste protocolo. De acordo com o tipo de aplicação desenvolvida e dos recursos exigidos pela aplicação, o container web, que lida com páginas JSP e Servlet, pode invocar outro container denominado EJB, que lida com controle de transações customizadas, balanceamento de carga, isto é, a possibilidade de distribuir a mesma aplicação em mais de um servidor na rede, ou mesmo, de recuperar recursos para a aplicação em servidores distribuídos geograficamente. Ambos os containers têm a possibilidade de se conectar com servidores de banco de dados, sendo que o EJB possibilita uma aplicação a interagir dinamicamente com mais de um repositório de dados de forma mais eficiente, pelo fato de disponibilizar suporte de infra-estrutura para acesso distribuído e concorrente [24].
A Figura 5 exibe o fluxo de operação no ambiente de servidor utilizado por uma aplicação J2EE que necessita dos dois containers: WEB e EJB.

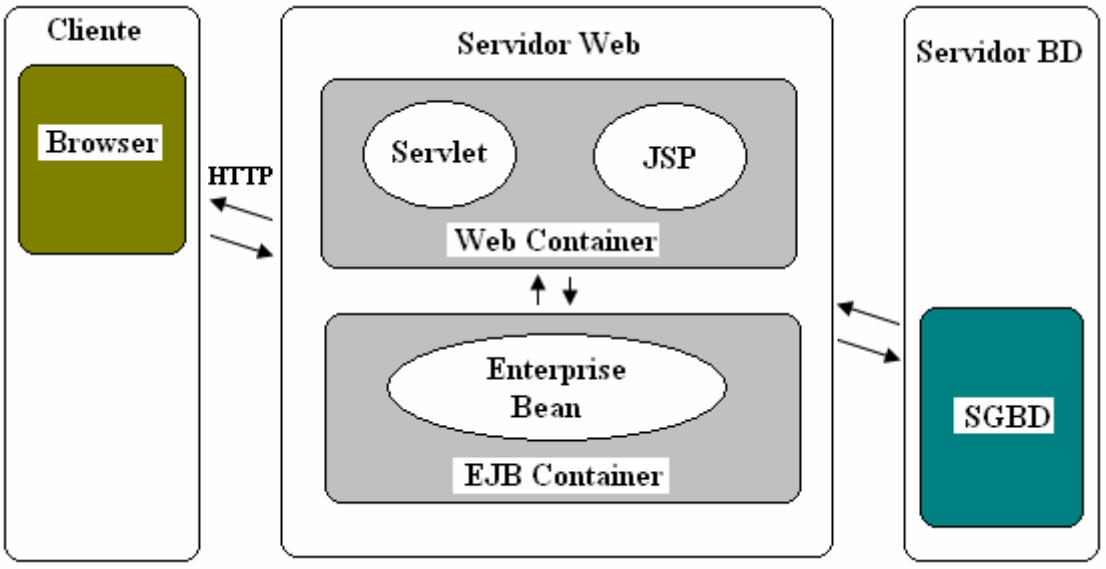


Figura 5. Aplicação J2EE utilizando os containers WEB e EJB.

Particularmente, a utilização de Enterprise Java Beans (EJB) é recomendada quando os requisitos da aplicação traduzem necessidades complexas de acesso concorrente, gerenciamento de transação, de segurança e processamento distribuído [18][19].

Para o projeto em questão, que não necessita destes recursos avançados do EJB, por se tratar de uma aplicação que utiliza apenas um único banco de dados e por ser executado em um único servidor, adotou-se, com relação ao servidor Web, apenas o Web container que disponibiliza bibliotecas para manipular com Servlets e JSP e suporte ao protocolo HTTP para comunicação cliente/servidor. Deste modo está sendo utilizado apenas um subconjunto da API J2EE.

3.2 Frameworks como Elementos de Produção

Os frameworks são estruturas de suporte ao desenvolvimento de software, oferecendo uma organização nos projetos a serem construídos e uma padronização imposta pela sua estrutura. Um framework é um esqueleto arquitetural, denominado arcabouço de uma determinada abstração de software [9]. Seu caráter reusável permite sua utilização para atender a um domínio do problema de construção de software. É constituído por uma coleção de classes ou componentes em cooperação e sua utilização se dá por meio de pontos de extensões. A partir dos pontos de extensões, os desenvolvedores constroem a real arquitetura imposta pelo framework.

Uma API (*Application Programming Interface*) é uma biblioteca de funções e rotinas que são invocadas por uma aplicação, isto é , são pequenas funcionalidade que os desenvolvedores utilizam de forma isolada ou combinada para montar um comportamento do sistema. Não é responsabilidade de uma API oferecer uma estrutura elaborada de acesso e manuseio das funções num sistema. Esta tarefa está relacionada com a necessidade dos desenvolvedores em montar uma arquitetura de projeto, a proteger ou encapsular, quando necessário, as reais funcionalidades de uma certa biblioteca. A diferença básica entre uma API e um Framework está no fato de que as bibliotecas oferecem funções e rotinas a serem invocadas pelo sistema e o framework provê classes e componentes genéricos num cenário montado e pronto para uso e, por sua vez, cabe ao framework invocar os pontos de extensões.

Uma vez compreendida a estrutura arquitetural e os contratos definidos para os pontos de extensões de um framework qualquer, pode-se dizer que é notório a praticidade de utilização e ganho de tempo em produção, principalmente em larga escala.

Existem vários frameworks para o desenvolvimento de aplicações web, cada um com sua própria política de extensão e neste trabalho estão sendo adotados o framework Struts, versão 1.1 [28] para o desenvolvimento da camada de apresentação, por se tratar de um framework de produção em grande escala e bastante disseminado no mercado e o framework Hibernate, versão 2.1 [30] para a camada de persistência, por oferecer uma abordagem orientada a objetos e pela abstração oferecida no manuseio da persistência dos dados.

3.2.1 Framework Struts

O Struts é um framework desenvolvido para construir e manipular um sistema web. Seu arcabouço é definido na arquitetura MVC (*Model-View-Controller*)[24][26]. A sua organização interna está focada principalmente para a camada de apresentação e controle. Os componentes de visão e controle incorporam uma série de recursos para facilitar a implementação da camada de apresentação. Componentes de visão representam as interfaces de um sistema com seus usuários enquanto os componentes de controle são responsáveis por selecionar e disparar as operações de negócio correspondentes às requisições de usuários realizadas a partir da visão.

A elaboração da camada de apresentação é composta de várias tecnologias atreladas, como exemplo, a linguagem de marcação HTML [33][34][35], JSP [19][24], JavaScript [36], Taglib [37][38] , JavaBeans [32] dentre outros. Desde a sua definição, o Struts foi projetado para agregar esses e outros recursos de forma a manter um sistema funcionando em harmonia. A sua arquitetura estrutural é composta de:

1. O modelo (*model*) está associado a camada de negócio. Esta parte da arquitetura é uma lacuna a ser implementada ou acoplada com as regras de negócio de todos os projetos que adotam o uso do framework.

2. A visão (*view*) está associada às páginas web, tanto com as estáticas (HTML) quanto com as dinâmicas (JSP), permitindo agregar bibliotecas de tags de forma a evitar adicionar regras de negócio e código Java nas páginas dinâmicas.
3. O controlador (*controller*) tendo a finalidade de receber as solicitações dos usuários e interligar a visão com a camada de negócio.

As principais funcionalidades do componente Struts são:

- Realizar a função de controle genérico configurável via arquivo XML.
- Fornecer bibliotecas de tag (taglibs) que facilitam a criação de componentes de apresentação.
- Fornecer um mecanismo declarativo para o tratamento de exceções.
- Fornecer um mecanismo próprio para facilitar a internacionalização de aplicações.

O Struts possui um único servlet denominado ActionServlet que é a porta de entrada para o ambiente Struts e por sua vez é a porta de comunicação com o web container. O ActionServlet é configurável a partir de um arquivo XML [23] descrito no próximo capítulo. Adota o padrão *Front Controller* [22], padrão baseado em aplicações cliente/servidor onde tem a finalidade de ser o canal de entrada na aplicação. Dentre suas responsabilidades o ActionServlet realiza o roteamento das requisições HTTP provenientes dos usuários para as classes de ações correspondentes.

Uma aplicação que adota a arquitetura MVC com Struts, após receber uma solicitação ("request") do browser do usuário encaminha a solicitação para um controlador de requisições. Este controlador invoca a camada de negócio da aplicação para incluir ou recuperar dados, e em seguida define qual a página Web será exibida ao usuário. As página de visão podem apresentar taglib que consiste de código Java dinâmico com uma sintaxe próxima do HTML. A Figura 6 ilustra os três itens da arquitetura MVC em um abordagem diagramática com seu fluxo de operação baseado no Struts.

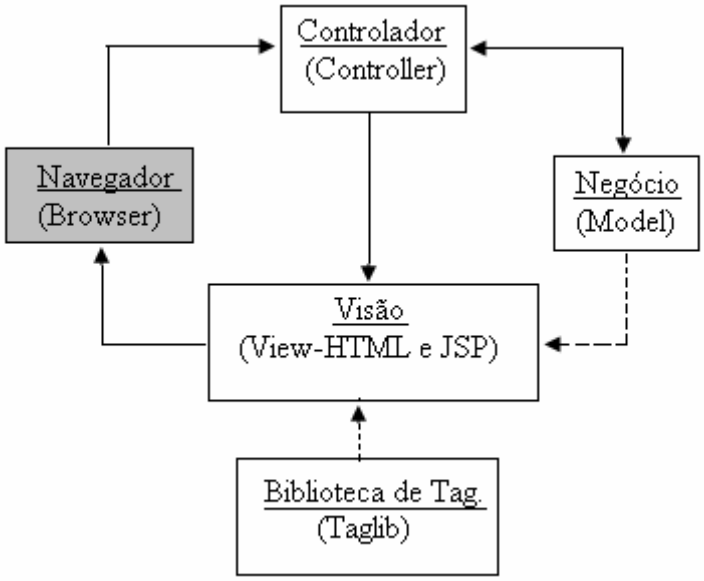


Figura 6. Estrutura de navegação lógica no Struts.

O Struts pode ser customizado estendo-se direta ou indiretamente suas classes de ação e suas taglibs. No projeto de controle de protocolos adotou-se como padrão para a criação de classes de ação estender-se a classe *br.poli.upe.web.controle.BaseDispatchAction*. Esta classe apresenta a responsabilidade de verificar a permissão de usuários na invocação de métodos de negócio como também validar a sessão do usuário autenticado.

3.2.2 Framework Hibernate

Muitos dos Sistemas de Informação Empresarial (SIE) utilizam sistemas de gerenciamento de banco de dados baseados em SQL [3]. A larga adoção da linguagem de programação Java alavancou a utilização do paradigma orientado a objetos no desenvolvimento de software. No entanto, a representação de dados em tabelas em um sistema relacional é bastante diferente da visão de objetos adotados pela orientação a objetos.

O Mapeamento Objeto/Relacional (ORM) é uma solução automatizada para aproximar os dois contextos: orientado a objetos e o relacional. O Hibernate é um framework de desenvolvimento da camada de persistência totalmente implementado em Java que adota a abordagem ORM. Tem como objetivo proporcionar uma elaboração de software, orientado a objetos para uma camada do software considerada crítica a qualquer sistema que necessite de uma persistência.

O framework possui uma linguagem de consulta própria, denominada HQL (*Hibernate Query Language* - Linguagem de Consulta do Hibernate), que expressa a consulta de objetos. A elaboração da camada de persistência com o Hibernate torna-se um processo automatizado e prático comparado com a tradicional abordagem da API JDBC [47], cuja estrutura de transação e consultas são montadas pelo desenvolvedor da persistência, diferentemente do que ocorre com o Hibernate, o qual possui uma estrutura já pronta para uso. A utilização da API JDBC tem maior probabilidade de gerar erros de construção e até mesmo de desempenho, pelo mau uso da SQL ou mesmo da própria biblioteca.

O framework permite uma aplicação orientada a objetos conservar o estado de seus objetos em um banco de dados, e possibilita que os mesmos objetos sejam recuperados ou recriados em um futuro próximo, quando necessário. É oferecida também, a persistência de todos os relacionamentos dos objetos e suas eventuais heranças [31], fato bastante interessante, principalmente levando em consideração que os bancos de dados SQL não têm uma noção de herança ou mesmo um modo para representar uma associação polimórfica.

A responsabilidade deste framework é de mapear as tabelas do modelo relacional para classes da linguagem Java, isto é feito por meio de arquivos XML. A partir da representação das classes Java em arquivos de formato XML o Hibernate gerencia todo o fluxo de objetos persistentes tornando a aplicação independente do Sistema Gerenciador de Banco de Dados (SGBD).

As aplicações que adotam o uso do Hibernate devem definir classes Java que representam as tabelas e relacionamentos do banco de dados, denominados de objetos persistentes. O Hibernate, por meio de mapeamento em arquivos XML relaciona tais classes nas tabelas correspondentes. Por meio do arquivo *hibernate.properties* é possível reunir todas as informações necessárias de inicialização do Hibernate, como exemplo,, identificação do banco de dados a ser utilizado, número de conexões possíveis simultaneamente, definição da lista de classes Java mapeadas em XML que estão no domínio do framework, dentre outros itens [31].

A Figura 7 ilustra a estrutura de uma aplicação desenvolvida com Hibernate seguindo a descrição feita anteriormente.

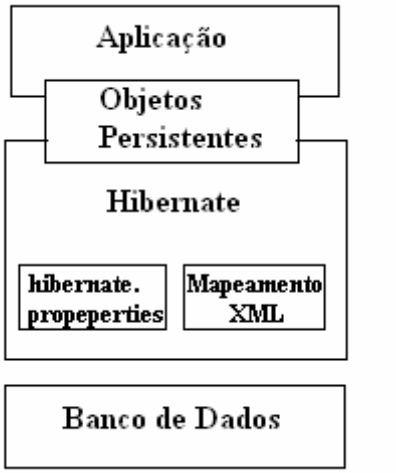


Figura 7. Aplicação desenvolvida com Hibernate.

Nota-se pela figura acima que o Hibernate é uma camada intermediaria entre a aplicação e o repositório de dados.

3.3 Servlets e JSP

Os servlets são classes em Java que estendem parte das funcionalidades do “servlet container”. O “container” é uma extensão do servidor Web que provê serviços de rede para permitir a comunicação entre os servlets e os clientes Web. É responsabilidade do container gerenciar o ciclo de vida dos servlets e dar suporte ao protocolo HTTP [6][7]. Os servlets interagem com clientes Web via requisições (*request* e *response*) e são carregados apenas uma única vez no container, permanecendo em memória disponíveis para atender novas solicitações. Os servlets trabalham em conjunto com o container no processamento de requisições e normalmente atendem as solicitações dos clientes por meio dos métodos GET e POST do protocolo HTTP.

O JSP consiste de uma página HTML com código dinâmico embutido em scriplets e/ou taglibs. Quando uma página JSP é solicitada pela primeira vez, ela é transformada em um servlet através de um processo conhecido como tradução [24]. Após esta transformação, a servlet correspondente é acionada de forma a tratar a requisição. Os acessos subseqüentes à página JSP serão atendidos pela servlet correspondente. A utilização dos servlets são mais adequados para a realização de funções envolvendo lógica ou geração de conteúdo binário. As páginas JSP são mais adequadas na exibição de dados em formato HTML, lidando com questões relacionadas com a apresentação das informações.

Capítulo 4

Desenvolvimento

Neste capítulo serão abordados a construção e fluxo de operação do sistema de controle de protocolos, testes de unidade, a representação e navegabilidade das principais telas.

4.1 Arquitetura do Sistema S@DPOLI

A definição e estruturação de uma arquitetura de projetos web tem uma forte dependência dos recursos disponibilizados de infra-estrutura [24]. A proposta inicialmente estabelecida para o desenvolvimento do projeto S@DPOLI foi de utilizar os recursos já existentes no departamento DSC, local onde será implantado o Sistema de Controle de Protocolos. O Departamento já disponibiliza de um ambiente web com o Tomcat [41] representando o Servlet Container [24][27] e banco de dados MySQL [40]. De posse destes recursos partiu-se para a definição arquitetural do projeto de forma a ser integrado ao departamento evitando potenciais incompatibilidades.

A arquitetura do sistema S@DPOLI foi constituída de uma abordagem em 3 (três) camadas: camada de apresentação, camada de negócio e camada de persistência e subdividida em duas partes distintas: o projeto SadApresentacaoPOLI e o SadProtocoloPOLI. O intuito da separação está relacionado tanto com o desacoplamento entre as etapas de construção quanto aos recursos distintos utilizados em cada uma das partes. A camada de apresentação, que constitui camada de visão, é representada pela parte SadApresentacaoPOLI, enquanto as camadas de negócio e de persistência são representadas pela parte SadProtocoloPOLI, formando dessa forma a arquitetura em três camadas de todo o sistema. Cada camada da arquitetura apresenta um domínio unilateral da camada subseqüente como ilustrada na Figura 8.

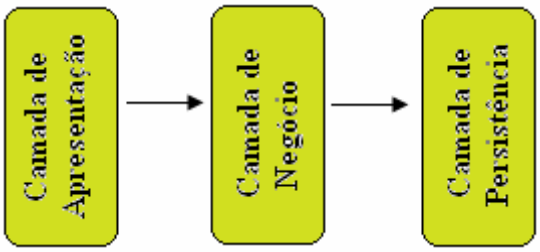


Figura 8. Estrutura de camadas.

A abordagem em duas partes distintas está relacionada ao fato da possibilidade de reutilizar uma das duas. O projeto SadApresentacaoPOLI, por exemplo, poderia ser substituído por um novo elemento web com outro framework de construção como o Java Server Faces (JSF) [42][43] ou mesmo definir a camada de apresentação para um ambiente *desktop*, baseado em Java Swing [44]. Note-se que a mesma linha de raciocínio pode ser tomada com relação ao projeto SadProtocoloPOLI, embora, muitas vezes, projetos relacionados às regras de negócio sejam difíceis de serem reutilizados devido a seu escopo e política de desenvolvimento estarem associados a uma estrutura imposta pelo cliente.

4.1.1 SadProtocoloPOLI

Nesta seção serão expostas a estrutura da camada de negócio e da camada de persistência do sistema como também a hierarquia de classes e descrição dos padrões utilizados nesta etapa de construção.

Para o desenvolvimento do SadProtocoloPOLI utilizou-se o J2SE 1.4 [25], que representa o Java básico e o Framework Hibernate 2.1 [30], conhecido como ORM. A construção reflete o entendimento de negócio proveniente do cliente (departamento DESAF).

4.1.1.1 Camada de Negócio

A camada de negócio é responsável pela implementação dos casos de uso do sistema, obedecendo às regras de negócio. É responsável também pela comunicação com a camada de persistência, que responde às requisições da camada de negócio obtendo informações do meio persistente, traduzindo para objetos e atualizando o meio persistente quando necessário. Na estrutura apresentada aqui, a camada de persistência é representada pelo framework Hibernate com a adoção do padrão DAO (*Data Access Object*) [18], adotado para abstrair e encapsular o acesso direto aos repositórios de dados. Para o projeto a camada de negócio está constituída basicamente de Fachada, Controlador, Entidade e classe de consulta como ilustrado na Figura 9.

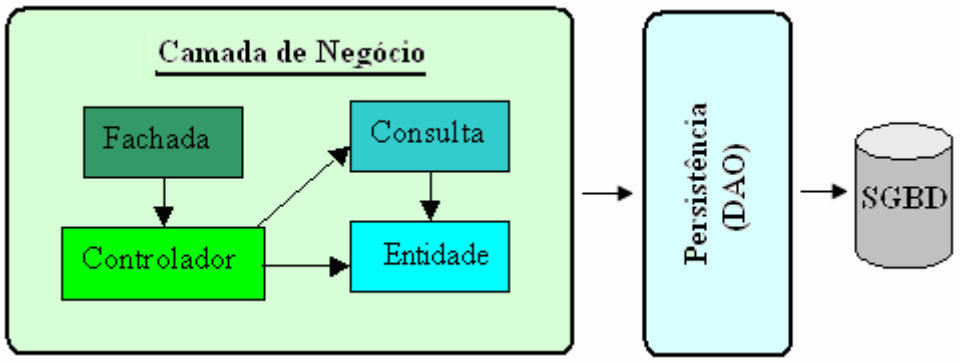


Figura 9. Camada de Negócio e Persistência.

A fachada representa a "porta de entrada" da camada de negócio, agrupando todas as suas operações em um único componente e tem a função de delegar as operações aos controladores. A definição da fachada segue o padrão *Façade* [20], onde este padrão enfatiza a necessidade de

encapsular os detalhes de implementação da camada de negócio, reduzindo o acoplamento e facilitando a manutenção do código.

Controladores são as classes responsáveis pela implementação dos casos de uso e pela centralização do acesso a determinada(s) entidade(s) ou recurso(s). Seus métodos são invocados pela fachada do sistema ou por outros controladores. Geralmente existe uma classe controladora por classe de entidade. Os controladores têm a responsabilidade de se comunicar com a camada de persistência através da biblioteca ou framework adotado para a implementação da persistência.

Os controladores também são responsáveis por todas as validações de dados em inclusões e atualizações. É recomendado efetuar todas as validações na camada de negócio, verificando formatos, obrigatoriedade de preenchimento de campos, domínio de valores, dentre outros, mesmo que os dados já tenham passado por validação na camada de apresentação.

Para o projeto em questão estão sendo adotados os seguintes controladores:

- ControladorDoc: referente a entidade *Documento* e a entidade de relacionamento *EncaminhamentoDoc*, tendo a finalidade de gerenciar o fluxo de documentos entre os departamentos.
- ControladorFunc: referente a entidade *Funcionário*, tendo por finalidade o gerenciamento de *manter funcionário.*
- ControladorDep: referente a entidade *Departamento*, tendo por finalidade o gerenciamento de *manter departamento.*

As entidades são classes de negócio representando elementos persistentes do sistema. São representadas como JavaBeans, possuindo propriedades que podem ser acessadas por meio de métodos *getter* e *setter* [32]. As instâncias dessas classes são materializadas pelo Hibernate na ocasião das consultas e usadas por ele para atualizar o meio persistente.

A principal função dos *Values Object*s -VOs é carregar dados de entidades isolando os clientes da implementação da camada de negócio [18][22]. Há pelo menos uma classe *Value Object* para cada classe de entidade. Os VOs são serializáveis e podem atravessar a fachada, tanto como resultado de alguma operação quanto como parâmetro, ao contrário das entidades que estão restritas à persistência e visíveis apenas aos controladores da camada de negócio. Os VOs utilizados no sistema trafegam dados formatados tanto para serem expostos ao cliente quanto compatíveis com a persistência no banco de dados.

As classes de consulta, também conhecidas como filtros, são classes especiais que têm como objetivo armazenar e transportar informações usadas para consultas na camada de persistência. São *Value Objects*, e trafegam dados formatados. São serializáveis e podem transitar pela fachada. Geralmente são instanciadas pela camada de apresentação e usadas como parâmetros de busca. Podem ser instanciadas também na camada de negócio para uso interno. Normalmente não retornam à camada de apresentação.

A vantagem de utilizar as classes de consulta é que uma alteração no conjunto de critérios de pesquisa não cria um efeito cascata de alterações em outras classes do sistema, como por exemplo, na fachada e nos controladores. Quando mudam os critérios de pesquisa, basta alterar a classe de consulta associada. Além disso, as consultas se tornam mais orientadas a objeto, permitindo aos desenvolvedores da camada de negócio abstrair das especificações impostas pelo meio persistente. A Figura 10 exibe a representação a navegabilidade de *Value Object* entre as camadas do sistema.

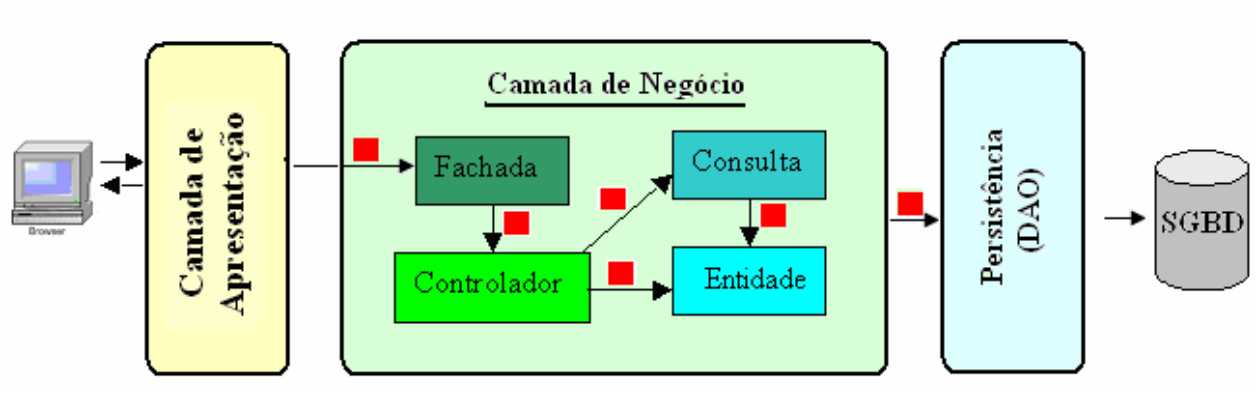


Figura 10. Navegabilidade do *Value Object* ( ) entre camadas.

Na arquitetura atual é responsabilidade da fachada controlar as transações do sistema, deste modo, a abertura e fechamento de conexão com o banco de dados é feito no nível mais alto, possibilitando aos controladores interagir com a camada de persistência mais de uma vez reutilizando uma única conexão. Sendo assim, falhas ou violação à regra de negócio em uma das operações realizada pelo controlador levará a fachada a desfazer todas as prévias alterações no banco de dados, restaurando o estado inicial do banco.

4.1.1.2 Camada de Persistência

O objetivo da camada de persistência é integrar a camada de negócio ao meio persistente sem criar acoplamento entre a aplicação e sua base de dados. Em alguns casos esta parte do sistema é também chamada de Camada de Integração [51].

Existem diferentes formas de acesso a banco de dados, por exemplo, por meio de SQL ou HQL, adotando padrões de projeto ou não. A camada de persistência do S@DPOLI resolve o problema da comunicação com diferentes bases de dados, adotando um conjunto padrão de interfaces, pelo qual as classes da camada de negócio obtêm acesso aos dados. As classes de repositório são um caso de implementação do padrão DAO. Elas encapsulam todo o código necessário para o acesso à fonte de dados, incluindo o HQL [31] e qualquer outra codificação.

O código das classes da camada de negócio possui acesso às interfaces da persistência, mas não às classes concretas que as implementam. Desta maneira fica garantida a independência da aplicação em relação a sua base de dados.

A base de dados do sistema S@DPOLI é constituída de tabelas no modelo clássico entidade – relacional [51], representado na Figura 11. Para o domínio de cadastro e armazenamento na aplicação têm-se as seguintes tabelas:

- *SAD_DEP :* cadastro de departamentos.
- *SAD_FUNC:* cadastro de funcionário.
- *SAD_FUNC_LOGIN:* cadastro de senha para autenticação de usuário.
- *SAD_FUNC_PAPEIS:* cadastro do perfil de cada usuário.
- *SAD_FUNC_PERFIL:* lista de perfis reconhecidos no sistema.
- *SAD_DOC_DEP:* cadastro de documentos.
- *SAD_SIT_DOC:* cadastro do ciclo de estados do documento.
- *SAD_ENC_DOC:* cadastro de encaminhamento de documentos.

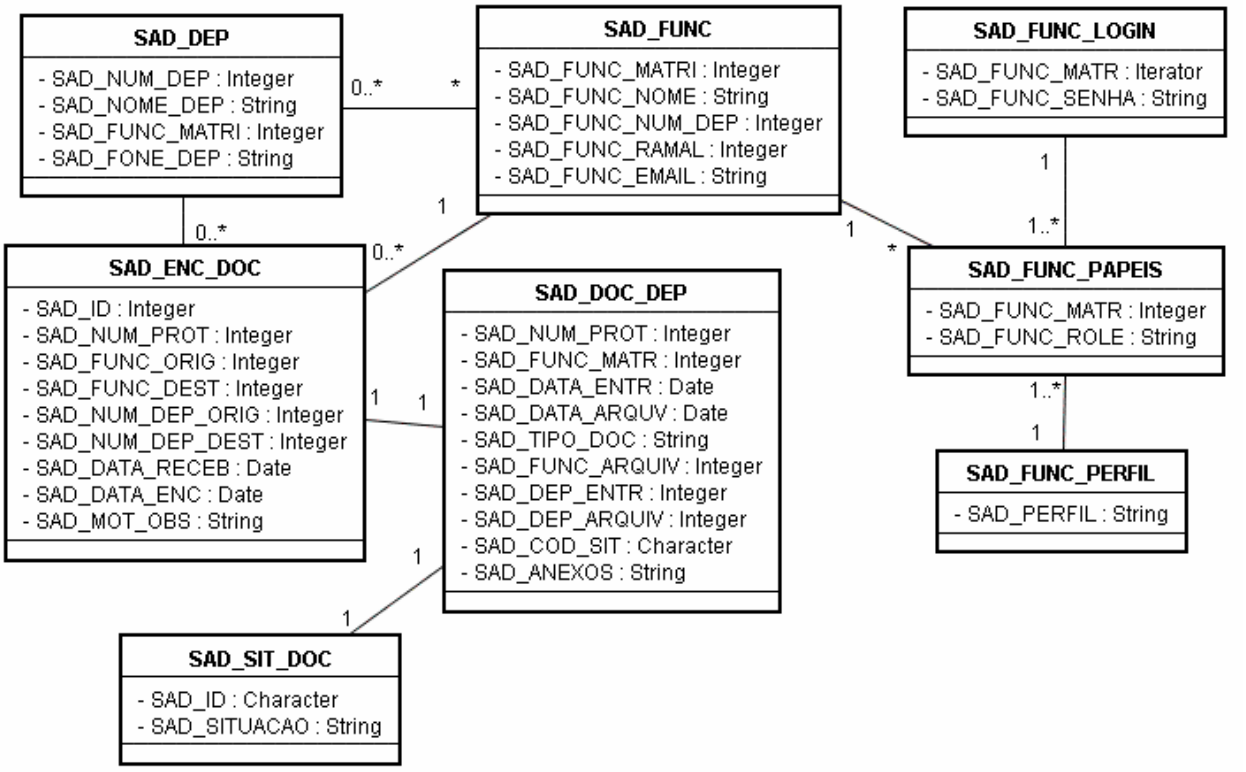


Figura 11 . Modelo de Dados do sistema S@DPOLI.

4.1.1.3 Diagramas de Classes

Para a hierarquia de classes adotada ao projeto SadProtocoloPOLI foram desenvolvidas duas fachadas de comunicação com o projeto SadApresentacaoPOLI. A primeira refere-se à Fachada de Documentos que concentra os casos de uso referentes ao fluxo de documentos e a segunda fachada, de Cadastro, refere-se ao fluxo básico de cadastro de funcionários e departamentos.

A opção em definir duas fachadas para o projeto está associada com o desacoplamento de responsabilidade e simplificação na manutenção. Tanto a estrutura de classes relacionada com Documento quanto com Cadastro refere-se à camada de negócio e camada de persistência do sistema, constituídas dos conceitos anteriormente citados: fachada, controlador, filtro de consulta, VO e DAO. As Figuras 12 e 13 representam a hierarquia de classes.

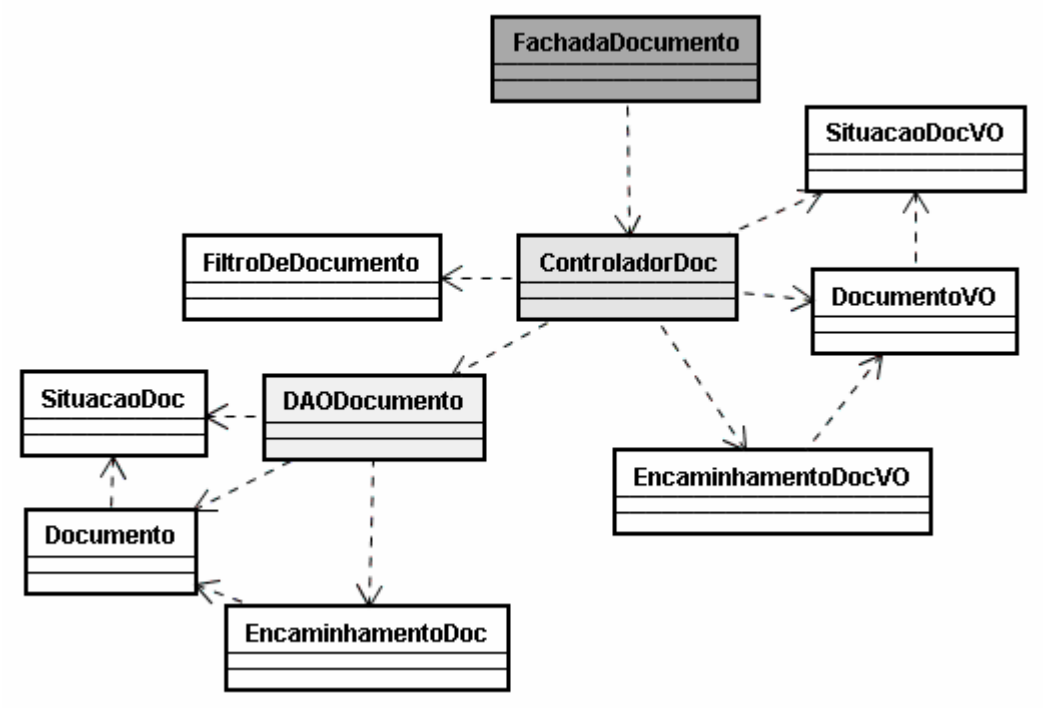


Figura 12. Diagrama de classes de Documentos.

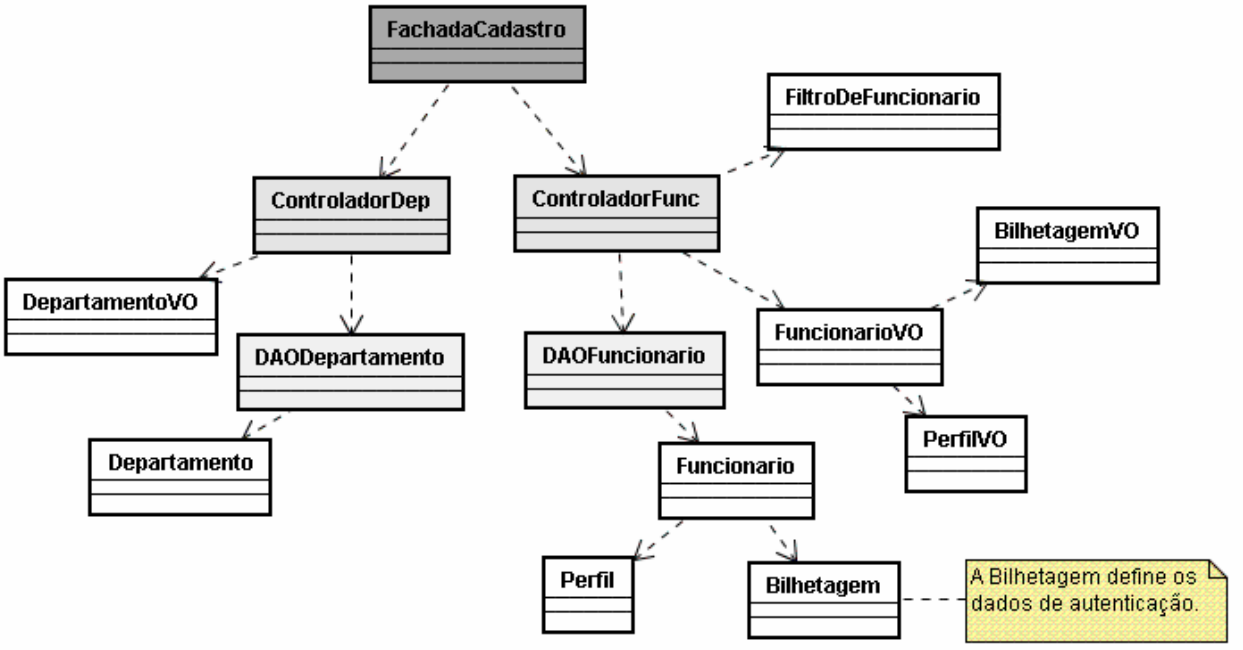


Figura 13. Diagrama de classes de Cadastro.

A hierarquia de classes de exceções foi desenvolvida de forma a oferecer maior legibilidade ao problema a ser tratado, em termo de código auto-explicativo. As exceções contribuem para materializar restrições às regras de negócio não atendidas no sistema. A hierarquia aqui proposta permite definir informações aos usuários do sistema de acordo com o tipo de violação definida internamente à aplicação.

Dentre as falhas internas ao sistema tem-se:

- exceção de infra-estrutura: falhas na conexão com o banco de dados ou falha na comunicação com componentes.
- exceção de negócio: violação as regras estabelecidas internamente ao sistema.

A Figura 14 ilustra o diagrama de exceções definidas ao sistema S@DPOLI.

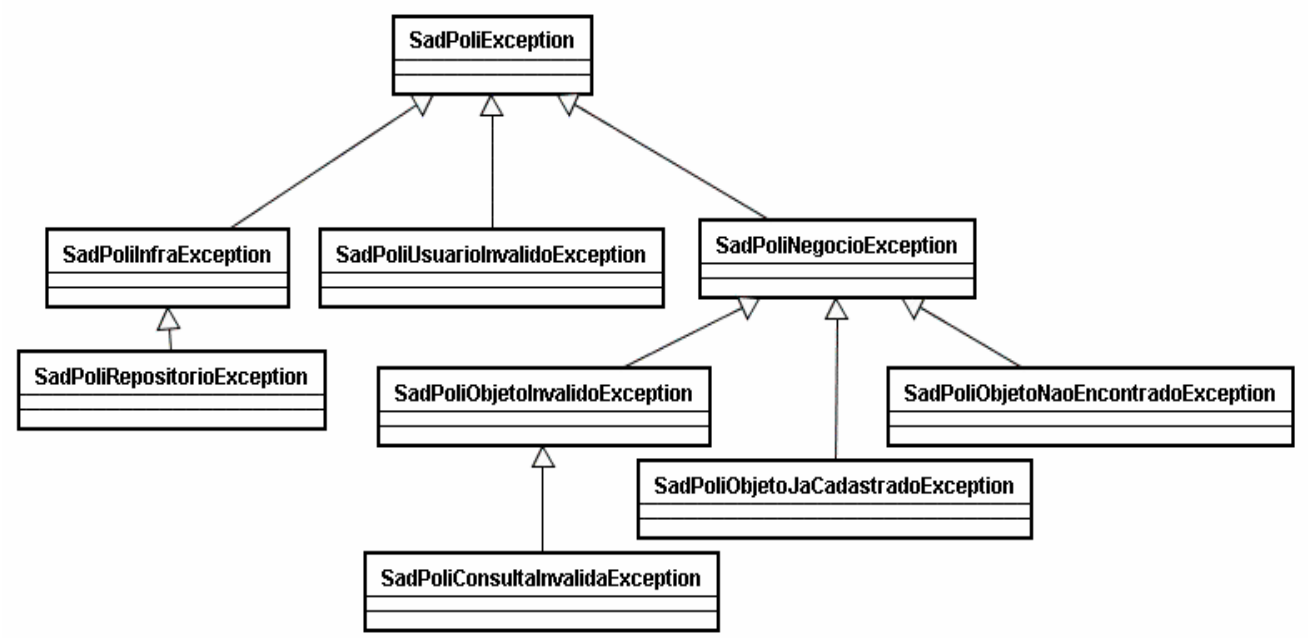


Figura 14. Diagrama de Exceções.

4.1.2 Projeto SadApresentacaoPOLI

Nesta Seção é relatada uma visão geral dos Servlets e JSP [24]; a montagem do projeto web com a estruturação de pacotes válidos para adição no servidor Web de forma a ser reconhecido como uma aplicação web, como também a visão de componentes utilizados no projeto.

Para o desenvolvimento do SadApresentacaoPOLI utilizou-se parte da API do J2EE versão 1.4, referente a Java Servlets, JavaServer Pages (JSP) e JavaMail como também o Struts versão 1.1.

4.1.2.1 Aplicação Web com Java na Montagem de Diretórios S@DPOLI

Segundo a especificação da plataforma J2EE, as aplicações web devem ser instaladas e executadas em web containers, os quais são os responsáveis diretos pelo gerenciamento do ciclo de vida dos componentes web da aplicação [19][24].

Para o projeto vigente o contexto é representado pelo diretório SadPoliWeb. Dentro do diretório contexto há possibilidade de se adicionar vários outros diretórios, no entanto, faz-se necessário agregar um especial denominado WEB-INF, com seus respectivos subdiretórios **classes** e **lib**. No WEB-INF também se encontra um arquivo de configuração chamado web.xml e conhecido na plataforma J2EE como *deployment descriptor* da aplicação web. O WEB-INF é um diretório de domínio privado, isto é, seu conteúdo não é acessível aos usuários da aplicação, por este fato é considerado o lugar ideal para agregar arquivos de configuração como o web.xml e até mesmo outros subdiretórios restritos à aplicação, caso necessário.

O *deployment descriptor* permite a configuração declarativa de uma aplicação web. Neste arquivo encontra-se a definição de segurança (autenticação e autorização), tratamento de erro, páginas de boas vindas, mapeamento das bibliotecas de tags adotadas para as páginas JSP, como também o tempo máximo de ociosidade (*timeout*) de sessão de usuários na aplicação, tudo isso apenas em um arquivo XML [23], sem a utilização de programação e eventuais compilações. A Figura 10 exibe a estrutura hierárquica de diretórios do SadApresetacaoPOLI atendendo a especificação J2EE para montagem de projetos web.

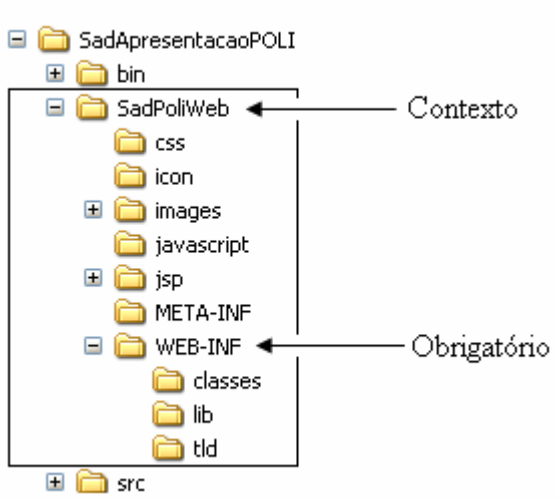


Figura 15. Estrutura de diretórios de projetos web.

Pelo fato de uma aplicação web possuir vários subdiretórios e arquivos, torna-se incômodo migrar a aplicação de um ambiente para outro, principalmente com relação ao tamanho do projeto em nível de bytes. Para simplificar este processo, os projetos web são empacotados em um único arquivo JAR, embora com a extensão **.war** (WAR-*Web Application Archive*) [19][24], tal extensão tem uma interpretação diferenciada aos Servlet Container. Por exemplo, ao exportar o SadPoliWeb.war para o diretório webapps do Tomcat, o Container automaticamente extrai todo o conteúdo deste arquivo, desempacota no diretório corrente com o mesmo nome do arquivo**.war**, sem a extensão, e o projeto estará pronto para ser acessado por meio da URL pré-estabelecida. Em tempo de desenvolvimento, tal URL por padrão refere-se ao http://localhost:8080/sua_aplicação. No caso em questão: http://localhost:8080/SadPoliWeb. Nota-se que atendendo a estruturação de diretórios pré-estabelecida aos projetos web a

implantação em um Container é relativamente simples. Para este fato, está sendo considerado a utilização básica dos recursos do Servlet Container.

O servlet container permite fazer configurações de portas, autenticação com um gerenciamento direto de conexão em um bando de dados, autorização de acesso a recurso, dentre outras características relacionadas aos projetos web. Na verdade, os Servlets estendem parte do comportamento do container e, por sua vez, o container monitora os servlets e seus comportamentos.

4.1.2. 2 Componentes Reutilizáveis

Uma aplicação web normalmente está associada a vários componentes de software. Os componentes são definidos como blocos de software que desempenham tarefas comuns a vários sistemas. São desenvolvidos para fornecer serviços, embora não sejam projetados para interagir diretamente com os usuários finais de um sistema [9]. A Figura 16 exibe as dependências de componentes associadas ao projeto SadPoliWeb.war implantado no servidor web.

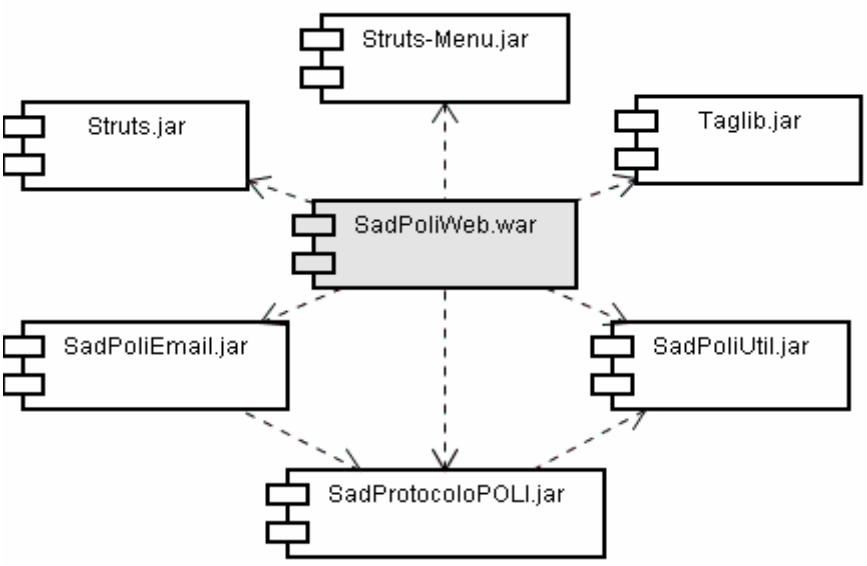


Figura 16. Diagrama de dependência de componentes da aplicação SadPoliWeb.war

- O Struts-Menu [29], refere-se a um componente de montagem dinâmica do menu com controle de acesso de usuários.
- Struts [28], refere-se ao framework para montagem da camada de apresentação.
- SadPoliEmail, componente desenvolvido para o controle de protocolos com o intuito de enviar emails para os envolvidos com a tramitação de documentos.
- SadProtocoloPoli, visto como um componente a oferecer serviços à aplicação.
- SadPoliUtil, componente desenvolvido para o controle de protocolos que disponibiliza formatações de campos, máscaras, etc.
- Taglib, referente as bibliotecas de tags da JSTL [38] e as oferecidas pelo Jakarta Struts [37].

A Tabela 5 descreve as versões dos componentes utilizados para manter a funcionalidade do sistema de controle de protocolos. Tratando-se de componentes, a identificação da versão utilizada é útil para notificar as limitações e o domínio de funcionalidades até então disponíveis para a versão em uso e servem como marco para atualizações futuras.

Tabela 5. Componentes utilizados pelo sistema de controle de protocolos.

Nome	Versão	Utilizado por
Struts	1.1	Camada de apresentação do Controle de Protocolos
Struts-Menu	1.3	Camada de apresentação do Controle de Protocolos
Taglib	1.3	Camada de apresentação do Controle de Protocolos
SadPoliUtil	1.0	Camada de apresentação e negócio do Controle de Protocolos
SadPoliEmail	1.0	Camada de apresentação e negócio do Controle de Protocolos
SadProtocoloPOLI	1.0	Camada de apresentação do Controle de Protocolos

4.1.2.3 Ambiente de Implantação

Adicionar um produto de software no meio físico requer um planejamento e conhecimento da infra-estrutura disponibilizada para apoiar o produto. O diagrama UML de implantação imprime uma visão do ambiente e do relacionamento entre o software e seu meio de operação [11][14]. O diagrama é um recurso válido para verificar e validar as cooperações entre os módulos envolvidos e representa uma maquete de implantação. Projetos distribuídos com instalações em mais de um terminal são bem mais representados e é necessária a utilização deste diagrama.

O projeto S@DPOLI adota uma abordagem cliente/servidor onde os recursos necessários para o funcionamento da aplicação ficam implantados apenas no lado do servidor. Deste modo, os terminais dos usuários precisam apenas ter acesso a rede Intranet da Escola e um navegador (browser) de sua preferência.

A Figura 17 ilustra o diagrama de implantação do sistema S@DPOLI no lado servidor. O módulo referente a componentes refere-se aos citados na Seção 4.1.2.2.

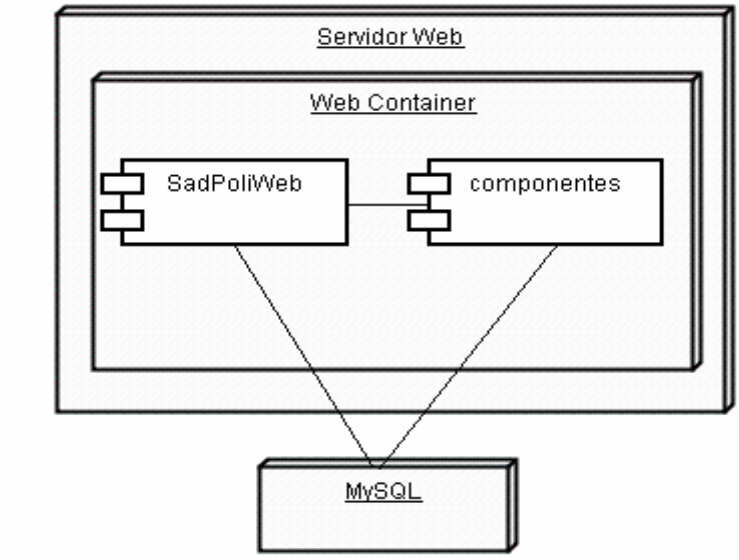


Figura 17. Diagrama UML de implantação.

O projeto SadApresentacaoPOLI apresenta uma estrutura montada pela arquitetura MVC do Struts e, por isso, não está sendo contemplado a representação das classes de ação que são constituídas basicamente de comandos acionados pelos componentes de visualização (HTML e JSP) para interagir com a camada de negócio da aplicação. Neste caso, os comandos apenas invocam as respectivas fachadas: Cadastro e Documento do SadProtocoloPOLI, em busca de incluir ou recuperar dados e informação.

4.1.3 Diagrama de Seqüência

O diagrama de seqüência é um tipo de diagrama usado em UML que deve ser utilizado em função dos casos de uso [9]. Cada diagrama de seqüência captura o comportamento de um único caso de uso. Seu objetivo é mostrar a interação entre os objetos ao longo do tempo, como também a seqüência de mensagens trocadas. Esse tipo de artefato contribui com a legibilidade da documentação técnica e antecipa para os engenheiros a estrutura do software, permitindo uma melhor aproximação com o código fonte necessário para a construção do produto.

Como exemplo ilustrativo está sendo aqui exposto o diagrama de seqüência relacionado ao caso de uso *Exibir Histórico do Documento.* Este caso de uso é exposto por meio do diagrama UML de seqüência, que representa uma visão reduzida do comportamento de um caso de uso no que diz respeito a execução de métodos implementados no software e ao seu tempo de operação [14]. A pesquisa do histórico é realizada por meio de duas possibilidades: número do protocolo e por situação do protocolo em uma data específica. Tem-se dois fluxos que convergem para a mesma atividade.

O *Exibir Histórico do Documento* permite aos usuários visualizar o fluxo do documento encaminhado e possibilidade de rastreamento das últimas operações realizadas, permitindo verificar se o documento foi recebido pelo departamento destinatário, se houve modificação na situação do documento ou mesmo se o documento fora reencaminhado.

Os conceitos básicos relacionados com o diagrama de seqüência são:

- *atores*: são entidades externas que interagem com o sistema oferecendo ou solicitando serviços, no exemplo exposto o ator é o funcionário.
- *Objetos*: representam instâncias das classes da arquitetura do sistema. No diagrama os objetos estão representados nos retângulos.
- *Mensagens:* são as informações trocadas entre objetos representados no diagrama por flechas.

A exibição ilustrada na Figura 18 refere-se ao fluxo da pesquisa por número do protocolo. Para maiores detalhes ver Apêndice A referente ao caso de uso *Exibir Histórico do Documento*.

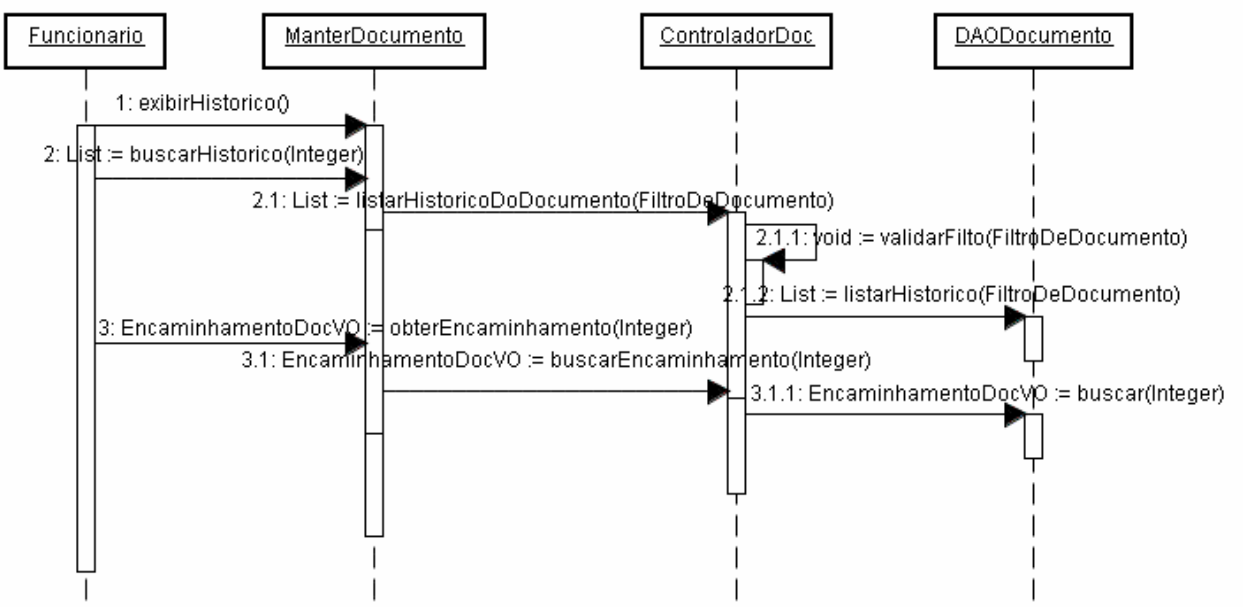


Figura 18. Diagrama de seqüência do caso de uso *Exibir Histórico do Documento.*

4.1.4 Testes

Projetos de teste são sempre válidos na construção de qualquer aplicação. Os testes verificam erros na semântica das funcionalidades e validam a lógica e as regras de negócio informando se estão sendo atendidas ou não, caso não estejam atendidas o problema provavelmente encontra-se relacionado a erros de estrutura (sintaxe) ou de lógica má formulada, isto é, não conforme com o esperado.

Quanto maior for uma aplicação, em nível de código, mais válidos serão os testes para manter o controle do que está sendo desenvolvido e as falhas poderão ser depuradas na própria fase de construção, evitando a identificação de defeitos de software em fases posteriores, como por exemplo, na fase da liberação (“release”) do produto ao cliente. Nesta fase o produto deve se apresentar numa versão estável, sem falhas de codificação (“bugs”).

Projetos de testes consomem grande parte de tempo e esforço com relação à própria construção do software e em muitos casos este tempo não é estimado de forma precisa pela gerência e engenheiros envolvidos. Os testes também encarecem o produto final.

A montagem de um plano de teste para uma aplicação não é considerada como um produto, mas sim um serviço que tem por objetivo maior controle na qualidade do produto de software a ser desenvolvido.

A busca pela qualidade do software não está exclusivamente relacionada aos testes, há outros itens que agregam valor como é o caso da inspeção de código com a finalidade de verificar o padrão de codificação pré-estabelecido ao projeto; definir comentários em todas as funcionalidades (Javadoc) facilitando o entendimento e a manutenção da estrutura programada; atualizar a documentação sempre que mudanças ocorrem na estrutura implementada, dentre outros.

O teste de unidade é uma atividade desempenhada pelos desenvolvedores de software. O software normalmente é subdividido em casos de uso e cada desenvolvedor constrói o teste de unidade para os casos de uso correspondentes ao seu trabalho. Na orientação a objetos (OO) uma unidade de código é geralmente um método de uma classe [9].

A existência de testes de unidade, entre outros benefícios, aumenta a confiança dos desenvolvedores no momento de realizar mudanças no sistema, facilitando a documentação e comunicação entre os membros da equipe.

Existem três abordagens de testes de unidade [48]:

- *Teste de lógica:* testam as classes isoladas do ambiente e das demais classes com que se relacionam. Facilitam a identificação da origem dos problemas. Na prática, deve ser escrita uma classe de teste de lógica para cada classe a ser testada.
- *Teste de integração:* testa a integração entre várias classes ou entre determinada classe e o ambiente em que é executada, por exemplo, teste de integração entre a camada de persistência de uma aplicação com o banco de dados.
- *Teste funcional:* testa o resultado da execução de determinada classe, do ponto de vista do usuário.

A escrita dos testes deve começar pelas unidades e situações mais importantes, já que é economicamente inviável testar todas as situações possíveis. Existem classes e métodos que não precisam ser testados, como é o caso dos métodos *setter* e *getter* de componentes JavaBeans, pelo fato de não estarem atrelados com regras de negócio.

A ferramenta de testes de unidade adotada para o projeto é o JUnit [48]. Esse framework é o padrão de fato para testes de unidade em Java e seus objetivos principais estão associados com:

- Facilitar a escrita dos testes, exigindo o mínimo de conhecimento para codificá-los.
- Permitir que o código necessário para configurar o teste seja reutilizado. Evitando que cada teste não repita o mesmo código de inicialização. Deste modo torna-se possível isolá-lo em método que é executado antes de cada teste.
- Permitir que os testes sejam usados como testes de regressão, facilitando sua manutenção para que possam ser executados repetidas vezes, possivelmente em outros ambientes e por outras pessoas, em conjunto com outros testes.

O projeto de teste do S@DPOLI foi constituído de testes de unidade em nível da fachada da aplicação do SadProtocoloPOLI, onde está todo o entendimento do negócio implementado. Os testes estão relacionados com a verificação de possíveis violações das regras estabelecidas como também com a verificação das funcionalidades isoladas, para confirmar se estão operando corretamente. O projeto SadApresentacaoPOLI, por não definir regras de negócio e apenas estar vinculado com a exibição dos dados e informação, não necessitou do teste de unidade, pois tal projeto não representa um elemento crítico ao domínio do problema.

No Apêndice B está um exemplo de classe de teste de unidade utilizando o framework JUnit.

4. 2 Representação Gráfica do Sistema S@DPOLI

A elaboração da interface gráfica de sistema (GUI) [32][33][44] é um processo bastante trabalhoso e de grande importância para os usuários finais. Na tentativa de atender um requisito imposto de fácil navegabilidade, o projeto de interface foi elaborado basicamente em três passos: consultar, detalhar e confirmar a ação. Cada uma destas atividades é representada em uma própria tela definida, oferecendo deste modo, uma seqüência lógica na operação .

Nesta seção serão exibidas apenas as telas principais referentes ao controle de protocolos acompanhadas de uma breve explicação quanto ao seu conteúdo.

- **Entrar e Encaminhar Documento**

A Figura 19 refere-se ao processo manuscrito de cadastrar um protocolo referente ao documento a ser encaminhado. O que está sendo registrado é:

1. Requisitante, referente ao responsável pelo documento a ser encaminhado.
2. Tipo de Documento, especificando a qual categoria de documento se classifica o encaminhamento.
3. Anexos, referente às possibilidades em que um documento também esteja atrelado a outros e que necessitam ser protocolado.
4. Observação, permitindo ao remetente descrever algo de valor referente ao encaminhamento.
5. Origem, descreve o operador (funcionário) e departamento do qual o encaminhamento foi originado.
6. Destino, descreve a qual funcionário se refere o protocolo e de qual departamento.

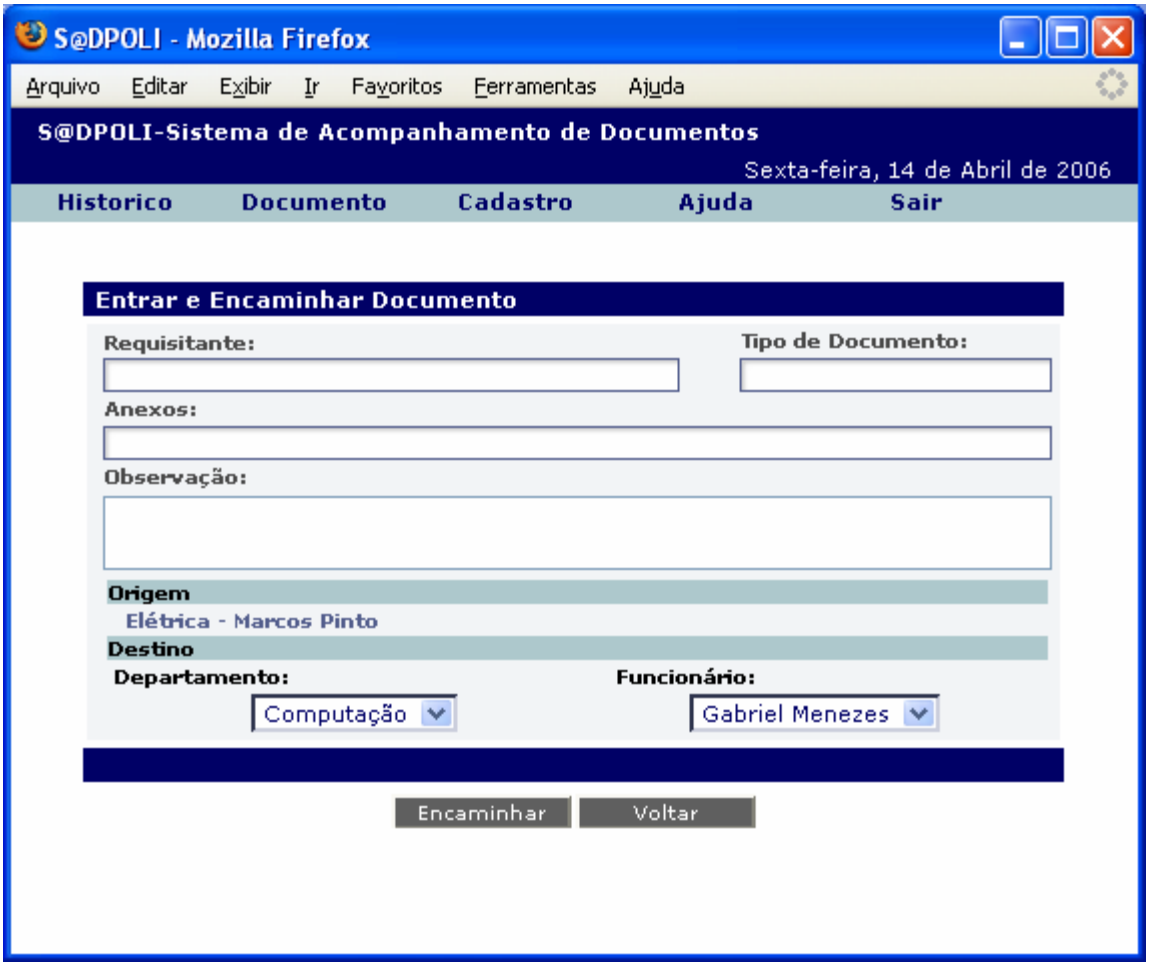


Figura 19 . Entrar e Encaminhar Documento.

- **Confirmar Recebimento**

A Figura 20 refere-se ao recebimento de documentos no departamento destinatário. Um documento encaminhado só poderá ser manipulado por parte do funcionário destinatário caso confirme o recebimento de documento. No primeiro momento será exibida uma lista de documentos a serem recebidos para, em seguida, confirmar de fato o recebimento exibido nas duas telas seguintes.

Na listagem, de pendências a ser recebido, está sendo exibido:

1. O Número do protocolo, referente ao identificador único documento encaminhado.
2. O tipo de Documento, especificando a qual categoria de documento se classifica.
3. O código de situação de documento podendo ser para esta ação A-Andamento, D-Deferido, I-Indeferido e V-Arquivado.
4. Requisitante, referente ao responsável pelo documento a ser encaminhado.
5. Data de Envio, referente a data em que o documento fora encaminhado.

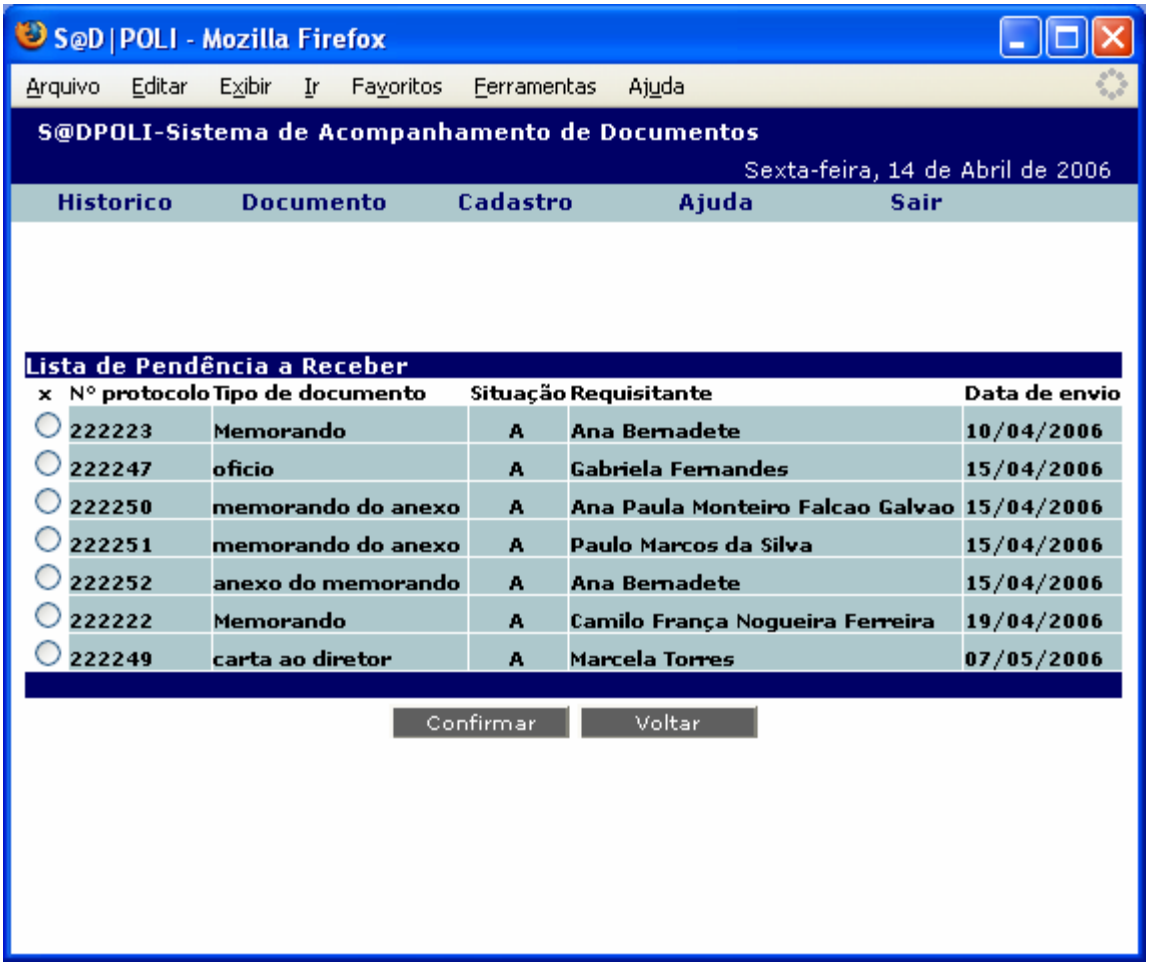


Figura 20. Lista de pendência a receber.

Na Figura 21, tela referente a confirmação do recebimento, está sendo exposto:

1. Requisitante, exibe o responsável pelo documento a ser encaminhado.
2. Número do Protocolo, exibe o identificador único documento encaminhado.
3. Tipo, especificando a qual categoria de documento se classifica.
4. Anexos, caso o documento encaminhado apresente outros documentos auxiliares.
5. Remetente, exibe o nome do funcionário e o seu respectivo departamento, refere-se a quem originou o encaminhamento.
6. Data de envio, exibe a data em que o documento fora encaminhado.
7. Destinatário, exibe o nome do funcionário e seu respectivo departamento, refere-se a quem deve receber o documento.
8. Observação, caso o encaminhamento apresente uma nota descritiva do que se refere ou grau de urgência.
9. Data de confirmação, referente a data em que o destinatário inicia a atividade com o documento.
10. Nova Situação, possibilitando ao destinatário expor uma nova situação ao documento na hora do recebimento.
11. Nota, permitindo ao destinatário acrescentar mais informações ao documento manipulado.

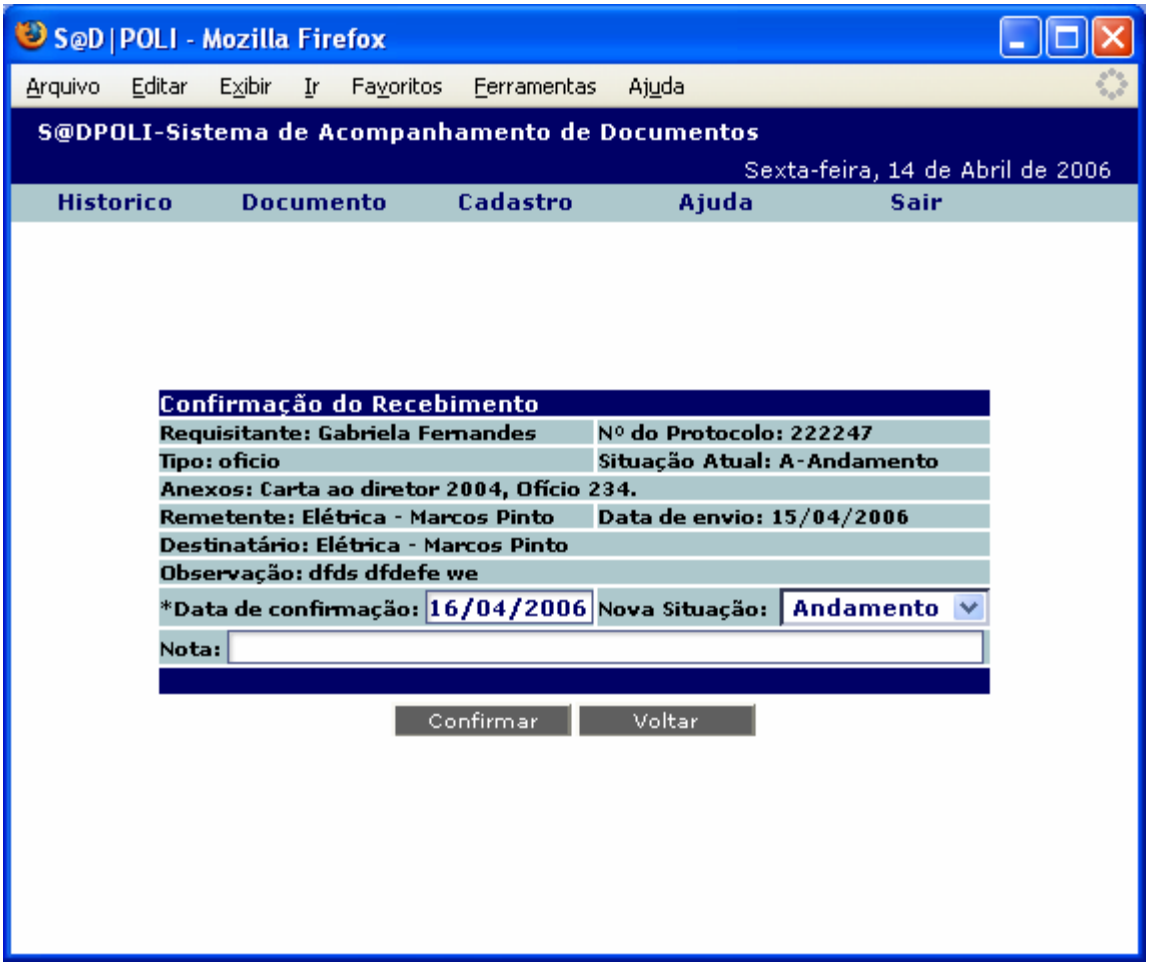


Figura 21. Confirmação do Recebimento.

- **Modificar a situação do documento**

Está relacionado com a possibilidade do departamento destinatário, modificar a situação do documento, tendo como pré-condição o seu recebimento prévio, expondo ao(s) interessados um parecer do andamento do documento.

Os campos exibidos são muito semelhantes aos definidos na Figura 21 e as ações referem-se à alteração da situação do documento, possibilitando ao documento passar para uma das seguintes possibilidades: deferido, indeferido ou suspenso, conforme pode ser visualizado na Figura 22.

Figura 22. Modificar a situação do documento.

- **Reencaminhar Documento**

A atividade é semelhante ao encaminhar documento, apenas neste caso o protocolo já existe e fora recebido pelo departamento. Neste caso, deve-se apresentar uma situação válida (Andamento, Deferido, Indeferido) para o reencaminhamento. Os únicos campos editáveis estão

relacionados a uma observação opcional para o reencaminhamento, como também o novo departamento e funcionário de destino, como exposto na Figura 23.

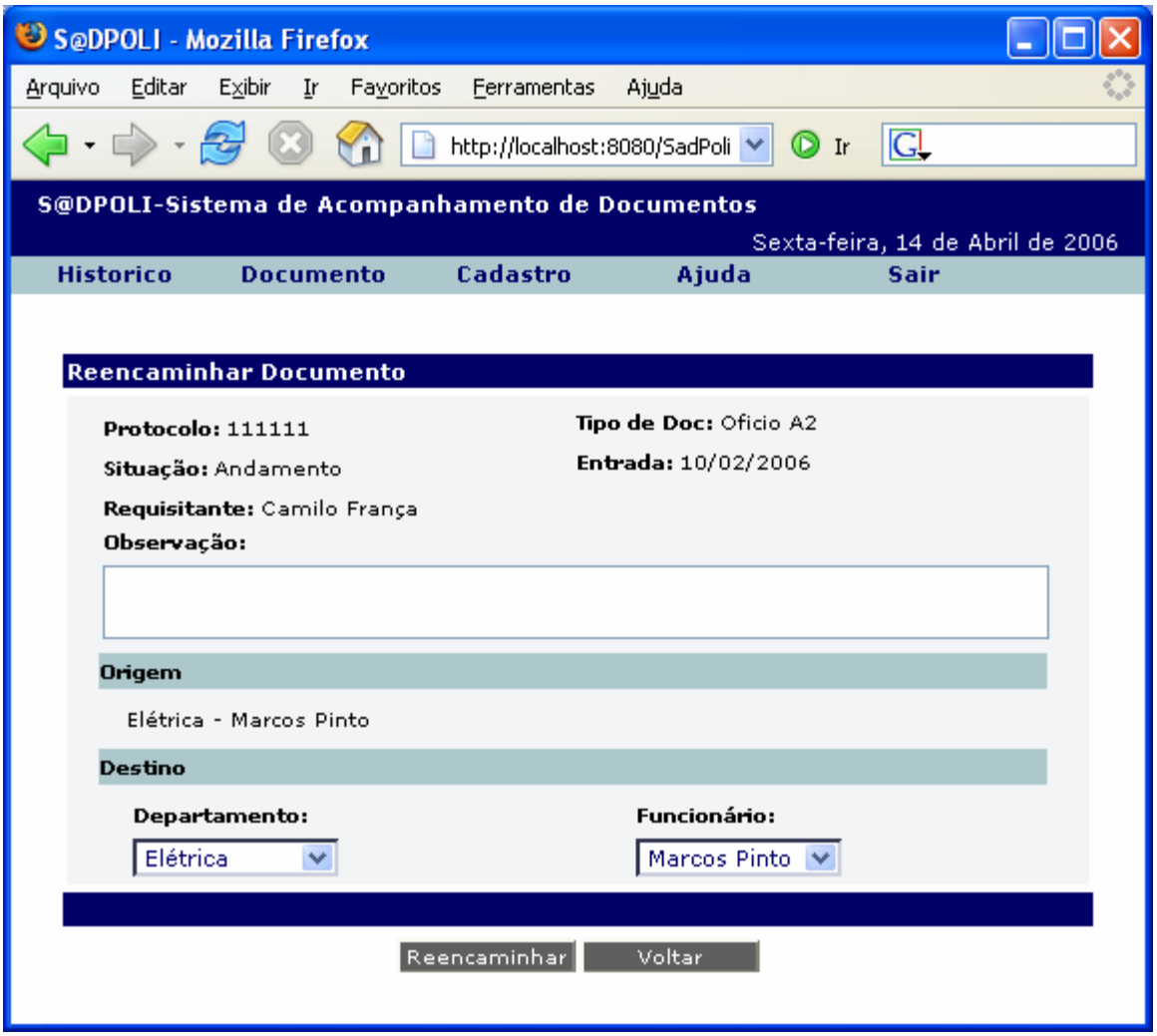


Figura 23. Reencaminhar documento.

- **Histórico do Documento**

Permite aos funcionários rastrear o andamento entre departamentos e a mudança de estado dos documentos de seu interesse. A busca do documento é feita pelo número do protocolo ou pela situação do documento em um determinado mês e ano. Vide Figuras 24 e 25.

Figura 24. Consultar histórico do documento.

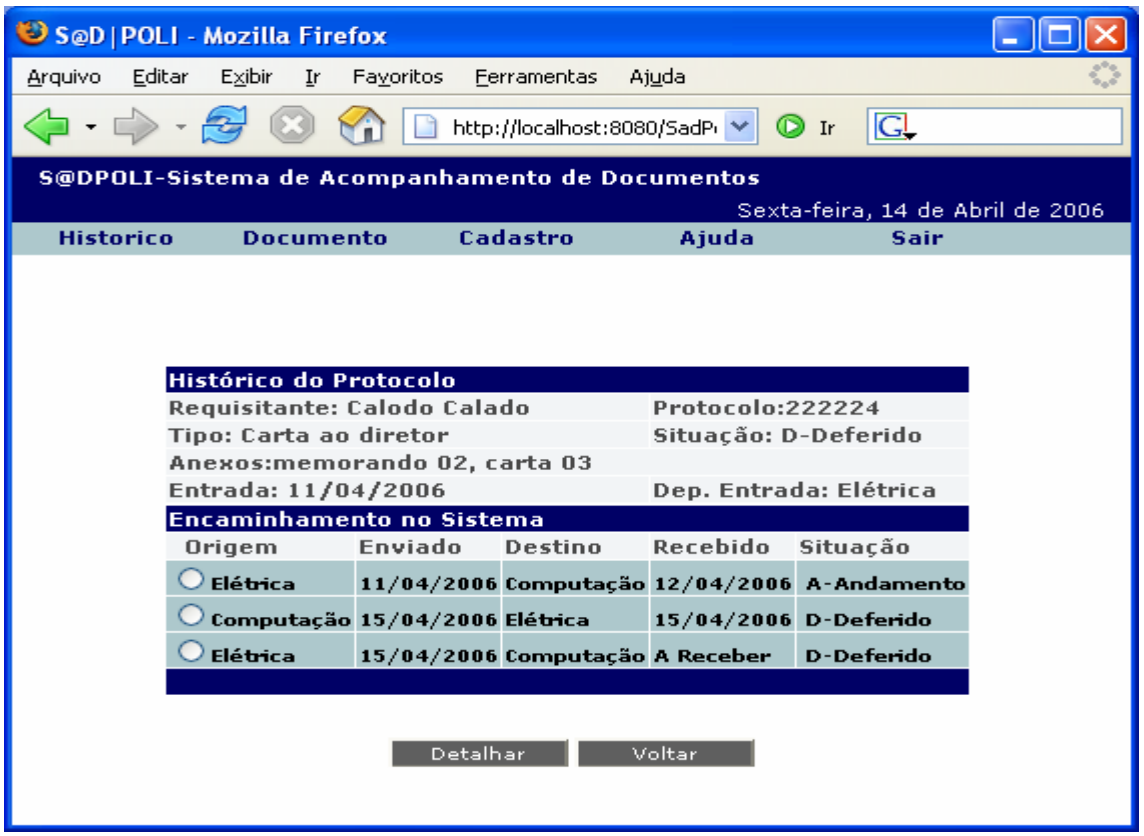


Figura 25. Exibir histórico do documento.

Capítulo 5

Lições Aprendidas

Neste capítulo serão abordadas as experiências, conflitos, riscos ocorridos antes do projeto e ao longo dele. O propósito é relatar e contribuir a quem possa interessar todo o processo vivenciado.

5.1 Levantamento de Requisitos

Como relatado no Capítulo 2, o levantamento de requisitos é um processo difícil e que exige um comprometimento de todos interessados com o produto final.

Ao se receber uma proposta de desenvolvimento de sistemas é importante verificar qual é o nível de domínio do problema por parte do cliente, pois quando o mesmo não sabe expor o que deseja, a elaboração do sistema torna-se mais trabalhosa e pode-se estimar tempo, preço e esforço de forma errada e ser penalizado durante todo o projeto [8][9]. Estas estimativas devem ser feitas bem no início de um contrato.

O desenvolvimento de software e sistemas raramente atende ao gosto e necessidade do cliente sem a sua participação direta, com sugestões, críticas e validações. Esta atividade foi iniciada no período correto para poder seguir com os demais ciclos do modelo espiral [9][10], num tempo suficiente para conclusão deste trabalho. O interesse por parte do cliente foi o fator positivo para o seu desenvolvimento, os softwares e sistemas são construídos para atender ao menos a necessidade de alguém, pois há custos envolvidos, quer de tempo quer financeiros.

Ser um analista de sistemas e passar pela fase de requisitos é um grande desafio. Na prática, ele precisa entender a necessidade do cliente, delimitando o problema e interpretando as informações fornecidas de modo a trazê-las para a realidade digital por meio do “mapeamento da visão do cliente”. A forma mais utilizada para se mapear a visão do cliente é o desenvolvimento de casos de uso. Não se pode estimar o grau de dificuldade do desenvolvimento de um sistema simplesmente pelo fato de ele ser tradicional e aparentemente sem grandes complexidades. O que vai determinar a facilidade ou dificuldade de construção é o quanto podemos contar com os interessados pelo produto. Trabalhar na elaboração seguindo as boas práticas da Engenharia de Software (tanto na organização quanto definição das atividades) sem o envolvimento de todos, acaba em algum momento desestimulando um ou mais membros da equipe gerando, no mínimo, atrasos. Tal dificuldade ocorreu na definição do presente projeto, bem na fase inicial foi percebido o quanto o analista e gerente do projeto estaria, praticamente só nesta jornada. O esforço em marcar reuniões, agendá-las e fazer daquela etapa do processo uma tarefa disciplinada foram exemplos desta fase difícil.

Deve-se levar em consideração que apenas uma minoria dos clientes de produtos de software tem noções do processo de engenharia, de como se comportar com tal atividade ou mesmo de como poder ajudar. Existem vários tipos de consumidores no mercado e dificilmente encontraremos aqueles que sabem de fato o que querem ao ponto de oferecer, sem dificuldades,

artefatos de entrada, regras claras, restrições de ambiente e principalmente que estejam prontos a negociar de forma consciente.

A coleta de dados foi feita totalmente no departamento DESAF com a responsável do setor e a estagiária. De início informaram, de forma transparente, que não entendiam de sistemas computacionais, de layout gráfico, e que sentiam a necessidade de ter um gerenciamento de documentos via computador com rápida assimilação por parte dos usuários. A partir deste fato, houve um grande esforço em definir uma estrutura gráfica para representar as informações de forma a não penalizar tanto o usuário quanto o “web design”.

Nem todo projeto de software e sistemas precisa passar pelas mesmas atividades da engenharia de requisitos para convergir à solução efetiva. Como exemplo, a utilização da prototipagem é útil e recomendada quando os requisitos estão confusos e a representação das entradas e saídas não estão definidas de forma clara [9]. A prototipação teve grande importância neste projeto até certo ponto. Inicialmente, foi feito um protótipo de interface gráfica, contudo o cliente desejava visualizar o fluxo em operação, portanto foi necessário tornar o protótipo funcional e as rodadas de verificação foram realizadas com um produto que já tinha acesso ao banco de dados.

A elaboração do protótipo seguiu a abordagem de "produto rápido", estágio de desenvolvimento no qual não há grandes preocupações com desempenho dos algoritmos, estruturação e restrições ao sistema [9]. Com o protótipo foi possível alcançar um nível de aceitação do sistema, por parte do cliente. Após três ciclos de apresentação do produto e do resgate dos demais requisitos, o software que constitui o protótipo fora abandonado por questão de qualidade.

A grande vantagem da utilização da técnica de prototipagem foi ter aproximado o cliente ao "produto" possibilitando coletar e validar requisitos de forma gráfica e ilustrativa. A adição de novos requisitos e correção de antigos deixava-o mais próximo do funcionamento, contudo o protótipo, se distanciava, cada vez mais, de um produto controlável e de qualidade. Foi difícil descartar este produto e partir para um projeto novo e com ênfase na qualidade porque a nova versão exigia mais tempo e esforço para disponibilizar um produto documentado e estável, como acordado com o cliente.

O desenvolvimento de software e sistemas web requer um processo de construção estruturado e planejado para possibilitar seu sucesso como produto final. A equipe de desenvolvimento precisa estar acessível e disponível ao diálogo durante toda as fases ou ciclos de iterações. Um modelo prescritivo é recomendado e utilizado para colocar ordem nos casos de desenvolvimentos de software [9]. No desenvolvimento deste projeto adotou-se o modelo prescritivo espiral, o qual foi abordado no Capítulo 2.

5.2 Retrabalhos

Uma atividade desgastante no desenvolvimento de um sistema diz respeito aos retrabalhos. Na construção do presente sistema, os retrabalhos ocorreram devido a falta de definição em alguns pontos e ausência de regras que possibilitaram a má interpretação por parte do analista. Por exemplo, realizaram-se várias alterações no protótipo funcional na tentativa de conseguir atingir o estimado pelo cliente. Percebe-se que as informações foram coletadas a partir do esforço em desenvolver um produto, instalar no DSC, agendar uma data de exibição, coletar as novidades e partir para a próxima rodada. Este processo foi encerrado no final de janeiro/2006.

5.3 Sistema com Estrutura Cliente/Servidor

O desenvolvimento de sistemas baseado no conceito de cliente/servidor [6][7], além de ser uma abordagem muito utilizada nas empresas e instituições, tem vantagens na distribuição do acesso entre os usuários. Ao se desenvolver um sistema é fundamental saber bem no começo da sua definição a quais usuários o projeto pretende atender e quais os recursos oferecidos para sua implantação a empresa/instituição disponibiliza. A arquitetura de um projeto desenvolvido para Web, Intranet ou outro tipo de rede, depende justamente destes fatores. O fato de o sistema de controle de protocolos da Poli utilizar os serviços Intranet da Escola, cujo domínio de usuários é limitado e exige autenticação, faz com que o sistema seja mais seguro comparado aos expostos a rede Internet. Projetos com estrutura cliente/servidor que envolvem redes, protocolos, segurança e acesso a banco de dados exigem grande esforço em configuração apenas no lado do servidor. A utilização de pequenas redes em escolas, empresas, instituições, com esta estrutura, é crescente devido os custos serem mais acessíveis nestes serviços.

5.4 Interface Gráfica

A elaboração de formulários HTML é basicamente definida em tabelas [35], a partir destas tabelas é possível definir layout diversificado para a exibição da informação. Durante o desenvolvimento do protótipo, a codificação de páginas HTML seguiu uma abordagem involuntária baseada por pixel de tela, ou seja, para posicionar uma tabela com 35% de distância do lado esquerdo do monitor e 50% com relação ao topo, estavam sendo ajustados os parâmetros do HTML em nível de pixel, como exemplo, align=“35px” e top=“50px”. No entanto, nesta abordagem passou despercebido o fato de que as máquinas dos usuários podem ter resolução diferente da utilizada na elaboração das telas. A comprovação deste fato foi notificado nas fases em que o protótipo fora implantado na Poli. Como em nenhum momento tinha sido imposto por parte do cliente qual seria o padrão de resolução dos navegadores (browser) utilizado na Escola, seguiu-se uma nova estratégia de estruturação de telas com o objetivo de independência tanto no tamanho de tabelas quanto na resolução, isto é, podendo ser 600 x 800 pixels, 1024 x 768 pixels ou superior.

Atualmente, as páginas HTML convertidas em JSP estão todas definidas em posição centralizada e sem comprimento fixo, e como conseqüência, elas vão sendo ajustadas de acordo com a informação recuperada do banco de dados. Esta abordagem permitiu que o comportamento visual ficasse mais homogêneo tanto com relação a resolução quanto à exibição da informação.

5.5 Criação do Protótipo

Em busca de concentrar os esforços e comprovar o entendimento descrito verbalmente nas reuniões, foi elaborado um documento contendo perguntas, em busca de solucionar dúvidas de negócio com relação ao fluxo de protocolos, e encaminhado aos interessados. O retorno foi : “*colocar o sistema em operação e a partir de então validaremos a sua corretude”*. Esta resposta, além de ferir o processo de Engenharia de Software também se mostrava inviável devido ao fato de no momento não haver nada definido quanto à representação da informação, navegabilidade,

restrições e papéis dos usuários. Para contornar a situação foi necessário desenvolver um protótipo gráfico inicial do sistema que possibilitou uma nova aproximação com o departamento DESAF. O protótipo foi implantado numa máquina do DSC - Departamento de Sistemas Computacionais e foi encaminhado a URL de acesso tanto para a Diretoria quanto para o departamento DESAF. A necessidade de implantação do protótipo em um servidor se deu por dois fatores:

5.6 Frameworks

Os frameworks são produtos de software com a finalidade de customizar a linha de produção de sistemas que necessitam agilidade em construção seguindo padrões que permitem ganho de produção em escala.

A escolha e utilização de frameworks populares pode contribuir bastante com uma etapa futura e natural no ciclo de vida dos software que é a manutenção, quer seja preventiva, corretiva ou evolutiva. É mais fácil aprender ou já ter conhecimento de uma estrutura padronizada a ter que aprender uma arquitetura fechada e definida para uma aplicação em particular. Este ponto de vista está baseado no fato de que os engenheiros que participam da fase de manutenção normalmente não são os mesmos que desenvolveram o produto. A manutenção é um momento delicado onde é preciso verificar se o que foi alterado em prol da correção ou evolução não interferirá com o que já existe em forma estável. Ao ser escalonado para um determinado tipo de manutenção de um sistema o engenheiro deverá possuir o conhecimento dos frameworks adotados. Quanto mais popular o framework mais chances se tem de encontrar profissionais aptos, com rápida assimilação do ambiente arquitetural.

O aprendizado dos dois frameworks adotados para o desenvolvimento do sistema S@DPOLI, Struts e Hibernate, se deu durante a trajetória de construção do projeto. O Struts é relativamente fácil de compreender e utilizar desde que haja um boa noção de rede cliente/servidor, protocolo HTTP e o conceito e uso de servlets e JSP. O Hibernate é um framework com uma abordagem ORM que abstrai o desenvolvedor da camada de persistência, ou seja, do contexto entidade-relacional.

Ao trabalhar com o Hibernate pela primeira vez, tendo como conhecimento prévio a API JDBC, percebe-se uma grande diferença. As tabelas relacionais ficam todas mapeadas em um arquivo XML a partir do qual todas as operações de acesso ao banco se dá pela invocação de objetos persistentes. Mudanças no estado de um objeto afetam diretamente ao banco de dados de forma transparente sem necessidade de criar novas HQL para alterar um atributo de um objeto, ou melhor de uma tabela. Já na utilização da API JDBC, para cada necessidade de alteração em um campo distinto de uma tupla (linha) da tabela, em momentos diferentes na execução do sistema, é necessário criar novos métodos com SQL próprias para a alteração do campo.

O Hibernate é econômico com relação às funcionalidades de persistência, isto é, ao resgatar um objeto do banco de dados o mesmo é considerado em estado persistente e neste estado qualquer alteração no objeto irá afetar também o banco. É importante relatar que este fato só ocorre enquanto a sessão ou conexão com o banco de dados ainda permaneça ativa. Diante deste fato, adotou-se uma abordagem de abrir e fechar as conexões no nível da fachada da aplicação, permitindo que os controladores, da camada de negócio, pudessem fazer consultas a objetos para verificação e validação, para alteração em um ou mais objetos que permanecem no estado persistente com apenas uma única abertura de conexão, oferecendo para o sistema um

ganho em desempenho, uma vez que o processo de abrir novas conexões oferecem um custo para as aplicações.

5.7 Testes de Aplicações

Os testes têm vários propósitos na construção de software e normalmente estão vinculados com o rastreamento de erros para posterior correções. É recomendado, segundo as boas práticas da Engenharia de Software, que não sejam aplicados apenas no fim da elaboração do projeto, mas durante toda a fase de construção [9][49][50]. Percebe-se então que ao definir um prazo de elaboração de um sistema, é importante alocar uma boa fatia de tempo para esta atividade.

Os projetos web estão atrelados a vários componentes e elementos de infra-estrutura como redes, navegadores, diferentes sistemas operacionais, e por estes fatores há uma tendência a um trabalho prolongado na atividade de teste.

Os testes em muitos casos são feitos de forma rápida, atendendo parte do domínio do sistema, a rapidez pode ser pela necessidade de liberar o produto ao cliente ou por não estar incluso no orçamento. [9][50]. Um teste mais detalhado faz ao menos as seguintes verificações e validações:

1. **Conteúdo**: o objetivo é tanto avaliar elementos sintáticos da linguagem, ortografia, pontuação como a semântica dos documentos, verificando consistência do texto e ausência de ambigüidades.
2. **Interface**: verificam erros de não conformidade com os requisitos do cliente, como exemplo a forma de representação da informação.
3. **Navegação**: verifica o fluxo de execução do sistema, em busca de falhas nos links.
4. **Componentes**: verifica o comportamento funcional dos componentes “encaixados” ao sistema.
5. **Configuração**: verifica o comportamento do sistema tanto no lado do servidor quando do cliente.
6. **Desempenho**: criam-se várias situações adversas de operação do sistema em busca de verificar a carga de processamento.
7. **Segurança**: busca verificar possíveis vulnerabilidades e caso a invasão tenha uma possibilidade de ocorrência, tem-se uma falha de segurança a ser corrigida.

Os testes adotados para o sistema S@DPOLI estão constituídos dos testes unitários [48], referente a codificação do sistema onde se verificou as pré-condições e pós-condições das funcionalidades implementadas; testes de interface e navegabilidade[9][49] feito pelo analista em acordo com o cliente; teste de configuração [9]; testes de verificação de casos de uso [9][50], realizado no final do projeto.

O teste de segurança pode levar a altos níveis de atividades [9][50]. No caso do sistema de controle de protocolos está sendo adotado o padrão J2EE de segurança declarativa [24] para o processo de autenticação e autorização de usuários no uso dos serviços. Com relação aos testes de domínio do negócio está sendo atribuída a responsabilidade para o cliente analisar, pois é a pessoa mais adequada devido entender melhor os conceitos de funcionalidade e limitação das atividades no contexto prático do dia-a-dia.

Capítulo 6

Conclusões

Este projeto teve o objetivo de desenvolver um sistema de informação de controle de protocolos da área administrativa da Escola Politécnica de Pernambuco com o intuito de oferecer uma nova abordagem de gerenciamento e envio de documentos entre seus departamentos, por meio do estudo e aplicação das regras e recomendações da Engenharia de Software durante o seu desenvolvimento, passando pelas etapas de planejamento, especificação, implementação até o teste do sistema, em acordo com a expectativa do cliente.

O desenvolvimento de software tendo como ferramenta de trabalho apenas a API Java, como no caso da API para projetos Web, possibilita ao desenvolvedor uma maior liberdade de estruturação e montagem de uma arquitetura própria e personalizada ao projeto, porém requer maior tempo para definição do arcabouço do software e validação da estrutura, tendo em consideração que uma API representa bibliotecas de funcionalidades a serem invocadas em uma estrutura previamente montada pelo desenvolvedor ou adquirida por meio de frameworks. Por este motivo, além da utilização das APIs foram utilizados componentes já validados para ganho de tempo durante a elaboração do sistema.

O sistema S@DPOLI foi desenvolvido seguindo uma abordagem de linha de montagem de software utilizando o framework Struts como arcabouço da camada de apresentação. Este framework possibilitou, por meio de pontos de extensão, a adição de novos componentes. Para a camada de persistência foi adotado o framework Hibernate que disponibiliza comandos como de inclusão, exclusão, atualização pré-compiladas e de imediato manuseio, diminuindo deste modo a implementação destes comandos por parte do desenvolvedor da camada de persistência. O Hibernate oferece um arcabouço orientado a objetos e abstrai o desenvolvedor da atividade mecânica de acesso ao banco de dados. Além destes dois frameworks utilizados na montagem da estrutura do sistema, outros componentes contribuíram no aperfeiçoamento dos detalhes.

A dificuldade por parte da responsável pelo fornecimento dos dados em descrever o que desejava de forma clara, tornou a etapa de levantamento de requisitos demorada, levando o analista a oferecer ao cliente, por meio de protótipo, alternativas para serem julgadas até a obtenção de um escopo satisfatório. Contudo, o escopo adotado não foi suficiente para evitar retrabalhos.

Concluindo, pode-se afirmar que apesar das dificuldades iniciais encontradas durante o levantamento de requisitos, o sistema de informação da Escola Politécnica de Pernambuco foi desenvolvido com a capacidade de atender o fluxo primário de controle de protocolo para o qual se propôs o presente projeto, permitindo o gerenciamento e controle de envio e recebimento de documentos entre seus departamentos, oferecendo ao usuário a oportunidade de conhecer a

localização, o estágio de apreciação ou o estado final dos mesmos. Também foram descritas, de forma resumida e não extensiva, as principais etapas do desenvolvimento do sistema de controle de protocolo, buscando uma abordagem técnica que contribua para a compreensão da construção de projetos Web.

6.1 Contribuições

O presente trabalho contribui para o melhoramento do gerenciamento dos documentos que trafegam entre os departamentos da Escola Politécnica de Pernambuco, resultando em maior agilidade na sua localização e no conhecimento do seu estágio de apreciação ou do seu estado final, além de permitir o cadastro de funcionários e dos departamentos que utilizam o sistema, oferecendo maior segurança no controle de tais documentos e minimizando perdas. Também contribui com uma abordagem técnica a respeito do desenvolvimento de um sistema Web.

6.2 Trabalhos Futuros

O sistema de controle de protocolos pode ser ampliado. Sugere-se a adição de novos elementos que permitam a geração de relatórios e o controle de outros tipos de documentos que ultrapassem a esfera administrativa, atendendo também as necessidades dos alunos, conseqüentemente com aumento do domínio de acesso ao usuário.

Bibliografia

[1] ELGIN, D. A. Dinâmica da Evolução Humana. ed. Cultrix 1999.
[2] STAIR, M. RALPH, Princípios de Sistemas de Informação. ed. LTC 2002.
[3] O' BRIEN, A. JAMES, Sistema de Informação e as Decisões Gerenciais da era da Internet. ed. Saraiva 2003.
[4] LAUDON, C. KENNETH; LAUDON, P. JANE, Sistemas de Informação. ed. LTC 1999.
[5] W3C. Disponível em <http://www.w3.org>. Visitado pela última vez em 24/06/2006.
[6] KUROSE, F. JAMES; ROSS, W. KEITH, Redes de Computadores e a Internet- Uma Nova Abordagem. ed. Pearson Brasil 2005.
[7] TANENBAUM, A. S. Redes de Computadores. ed CAMPUS 2003.
[8] REZENDE, D. A. Engenharia de Software e Sistema de Informação. ed. Brasport 2005.
[9] PRESSMAN, S. ROGER, Engenharia de Software. ed McGraw-Hill 2006.
[10] SOMMERVILLE, I. Engenharia de Software, ed. Prentice- Hall 2003.
[11] ERNANI, M. Desenvolvendo Software com UML 2.0 – Definitivo. ed. Pearson Brasil 2004.
[12] BOOCH, G; RUMBAUGH J; JACOBSON IVAR, UML Guia do Usuário. ed Campus 2000.
[13] ALISTAIR, C. Escrevendo Casos de uso Eficazes - Um Guia Prático para Desenvolvedores de Software. ed. Bookman 2005.
[14] BOOCK, G; RUMBAUGH J; JACOBSON IVAN, UML Guia do Usuário. ed. Campus 2000.
[15] SCHMULLER, J. Sam's Teach Yourself UML in 24 Hours, ed. Sams 2004.
[16] GALITZ, O. WILBERT, The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techiques. ed. Wiley Publishing, Inc. 2002.
[17] ARRINGTON C. T.; RAYHAN S. H. Enterprise Java with UML. ed. Wiley Publishing, Inc. 2003.
[18] BROEMMER D. J2EE Best Practices Java Design Patterns, Automation, and Performance. ed. Wiley Publishing, Inc. 2003.
[19] BOND, Martin, et al.Aprenda J2EE em 21 dias. ed Pearson Education 2003.
[20] METSKER, J. STEVEN, Padrões de Projetos em Java. ed. Bookman 2004.
[21] CRAIG, L. Utilizando UML e Padrões - Uma Introdução a Análise e aos projetos orientados a objetos. ed. Bookman 2004.
[22] CRUPI, J; MALKS, D; ALUR, D. Core J2EE Patterns. ed. Campus 2004.
[23] WEISS, D; GABRICK, K. A. J2EE and XML Development. ed. MANNING 2002.
[24] BODOFF, S. Tutorial do J2EE-Enterprese Edition 1.4. ed. Ciência Moderna 2005.
[25] J2SE 1.4 . Disponível em: <http://java.sun.com/j2se/1.4.2/download.html>. Visitado pela última vez em 24/06/2006.
[26] BAUER, C.; KING, G. Struts in Action. ed. MANNING 2003.
[27] CAVANESS, C. Programming Jakarta Struts. ed. Oreilly 2004.

[28] Framework Struts. Disponível em: <http://struts.apache.org/>. Visitado pela última vez em 15/04/2006.
[29] Componente Struts Menu. Disponível em: <http://sourceforge.net/projects/struts-menu/>. Visitado pela última vez em 14/04/2006.
[30] Framework Hibernate. Disponível em <http://www.hibernate.org> . Visitado pela última vez em 10/04/2006.
[31] BAUER, C,; KING, G. Hibernate in Action. ed. MANNING 2005.
[32] DEITEL, H. M.; DEITEL, P. J. Java Como Programar. ed. Campus 2003.
[33] OLIVER, D. Sams yourself HTML and CSS in 24 Hours. ed. SAMS 2005.
[34] FREEMAN, E. Head First HTML & CSS. ed. Oreilly & Assoc 2005.
[35] MARCONDES, C. A. HTML 4.0 Fundamental. ed. Erica 2005.
[36] GOODMAN, D. JavaScript -A Bíblia. ed. Campus 2001.
[37] Taglib. Disponível em: <http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html>. Visitado pela última vez em 28/04/2006.
[38] SHAWN, B. JSTL in Action. ed. MANNING 2003.
[39] BUTCHER, A. Sam's Teach Yourself MySQL in 21 Days. ed. SAMES 2002.
[40] MYSQL. Disponível em: <http://www.mysql.com>. Visitado pela última vez em 10/04/2006.
[41] Tomcat. Disponível em: <http://tomcat.apache.org>. Visitado pela última vez em 20/03/2006.
[42] JavaServer Face. Disponível em: <http://java.sun.com/javaee/javaserverfaces>. Visitado pela última vez em 28/04/2006.
[43] TOPLEY, K. Pro JSF – JavaServer Face. ed. SPRINGER VERLAG NY 2004.
[44] LOY, M.; ECKSTEIN, R.; ELLIOTT, J.; WOOD, D.; COLE, B. Java Swing. ed Oreilly & Associates,Inc. 2002.
[45] Mantis. Disponível em: <http://www.mantisbt.org>. Visitado pela última vez em 10/04/2006.
[46] PHP. Disponível em: <http://www.php.net>. Visitado pela última vez em 10/04/2006.
[47] FISHER, M.; ELLIS, J.; BRUCE, J. JDBC API Tutorial and Reference. Ed. Addison-Wesley 2003.
[48] VICENT, M.; HUSTED, T. JUnit em Ação. ed. Ciência Moderna 2005.
[49] BARTIÉ, A. Garantia da Qualidade de Software - Adquirindo Maturidade Organizacional. ed. Campus 2002.
[50] MOLINARI, L. Testes de Software - Produzindo Sistemas Melhores e mais Confiáveis. ed. Érica 2005.
[51] HEUSER, A. C. Projeto de Banco de Dados. ed. Sagra Luzzatto 2004.

Apêndice A

Caso de Uso UC_06 Exibir Histórico do Documento

Histórico de Revisão

Data	Versão	Descrição	Autor
25/11/2005	1.0	Versão Inicial	Paulo André
12/01/2006	2.0	Correção e mudança de campos e cor do layout de telas	Paulo André
26/03/2006	3.0	Adição de novos requisitos e mudança no layout de telas	Paulo André

3 1. Descrição

Esse caso de uso especifica a consulta do histórico do documento com a finalidade de acompanhar o andamento do documento.

4 2. Pré-condições

O usuário deve estar autenticado no sistema apresentando o perfil de Chefe de departamento ou Funcionário.

3. Fluxo de Eventos

3.1 Fluxo Básico

[P1] O caso de uso começa quando o usuário seleciona a opção “Histórico→ “Pesquisar” na Interface Principal.

[P2] O sistema exibe a interface *(I01) – Consultar Histórico do Documento.*
[P3] O usuário preenche o(s) campos de pesquisa e aciona o comando "Pesquisar".
[P4] O sistema valida os parâmetros da pesquisa informados no passo [P3] de acordo com a seção 6.1.2.
[P5] O sistema exibe a (I02) – *Lista de Documentos* com todos os protocolos que atenda ao critério de pesquisa.
[P6] O usuário seleciona o documento a ser exibido o histórico de andamento no sistema.
[P7] O sistema exibe a (I03) – *Histórico do Protocolo* com todos os andamentos do documento.
[P8] O usuário aciona o comando "detalhar" um encaminhamento
[P9] O sistema exibe a (I04) – *Detalhe de Encaminhamento*.

3.2 Subfluxos

Não há.

3.3 Fluxos Alternativos

Não há.

3.4 Exeções

3.4.1 (E01) – Número de Protocolo inexistente no departamento.

3.4.1.2 Pré-condições

No passo [P3] do fluxo básico o usuário informou o número de protocolo inexistente.

3.4.1.3 Passos

[P1] O sistema exibe a *(I01) Consultar Histórico do Documento* com a mensagem "Protocolo não encontrado.".
[P2] O sistema retorna ao passo [P3] do fluxo básico.

4. Pós-condições

Não há.

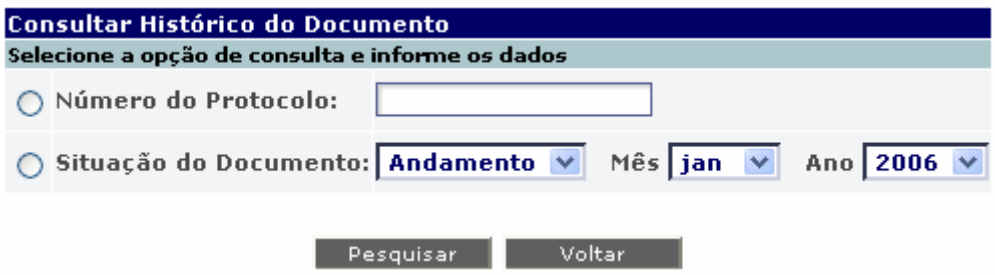
5. Regras de negócio específicas

Não há.

5 6. Detalhamento das Interfaces com Usuários

6.1 (I01) – **Consultar Histórico do Documento**

6.1.1 Imagem



6.1.2 Campos

No.	Nome	Descrição	Valores Válidos	Tipo	Restrições
1	Nº do Protocolo	Botão de seleção	----------	String	----------
2	Situação do Documento	Botão de seleção		Inteiro	Obrigatório caso o botão de radio tenha sido selecionado.
3	Mês	Mês da pesquisa	Um dos meses do ano.	Data	Obrigatório caso o botão de radio tenha sido selecionado
4	Ano	Ano da pesquisa	Nome do requisitante	String	Obrigatório caso o botão de radio tenha sido selecionado.

6.1.3 Comandos

No.	Nome	Ação	Restrições
1	Pesquisar	Exibe a interface (I02) caso não haja violação na pesquisa.	Sempre exibido. Sempre habilitado
2	Voltar	O sistema retorna para o menu da aplicação.	Sempre exibido. Sempre habilitado.

6.2 (I02) – Lista de Documentos

6.2.1 Imagem

6.2.2 Campos

No.	Nome	Descrição	Valores Válidos	Tipo	Restrições
1	Nº. Protocolo	Exibe o nº. do protocolo do documento.		Inteiro	Não habilitado para alterar.
2	Tipo de Documento	Especifica em qual categoria o documento se enquadra.	Ex: memorando, ofício, etc.	String	Não habilitado para alterar.
3	Situação	Situação do documento até o presente momento no sistema.	A-Andamento S-Suspenso D-Deferido I-Indeferido V-Arquivado	Caracter	Não habilitado para alterar.
4	Requisitante	Exibe o nome de quem solicitou/ encaminhou o documento		String	Não habilitado para alterar.

5	Data de Recebimento	Exibe a data em que o documento foi recebido no departamento do usuário autenticado.	Data válida no formato dd/mm/aaaa.	Data	Não habilitado para alterar.

6.2.3 Comandos

No.	Nome	Ação	Restrições
1	Detalhar	Exibe a tela (I03) com os dados referentes ao documento selecionado.	Sempre exibido. Sempre habilitado
2	Voltar	O sistema retorna para (I01)	Sempre exibido. Sempre habilitado.

6.3 (I03) - Histórico do Protocolo

6.3.1 Imagem

6.3.2 Campos

No.	Nome	Descrição	Valores Válidos	Tipo	Restrições
6	Requisitante	Exibe o nome de quem solicitou/ encaminhou o documento		Inteiro	Não habilitado para alterar.
7	Situação	Situação do documento mais recente	A-Andamento S-Suspenso D-Deferido I-Indeferido V-Arquivado	String	Não habilitado para alterar.
8	Tipo de Documento		Ex: memorando, ofício, etc.	String	Não habilitado para alterar.

9	Anexos	Exibe o nome de quem solicitou/ encaminhou o documento		String	Não habilitado para alterar. Campo dinâmico exibido apenas quando houver anexos associado ao documento
10	Data de Entrada	Exibe a data em que o documento foi encaminhado.	Data válida no formato dd/mm/aaaa	Data	Não habilitado para alterar.
11	Departamento de Entrada	Exibe o nome do departamento que deu entrada no documento	Departamento previamente cadastrado	String	Não habilitado para alterar.
12	Origem	Nome do departamento que enviou o encaminhamento		String	Não habilitado para alterar.
13	Enviado	Data do envio do documento	Data válida no formato dd/mm/aaaa	String	Não habilitado para alterar.
14	Destino	Departamento destinatário do documento		String	Não habilitado para alterar.
15	Recebido	Data em que o documento fora recebido pelo destinatário	Data válida no formato dd/mm/aaaa	String	Não habilitado para alterar.

6.3.3 Comandos

No.	Nome	Ação	Restrições
1	Detalhar	Aciona a tela referente a (I04).	Sempre exibido. Sempre habilitado
2	Voltar	O sistema retorna para (I01)	Sempre exibido. Sempre habilitado.

6.4 (I04) - Detalhe de Encaminhamento

6.4.1 Imagem

6.4.2 Campos

No.	Nome	Descrição	Valores Válidos	Tipo	Restrições
1	Protocolo	Exibe o nº. do protocolo referente ao critério de pesquisa		Inteiro	Não habilitado para alterar.
2	Situação	Situação do encaminhamento selecionado previamente	A-Andamento S-Suspenso D-Deferido I-Indeferido V-Arquivado	String	Não habilitado para alterar.
3	Origem	Departamento + telefone de quem enviou o encaminhamento		String	Não habilitado para alterar.
4	Destino	Departamento + telefone de quem recebeu/ ou está pra receber o encaminhamento		String	Não habilitado para alterar.
5	Remetente	Funcionário + fone/ramal		String	Não habilitado para alterar.
6	Destinatário	Departamento + Funcionário autenticado no sistema.		String	Não habilitado para alterar.
7	Nota (s)	Exibe todos os comentários referente ao encaminhamento		String	Não habilitado para alterar.

6.4.3 Comandos

No.	Nome	Ação	Restrições
3	Voltar	O sistema retorna para (I01)	Sempre exibido. Sempre habilitado.

Apêndice B

Classe de Teste

Os passos básicos para a criação de um teste unitário com o framework JUnit são os seguintes:

- Definir uma classe de teste que herda de junit.framework.TestCase.
- Sobrescrever os métodos setUp() e tearDown(), chamados respectivamente antes e depois de cada um dos métodos de teste. O setUp() geralmente cria objetos utilizados por todos os testes e realiza qualquer outra inicialização necessária. O tearDown() finaliza os testes geralmente fechando conexões ou arquivos utilizados pelos testes.
- Criar os métodos de teste, cujos nomes devem ser prefixados pela palavra “test”, chamando métodos *assert* do framework para verificar resultados fornecidos pela classe de teste.
- Opcionalmente, criar conjuntos de testes ou suítes que permitirão a execução de todas as classes de teste em um único ciclo.

Exemplo de Teste para o fluxo de protocolos

A classe de teste referenciada aqui trata-se apenas as operações básicas do controle de documentos.

```
/*
 * Sistema: S@DPOLI
 * TestControladorDocumento.java
 *
 * Copyright (c) Escola Politécnica de Pernambuco - Poli.
 * Este software pertence a Escola Politécnica de Pernambuco.
  */

package br.poli.upe.teste;

import junit.framework.TestCase;
import br.poli.upe.fachada.FachadaCadastro;
import br.poli.upe.fachada.FachadaDocumento;
import br.poli.upe.negocio.dep.DepartamentoVO;
```

```
import br.poli.upe.negocio.doc.DocumentoVO;
import br.poli.upe.negocio.doc.EncaminhamentoDocVO;
import br.poli.upe.negocio.doc.SituacaoDocVO;
import br.poli.upe.negocio.excecao.SadPoliException;
import br.poli.upe.negocio.func.FuncionarioVO;
import br.poli.upe.util.Util;

/**
 *  Classe de teste referente a verificação e validação do fluxo de documentos
no sistema.
 * @author Paulo André
 */
public class TestControladorDocumento extends TestCase {
 private  EncaminhamentoDocVO encaminhaVO = null;
 private  DocumentoVO documentoVO = null;
 private final  Integer protocolo = new Integer("222298");
 private final Integer codEncaminha = new Integer("47");

 public TestControladorDocumento(String nome){
        super(nome);
}

/**
 * Método responsável por inicializar objetos para a execução.
 */
protected void setUp () throws Exception {
  super.setUp ();
  documentoVO = getDocumentoVO();
  encaminhaVO = getEncaminhamentoVO();
  encaminhaVO.setDocVO(getDocumentoVO());
}

/**
 * Método responsável por liberar os recurso alocados para o teste no final da
execução.
 */
protected void tearDown () throws Exception {
  super.tearDown ();
  encaminhaVO = null;
}

/**
*  Este  teste  verifica  se  o  envio  de  um  encaminhamento  foi  de  fato
*  concretizado com a situação "Andamento" e verifica se o sistema definiu um
*  número de protocolo para este encaminhamento.
*/
public void testEncaminharDocumento(){
   EncaminhamentoDocVO enVO = null;
    try{
     encaminhaVO = FachadaDocumento.cadastrarEncaminharDocumento(encaminhaVO);

    assertEquals(encaminhaVO.getDocVO().getProtocolo(), protocolo);
    assertEquals(SituacaoDocVO.CODIGO_ANDAMENTO,encaminhaVO.getCodSituacao());
   }catch(SadPoliException sad){

/*caso o envio falhe o encaminhamento será nulo e a falha ocorrerá*/
      assertNotNull(encaminhaVO);
  }
}
```

```
/**
* Método responsável por confirmar o recebimento do documento no departamento
* destinatário.
*/
public void testConfirmarRecebimento(){
   EncaminhamentoDocVO enVO = null;
     try{
      encaminhaVO.setCodigo(codEncaminha);
      encaminhaVO.getDocVO().setProtocolo(protocolo);
      encaminhaVO.setDataDeRecebimento(Util.dateToString(Util.MASCARA_DD_MM_YY
YY,Util.getDataAtual()));
   FachadaDocumento.confirmarRecebimento(encaminhaVO);
    enVO =  FachadaDocumento.obterEncaminhamento(codEncaminha);


assertEquals(enVO.getDataDeRecebimento(),Util.dateToString(Util.MASCARA_DD_MM_
YYYY,   Util.getDataAtual()));
 }catch(SadPoliException sad){
      assertNotNull(encaminhaVO);
 }
}

/**
*  Método  responsável  por  modificar  a  situação  do  documento  previamente
*recebido.
*/
public void testAlterarSituacaoDoDocumento(){
   DocumentoVO docVO = null;
    SituacaoDocVO sitVO = new SituacaoDocVO();
      sitVO.setCodSituacao(SituacaoDocVO.CODIGO_DEFERIDO);
      sitVO.setSituacao(SituacaoDocVO.DEFERIDO);

      try{
          docVO = FachadaDocumento.obterDocumento(protocolo);
          docVO.setSituacaoDocVO(sitVO);
          encaminhaVO.setDocVO(docVO);
          encaminhaVO.setCodigo(codEncaminha);

          FachadaDocumento.atualizarSituacaoDoDocumento(encaminhaVO);
          DocumentoVO docVO2 =FachadaDocumento.obterDocumento(protocolo);
           Character codSituacao = docVO2.getSituacaoDocVO().getCodSituacao();

        assertEquals(codSituacao,SituacaoDocVO.CODIGO_DEFERIDO);
      }catch(SadPoliException sad){
       assertNotNull(docVO);
      }
}

/**
 * Método responsável por reencaminhar um documento.
 */
public void testReencaminharDocumento(){

try{
    FuncionarioVO     funcOrigem     =     FachadaCadastro.obterFuncionario(new
Integer("331044"));
    encaminhaVO.setFuncionarioOrigem(funcOrigem);
    FuncionarioVO     funcDestino     =     FachadaCadastro.obterFuncionario(new
Integer("191919"));
   encaminhaVO.setFuncionarioDestino(funcDestino);
```

```
   documentoVO.setProtocolo(protocolo);
   documentoVO.setAnexos("Memorando 2141, Carta ao Diretor");
   encaminhaVO.setDocVO(documentoVO);
   encaminhaVO =FachadaDocumento.cadastrarEncaminharDocumento(encaminhaVO);

/*caso um novo encaminhamento tenha sido realizado então este encaminhamento
apresentará um novo código interno.  */
      assertFalse(encaminhaVO.getCodigo().equals(codEncaminha));
  }catch(SadPoliException sad){
         assertNotNull(encaminhaVO);
 }
}

/**
* Método responsável por arquivar um documento.
*/
public void testArquivarDocumento(){
 try{
documentoVO.setProtocolo(protocolo);
documentoVO.setDataDeArquivamento(Util.dateToString(Util.MASCARA_DD_MM_YYYY,
Util.getDataAtual()));
DepartamentoVO depVO =encaminhaVO.getFuncionarioDestino().getDepartamento();
documentoVO.setCodDepArquivamento(depVO.getNumero());
encaminhaVO.setDocVO(documentoVO);
encaminhaVO.setCodigo(codEncaminha);
FachadaDocumento.arquivarDocumento(encaminhaVO);

/*
 * Documentos com situação em "Andamento" não pode ser arquivado.
 * */

 }catch(SadPoliException sad){
      assertEquals(sad.getMessage(),"error.situacao.invalida.arquivar");

 }
}

/**
 * Criação de um objeto documento a ser manipulado pelo teste.
 * @return um objeto documento
 */
private DocumentoVO getDocumentoVO(){
  DocumentoVO vo = new DocumentoVO();
    vo.setCodDepEntrada(new Integer(1));
    vo.setDataDeEntrada( Util.dateToString(Util.getDataHoraAtual ()));
    vo.setMatricula(new Integer("221044"));
    vo.setRequisitante("Camilo França");
    vo.setTipoDocumento("Oficio A2");

   SituacaoDocVO sitVO = new SituacaoDocVO();
   sitVO.setCodSituacao(SituacaoDocVO.CODIGO_ANDAMENTO);
   sitVO.setSituacao(SituacaoDocVO.ANDAMENTO);
   vo.setSituacaoDocVO(sitVO);
  return vo;

}

/**
 * Método responsável por criar um objeto encaminhamento para um documento a
ser registrado no sistema.
```

```
 * @return um objeto encaminhamento
 *   @throws   SadPoliException   referente   a   qualquer   problema   ocorrido
internamente ao sistema.
 */
private EncaminhamentoDocVO getEncaminhamentoVO()throws SadPoliException{
   EncaminhamentoDocVO vo = new EncaminhamentoDocVO();
   vo.setCodSituacao(SituacaoDocVO.CODIGO_ANDAMENTO);
   vo.setDataDeEncaminhamento(Util.dateToString(Util.getDataHoraAtual ()));

DepartamentoVO depOrigem = FachadaCadastro.obterDepartamento(new Integer(1));

vo.setDepartamentoOrigem(depOrigem);

DepartamentoVO depDestino = FachadaCadastro.obterDepartamento(new Integer(2));
vo.setDepartamentoDestino(depDestino);

FuncionarioVO      funcOrigem      =      FachadaCadastro.obterFuncionario(new
Integer("221044"));
vo.setFuncionarioOrigem(funcOrigem);

FuncionarioVO      funcDestino      =      FachadaCadastro.obterFuncionario(new
Integer("331044"));
vo.setFuncionarioDestino(funcDestino);

vo.setObservacao("O documento deve ser tratado com urgência");
 return vo;
}

}
```