Universidade de São Paulo
Instituto de Matemática e Estatística

MAC499 - Trabalho de Formatura Supervisionado

Sistema de Gerenciamento de Workflows

Monografia

Carlos H. de Fernandes - N^{o}USP:3286544 - carloshf at linux.ime.usp.br
Cleber Miranda Barboza - N^{o}USP:3286353 - cleberc at linux.ime.usp.br
Giuliano Mega - N^{o}USP:3286245 - giuliano at linux.ime.usp.br
Pedro Losco Takecian - N^{o}USP:3286307 - plt at linux.ime.usp.br

Orientador

Alfredo Goldman vel Lejbman - gold at ime.usp.br

8 de dezembro de 2003

Sumário

1 Introdução

1.1 Visão geral

Este trabalho teve como objetivo o desenvolvimento de um sistema de gerenciamento de processos de produção (*workflows* ou fluxos de trabalho), desenvolvido partindo-se da análise detalhada de diversos softwares já consolidados no mercado.

As *Workflow Engines* são ferramentas especificamente voltadas para a modelagem de *business processes*; isto é, processos dinâmicos cuja evolução é condicionada ao cumprimento de tarefas pré-definidas. Estes processos são constituídos por diversos estados; é trabalho da *Workflow Engine* fornecer o ambiente e as ferramentas necessárias para que seja possível modelar, com o maior grau de flexibilidade e abrangência possíveis, tais processos. É também papel da *Workflow Engine* refletir no sistema, a cada instante, o estado global desses processos, bem como distribuir as tarefas e colher os resultados das diversas partes integrantes.

Para uma organização, há varias vantagens em utilizar um Sistema de Gerenciamento de Fluxos de Trabalho, entre elas, podemos citar:

- melhora da eficiência da organização;
- aumento de produtividade;
- aperfeiçoamento de processos e relatórios de controle;
- maior aceitação, pelos empregados, dos regulamentos internos e externos;
- melhora da vantagem competitiva;
- aumento do conhecimento dos processos da organização.

Nos tópicos que se seguem, iremos abordar um caso de uso da aplicação, a arquitetura utilizada, algumas decisões de implementação, e a organização adotada durante o desenvolvimento do projeto.

1.2 Definições

Antes de começar, é importante estabelecer algumas definições que serão essenciais para o entendimento do projeto:

- ***Workflow*** (WF): automação de um processo de negócio, por inteiro ou em parte, durante o qual informações, tarefas e documentos são passados de um participante para outro, respeitando um conjunto de regras procedurais.
- ***WorkItem***: chamamos de *WorkItem* uma atividade isolada de um processo definido. Uma instância de um processo é uma coleção de *WorkItems* (um por atividade executada ou em execução).
- **Agentes**: chamamos de agentes **usuários** ou **papéis**. Os papéis são estruturas hierárquicas que foram utilizadas para a representação de grupos de usuário. Cada usuário deve pertencer a pelo menos um papel.

1.3 Caso de uso

Aqui, o intuito é mostrar um exemplo de como este sistema poderia ser utilizado, ou seja, uma situação em que um Sistema Gerenciador de *Workflows* pode ser bastante útil.

Empresa de desenvolvimento de software:

Em uma empresa de desenvolvimento de software, temos uma série de etapas que devem ser seguidas para começarmos a desenvolver um software.

Estas etapas são constituídas por diferentes atividades e exercidas por várias pessoas. Algo importante a ser notado é que essas etapas são interdependentes, isto é, as informações geradas em uma etapa são, muitas vezes, utilizadas nas outras. O fluxo de informações tem, portanto, vital importância neste processo.

Basicamente, temos o seguinte fluxo: Tudo começa com o encontro de um funcionário da empresa com o cliente, que dirá o que quer como resultado do desenvolvimento. Este funcionário tem como objetivo extrair o que é realmente importante das informações que foram passadas a ele. Em seguida entrará com os dados do projeto e do cliente no sistema interno da empresa.

A seguir, estes dados irão para as mãos de um projetista, que será responsável por desenvolver uma solução para o projeto, ou seja, fazer uma modelagem do sistema a ser desenvolvido.

Posteriormente, esta modelagem deverá ir para as mãos de outro funcionário, responsável por apresentá-la ao cliente, de um modo que seja fácil de entender e que faça o cliente perceber se é isso o que ele realmente quer. Neste ponto, o cliente tem duas escolhas. Ou ele aceita o projeto, ou ele o rejeita. Caso ele aceite, esta informação irá ao gerente da empresa, que fará um contrato com o cliente e dará o aval para o início do desenvolvimento. Caso o cliente rejeite, esta informação também irá ao gerente, que decidirá se este projeto volta para as mãos do projetista para ser modificado ou se a empresa desistirá do desenvolvimento, acabando por aqui o processo. Percebe-se aqui que houve um fluxo de trabalho entre os diversos participantes.

Um sistema gerenciador de *workflows* seria perfeito para ser utilizado como sistema interno desta empresa, pois ele tem justamente a função de mostrar para os participantes corretos quais atividades devem desempenhar e o momento certo em que devem agir. Além disso, o gerente poderia ter um controle completo sobre o andamento do projeto, sobre a produtividade de cada funcionário ou de um grupo de funcionários, fazer auditorias, encontrar possíveis gargalos no processo de produção entre várias outras vantagens.

Vamos supor que a empresa quer, agora, criar um novo departamento: controle de qualidade. Agora, antes da modelagem ir parar nas mãos do funcionário que a apresenta ao cliente, o modelo deve ir para um funcionário do controle de qualidade. O que temos aqui é uma mudança no processo de produção da empresa. Um sistema gerenciador de *workflows* tem também, como uma de suas características, tornar fácil este tipo de atualização e até mesmo a criação de novos processos.

2 Arquitetura do sistema

2.1 Visão geral

A arquitetura utilizada é baseada em *thin client*, que impõe poucos requisitos sobre os clientes (apenas um navegador que suporte HTML), e dividida em 3 camadas: camada de apresentação, camada de negócio e camada de dados. Os clientes comunicam-se apenas com a camada de apresentação. A lógica de negócios encontra-se no servidor, único que tem acesso à base de dados.

Através da escolha dessa arquitetura, conseguiu-se atingir alguns objetivos, tais como a facilidade para desenvolver um sistema distribuído, que fosse escalável, seguro e de alta disponibilidade, além de facilitar sua manutenção.

A opção por *thin clients* torna o sistema muito flexível, pois o acesso ao sistema não depende de um microcomputador. O usuário pode acessá-lo de qualquer aparelho (como PDAs e celulares) que possua um navegador.

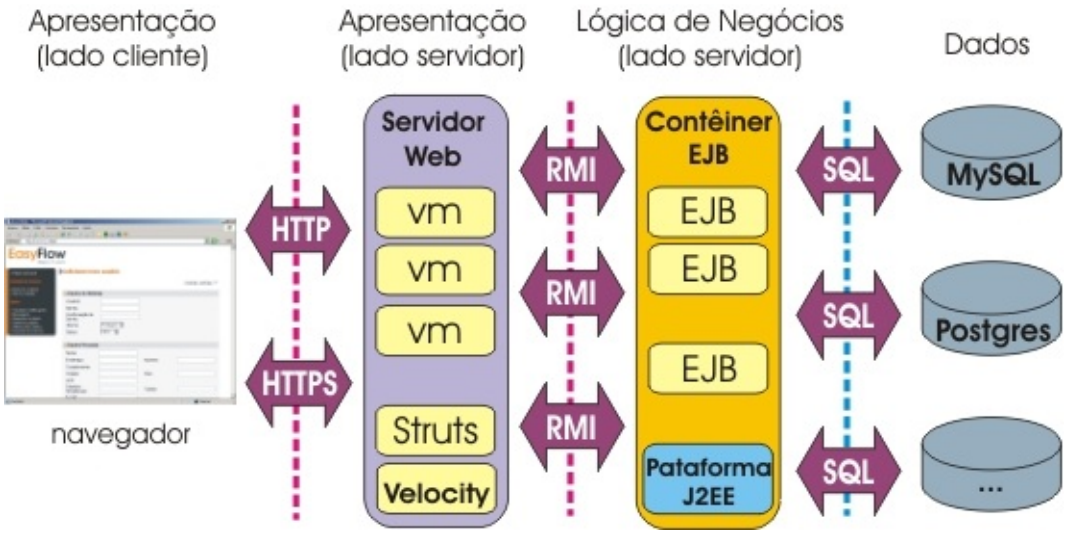

Figura 1: Arquitetura do sistema dividida em 3 camadas.

Nas 3 camadas foram utilizados softwares livres:

- servidor web: Tomcat;
- servidor de aplicação: JBoss;
- banco de dados: Postgres.

Optou-se por um servidor de aplicação que implementasse a especificação J2EE pelos seguintes motivos:

- flexibilidade pelo fato de ser multiplataforma;
- é um padrão de fato, uma tecnologia muito disseminada;
- fornece serviços avançados de *middleware*;

- integra-se facilmente com outras tecnologias.

Nos próximos tópicos, cada camada será detalhada.

2.2 Camada de apresentação

No desenho deste projeto, os principais componentes presentes na camada de apresentação são:

- **controle de fluxo de atividades**: como o sistema possui uma interface web, teoricamente, seria possível acessar qualquer página tendo conhecimento prévio de seu endereço, pois não haveria nada que impedisse o usuário de acessar o que ele quisesse na ordem que ele quisesse. Isto constitui uma falha de segurança do sistema. O controle de fluxo vem, justamente, determinar quais são as seqüências de acessos que devem ser seguidas, forçando o usuário a seguir os passos corretos;

- **controle de acesso**: o acesso ao sistema só é permitido a usuários previamente cadastrados e que, portanto, possuem login e senha. Objetivando aumentar a segurança, utilizamos o protocolo https para a comunicação criptografada do cliente com o servidor. Assim, a senha fica protegida contra a interceptação de terceiros;

- **controle de visibilidade**: através desse controle é possível determinar o que usuário pode ou não enxergar após ter acesso ao sistema, isto é, determina quais são os recursos do sistema que ficarão disponíveis ao usuário e quais ele não terá acesso;

- **desenho de WF's**: o sistema fornece interfaces para o desenho de WF. Assume-se que o usuário responsável pelo desenho de um WF tenha conhecimentos de *Redes de Petri Coloridas*;

- **controle de WF's**: após o desenho de um WF, pode-se criar várias instâncias do mesmo. Assim, o sistema fornece meios para controlar essas instâncias, além do controle dos agentes responsáveis por cada item de trabalho associado a cada fase de processo de um WF.

Para o desenvolvimento da camada de apresentação fez-se uso do modelo MVC (*Model-View-Controller*). Este modelo descreve a organização de uma aplicação em três módulos separados:

- ***Model*** **("Modelo")**: um para o modelo de aplicação, contendo a representação de dados e lógica de negócios;

- ***View*** **("Apresentação")**: um segundo que trata aspectos relacionados à apresentação e entrada de dados de usuário;

- ***Controller*** **("Controle")**: e um terceiro que é responsável por despachar requisições e controlar seus fluxos.

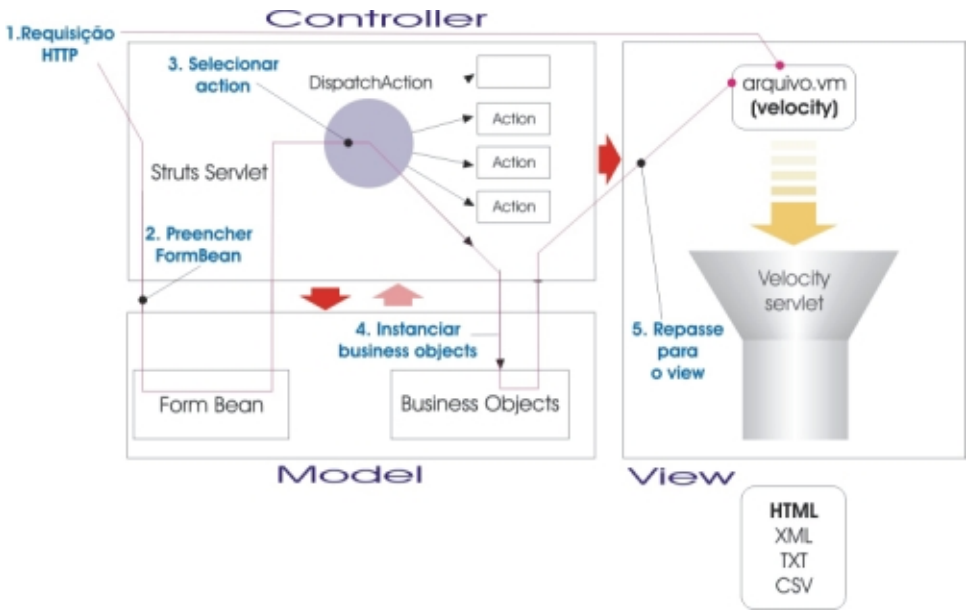

Figura 2: Modelo MVC aplicado a camada de apresentação.

Na literatura, o modelo MVC aplicado à apresentação é conhecido em duas versões que são chamadas de **Modelo 1** e **Modelo 2**. A arquitetura proposta no Modelo 1 diz que um navegador, por exemplo, tem acesso direto ao módulo de apresentação e é o módulo de apresentação que faz a comunicação direta com o módulo responsável pelo modelo. Já na arquitetura proposta no Modelo 2, há a introdução do módulo de controle que desempenha o papel descrito acima.

Para a elaboração desse projeto, fez-se uso do Modelo 2. Através desse modelo foi possível centralizar a lógica de despachar requisições em uma única classe (ver figura), o que facilitou a implementação do controle de segurança e acesso ao sistema.

Para facilitar o uso do Modelo 2, foi utilizado o *framework* ("arcabouço") Struts. O Struts é um *framework* de código aberto que fornece componentes de controle e interage com outras tecnologias de acesso à base de dados padrões como JDBC e EJB, além de interagir com várias ferramentas fornecidas por terceiros como Hibernate, iBATIS, ou Object Relational Bridge.

A princípio, o Struts também poderia ser utilizado como ferramenta para o módulo de apresentação. Entretanto, para que isso ocorresse, as páginas web deveriam ser implementadas utilizando-se JSP's (*Java Server Pages*), que apresentam a desvantagem de serem difíceis de serem depuradas.

Por este motivo, optou-se pelo uso da ferramenta de templates Velocity para ser utilizado no módulo de apresentação. O Velocity possui uma linguagem de templates (*Velocity Template language* - VTL) simples, porém muito poderosa. Templates são arquivos com grande parte de sua estrutura e formatação prontos, contendo diretivas indicando valores a serem inseridos e ações a serem executadas pelo sistema.

2.2.1 Interface web

Houve uma preocupação particular com a interface. Precisava-se facilitar ao máximo a navegação pelo sistema, pois os usuários que o utilizariam não têm, necessariamente, familiaridade com informática.

Ao mesmo tempo que havia necessidade de facilitar a navegação, seria desejável que se reutilizasse ao máximo o código HTML, facilitando a manutenção das páginas. Nem sempre o *design* de uma página era ideal para todos os tipos de dados que desejava-se apresentar, e a escolha de um *design* padrão não foi uma tarefa fácil. Um *design* padrão era essencial, pois desejava-se manter uma similaridade entre as páginas ao mesmo tempo que a apresentação dos dados fosse satisfatória.

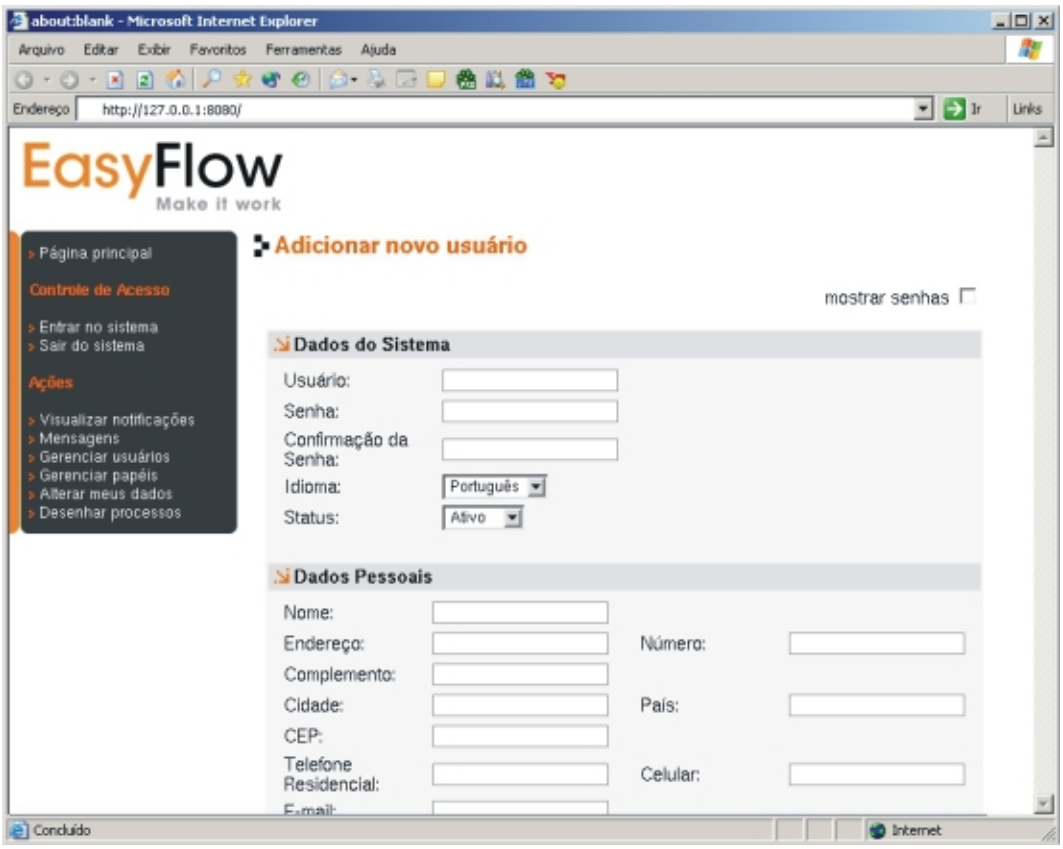

Figura 3: Uma tela do sistema.

Uma solução adotada, que facilitou bastante o desenvolvimento das páginas e diminuiu a possibilidade de ocorrência de erros, foi o controle de *layout*. Esta solução é uma extensão do Velocity, e está disponível na página do projeto Jakarta. Na solução convencional, um template possui diretivas que indicam a inclusão do código de outras páginas (usadas para reutilização de código que são comuns a muitas páginas, como cabeçalho, rodapé e menu), porém há a desvantagem de se repetir a diretiva em todas as páginas que a utilizarão, o que dificulta uma futura alteração da diretiva. Já na solução adotada, um arquivo de template contém apenas o corpo de uma página, e este corpo é inserido em uma página de *layout* automaticamente pelo sistema de templates.

Uma dificuldade encontrada no desenho da interface está relacionada a montagem de um

processo. O objetivo inicial era que o usuário fosse capaz de montar um processo da forma mais intuitiva possível, sem a necessidade de conhecer os conceitos de uma Rede de Petri colorida. Nenhuma interface conseguiu atingir este objetivo, e após as tentativas fracassadas, observou-se que não seria um problema o usuário conhecer Redes de Petri, pois a criação de um processo não seria uma tarefa rotineira, além desse conhecimento ser necessário para analisar as atividades de uma organização e definir quais serão os processos.

Outro problema enfrentado foi a modificação das interfaces durante o desenvolvimento do sistema. Nem sempre se conseguiu manter o desenho original das interfaces, o que, na maioria das vezes, gerava alterações em outras interfaces, atrasando o desenvolvimento do projeto.

2.3 Camada de negócio

A camada de negócio compreende dois subconjuntos:

- **Extensões** englobando tudo aquilo que foi necessário para, partindo da biblioteca (Núcleo), contruir um sistema completo de gerenciamento de workflow (interfaces para o controle de usuários, interface para uso de contêiners J2EE, mbeans para integração).
- Uma biblioteca genérica (**Núcleo**) para manipulação de WF's que fornece facilidades de modelagem, verificação formal e um ambiente de execução para os modelos cujo mecanismo de auto-persistência é baseado na API JDO (RFS-12).

2.3.1 Extensões

O núcleo fornece interfaces adequadas para a manipulação de uma Rede de Petri. Entretanto, uma Rede de Petri não é suficiente para descrever um WF. Um WF é composto por *WorkItens*, e estes, por sua vez, estão associados a telas, campos, botões, agentes e ações. O problema de relacionar esses elementos de maneira consistente e eficiente é um pouco mais complicado do que parece, pois existem as chamadas instâncias de um WF.

As instâncias de um WF são, abstratamente, clones de um WF. Cada instância está associada a um único WF, além de possuir seu próprio estado e ser independente de outras instâncias. De uma certa maneira, uma instância possui um tipo, ou seja, ao olharmos para uma instância podemos dizer a qual WF ela está associada. O único elo de ligação entre duas instâncias de um mesmo "tipo" é o WF do qual foram clonados.

Para entrarmos num campo mais familiar, podemos fazer a associação de instâncias de um WF com as instâncias de classes presentes no paradigma de Programação Orientada a Objetos (POO), onde as classes poderiam ser interpretadas como sendo os WF's. Do mesmo jeito que no paradigma POO, onde um objeto (instância de uma classe) possui estados, membros e métodos para a manipulação de seus membros, uma instância de um WF também possui estados (os estados descrevem em que fase do processo um WF se encontra - é basicamente para esse controle de estados que o núcleo foi desenvolvido), membros (campos, botões, telas e agentes associados) e métodos que são nada mais do que interfaces flexíveis para a manipulação de seus membros.

Para a representação desses elementos na modelagem do sistema, fez-se uso de metaclasses. A idéia de metaclasses está relacionada à possibilidade de criar classes em tempo de execução, ou seja, criar novos tipos (WF's) sem alterar o código do sistema. Optou-se por essa idéia pelo fato de metaclasses serem flexíveis, configuráveis e facilmente adaptáveis.

Tanto o núcleo como as extensões utilizaram essa idéia através da aplicação do padrão *TypeSquare* (ver [5]) o qual permitiu descrever e relacionar tipos de dados e instâncias desses tipos de maneira eficiente e consistente.

Além disso, foi utilizado o padrão Fachada para evitar que camadas superiores tenham acesso direto aos mecanismos de persistência de dados, ou seja, a única maneira de realizar persistência de dados é através das interfaces fornecidas pelos chamados Beans de Fachada. Os Beans de Fechada são classes que seguem um padrão estabelecido pela arquitetura J2EE e são utilizados pelo contêiner J2EE para prover facilidades quanto à segurança e chamada de métodos remotos. A chamada a esses métodos pode ser feita por qualquer camada acima da camada de negócios.

Como dito, camadas superiores não têm acesso direto aos mecanismos de persistência de dados, entretanto, como em muitos sistemas, determinados dados precisam ser persistidos. No caso, a camada de apresentação provê interfaces para entrada de dados ao usuário que precisam ser persistidos. Para repassar esse dados à camada de negócios, para que ela realize a persistência deles, foi utilizado o padrão *Data Transfer Object* ("Objeto de Transferência de Dados"), também conhecido como DTO. O padrão DTO encapsula os dados num único objeto. O benefício desse padrão é evidente, pois ao invés da camada de apresentação realizar diversas chamadas remotas à camada de negócios para a persistência de cada dado, faz-se uma única chamada remota, passando o DTO, com todos os dados encapsulados, como parâmetro.

2.3.2 Núcleo

O núcleo surgiu como conseqüência natural da filosofia de particionamento de funcionalidades adotada no projeto. Foi formulado, inicialmente, como sendo a camada mais interna na arquitetura do sistema - seu papel seria prover funcionalidades básicas, tais como a criação, gerenciamento e atualização dos modelos de fluxo, através de primitivas pouco abstratas.

Logo no início do processo de modelagem, todavia, notamos ser o núcleo possuidor de uma grande vocação a biblioteca independente, já que fornece uma interface bem definida com um conjunto de primitivas genéricas, sendo completamente desacoplado do restante do sistema. Tais características permitiriam o seu uso em qualquer projeto que necessitasse de funcionalidades de workflows básicas ou simulações de fluxos (fluxos e *workflows* serão utilizados aqui como palavras equivalentes).

O núcleo provê, essencialmente, as seguintes funcionalidades:

- Interface para modelagem de fluxos, criação e instanciamento de templates (metaclasses);
- facilidades que permitem acompanhar o estado dos workflows instanciados: incluem formas de determinar, a qualquer instante, quais as tarefas ativas e quais os participantes nelas envolvidos;
- controle centralizado de recursos, portando interfaces de acesso aos repositórios de modelos e participantes;
- controle interno de persistência através do JDO, comportamento transacional.

Além disso, existe uma coleção de Beans de sessão (que cumprem o papel de fachadas), criados para uso exclusivo da biblioteca em ambientes gerenciados (contêiners J2EE, especificamente).

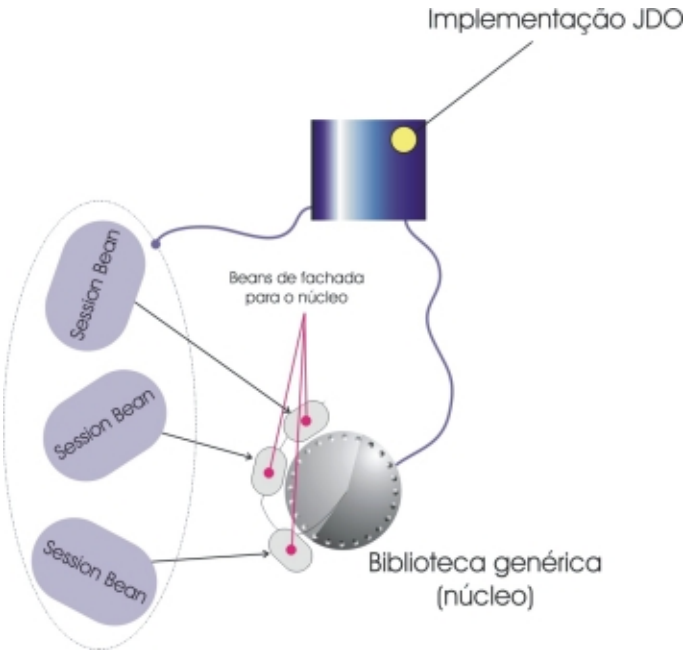

Figura 4: Núcleo e suas fachadas para uso em ambiente J2EE.

Representação Interna

Existem muitas maneiras conhecidas para representar *workflows* - a maior parte delas envolve grafos, a grande maioria cumpre bem o seu propósito de modelagem. O problema encontrado em muitas dessas representações está, portanto, não nas representações em si, mas no fato delas serem heterogêneas e restritas a nichos ou grupos de desenvolvimento. Isso gera uma porção de dificuldades com relação às ferramentas de análise empregadas, que diferem significativamente de uma representação para outra. Além disso, certos modelos são mais próprios a determinados tipos de análise do que outros.

Ferramentas de análise são importantes na medida que tornam possível avaliar um fluxo quanto a diversos aspectos qualitativos e quantitativos antes de colocá-lo em prática (evitando, potencialmente, prejuízos à empresa utilizando o sistema de *workflows*).

A escolha do nosso modelo interno de representação levou em conta, ainda, os seguintes aspectos:

- **Manipulação** - o modelo não pode ser críptico. Um modelo muito complexo tem sua aplicabilidade condicionada à presença de profissionais qualificados. Gostaríamos de um modelo que fosse intuitivo.

- **Expressividade** - não adianta o modelo ser fácil de usar e possuir boas propriedades de análise se ele não for capaz de representar a diversidade de fluxos requerida nos ambientes aos quais a aplicação é voltada. Precisamos, portanto, de um modelo expressivo.

- **Propriedades de análise** - além de darmos preferência a modelos com boas propriedades de análise, foi necessário também separar quais dessas propriedades eram relevantes ao nosso usuário final esperado e quais não eram.

Após estabelecidas as metas, teve início um processo de pesquisa e seleção do modelo. A representação que melhor se enquadrou nos nossos requisitos foi a das Workflow Networks, ou WF-nets.

WF-Nets

WF-nets são, essencialmente, redes de Petri acrescidas de algumas particularidades. De maneira semelhante às redes de Petri clássicas, as WF-nets possuem lugares, transições, arestas (com peso) e tokens. Além disso, todavia, nas WF-nets temos que:

As arestas que partem de transições para lugares podem ter, associadas a elas, expressões lógicas cujos átomos são propriedades. Essas propriedades assumem valores do tipo verdadeiro ou falso, que serão utilizados para avaliar a expressão associada. Durante o disparar de uma transição (onde tokens são consumidos do(s) lugar(es) de origem e produzidos no(s) lugar(es) de destino), apenas aquelas arestas cujas expressões forem verdadeiras produzem tokens. Dessa maneira podemos modelar todos os construtos de roteamento requeridos pelos diversos tipos de *workflow*. Além disso, as propriedades atuam como um equivalente das cores encontradas nas redes de Petri coloridas (*colored Petri nets*).

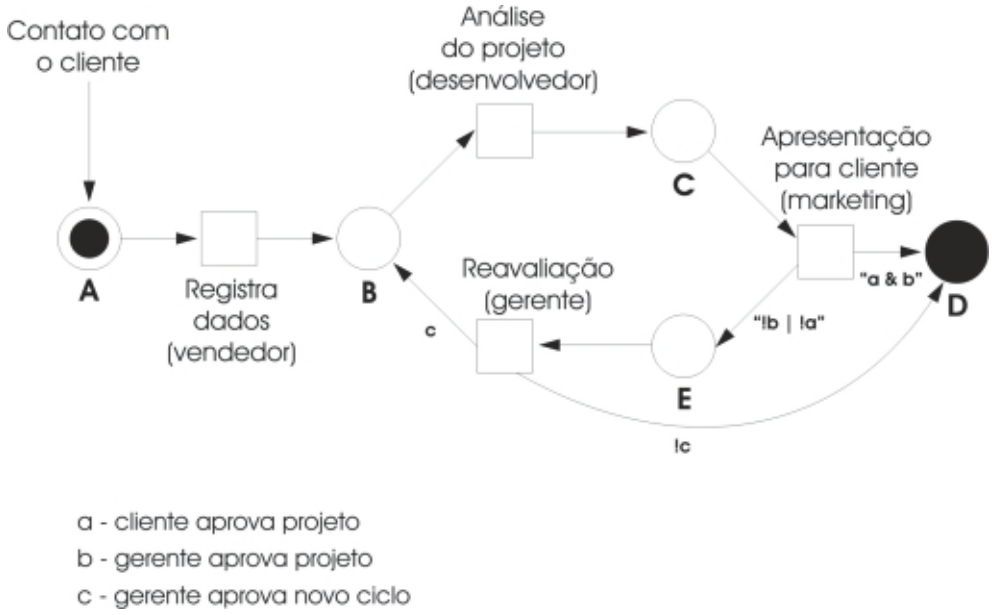

Figura 5: Um exemplo de uma WF-net. Lugares são círculos, transições são quadrados.

Dessa maneira, no nosso exemplo (ver figura nº5) de WF-net, um token que se encontra no lugar **C** irá passar, mediante uma transição, para o lugar **D**, se e somente se a expressão (**a & b**) for verdadeira; isto é, se **a** for verdadeira (cliente aprova projeto) e **b** for verdadeira (gerente aprova projeto). Caso contrário, se **!b** ou **!c** forem verdadeiros (isto é, o cliente **OU** o gerente não aprovam o projeto), a transição irá consumir o token em **C** e produzir um token em **D**. O estado da WF-net num dado instante é representado pelo conjunto dos tokens e suas respectivas posições (em quais lugares eles se encontram).

Quanto às transições, podem ser disparadas tanto por ação do usuário (por exemplo, o usuário completa um item de trabalho como o apresentado na transição "registra dados") quanto pelo expirar de um timer (timeouts). Para o núcleo isso não faz diferença, todas as

entidades que participam de interações com as WF-nets são representadas através de recursos (*resources*). Recursos podem representar usuários, grupos ou até mesmo timers e aplicações - o significado é dado pelas camadas mais externas.

Como nas redes de Petri clássicas, transições (na verdade workitems, como veremos mais adiante) só são ativadas mediantes à presença de um número suficiente de tokens nos lugares conectados a ela em leque de entrada (fan-in). Por outro lado, contrário às redes de Petri clássicas, transições não são disparadas aleatoriamente - requerem, como descrito acima, a participação de um terceiro elemento, uma *resource*. Uma *resource* deve possuir permissão para disparar uma dada transição.

Segue abaixo o diagrama de transição de estados para um item de trabalho (*workitem*), cujo papel é associar as transições (dadas num template) a uma dada instância de um template de fluxo de trabalho. No caso do exemplo dado acima, cada vez que um contato com um cliente for estabelecido, será posta em execução uma nova instância do template que modela esse fluxo, gerando novos *workitems*. O *workitem* é a ponte natural entre as instâncias de um template e o template.

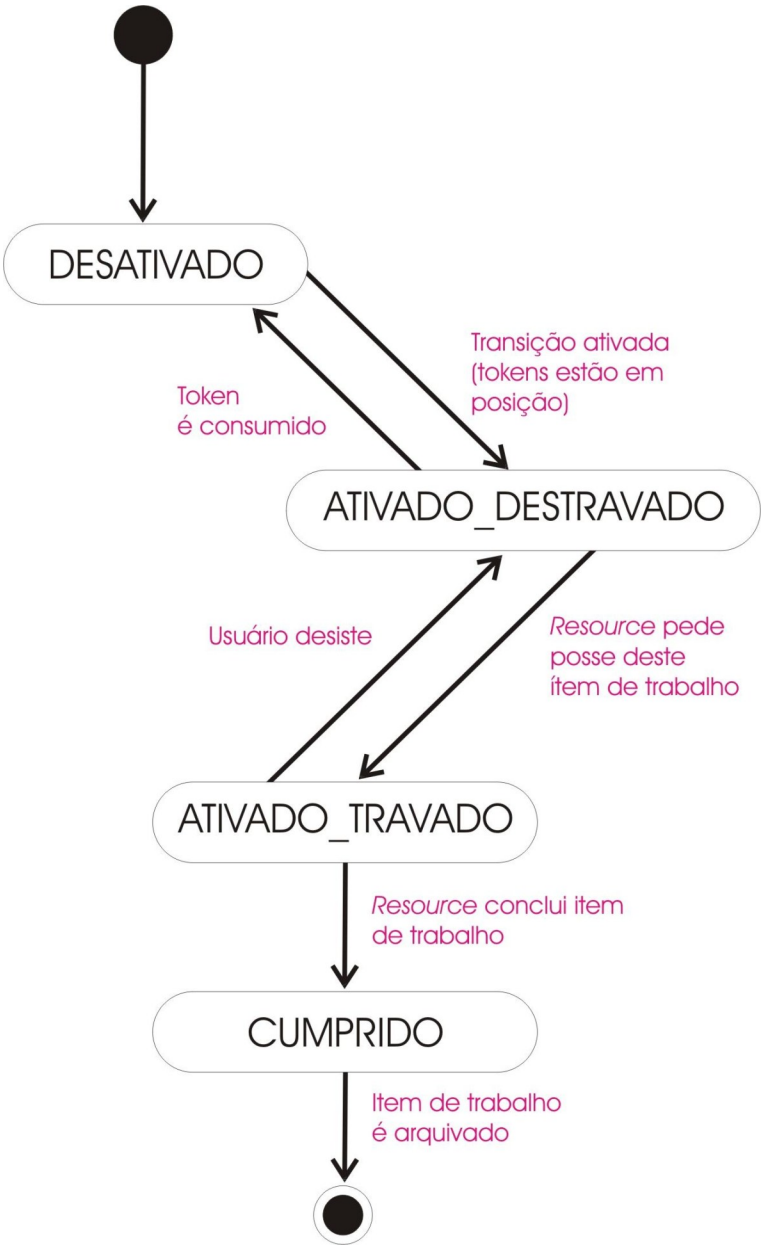

Figura 6: Possíveis estados para um item de trabalho.

O núcleo é, portanto, o responsável por criar, manter e fornecer acesso a todas essas informações.

2.4 Camada de dados

Parte da persistência dos dados no projeto (em especial dentro do núcleo) é feita através de uma implementação da especificação JDO (*Java Data Objects*), que consiste numa especificação de um conjunto de interfaces para fácil persistência e recuperação de modelos de objetos.

Uma das maiores vantagens da especificação JDO é que ela é independente de base de dados; isto é, existem implementações bastante estáveis disponíveis para a maior parte das bases de dados mais importantes. Optou-se pela implementação JPOX, que é livre e que fornece a maior parte das funcionalidades adicionais das quais necessitamos. Além disso, a JPOX pode ser utilizada em conjunto com MySQL ou o Postgres, que são os bancos de dados relacionais livres mais difundidos no momento.

A especificação JDO define inúmeras facilidades que permitem persistência quase transparente do modelo de objetos. Dentre elas, podemos destacar:

- Demarcação de transações;
- persistência por transitividade;
- recuperação transparente (num dado grafo de objetos);
- possibilidades de buscar objetos satisfazendo determinado critério;
- outras facilidades de consulta.

JDO ainda fornece algumas outras facilidades, mas essas são as principais.

3 Organização do projeto

Desde o início, tínhamos conhecimento da complexidade do projeto, e sabíamos que a organização seria essencial para seu sucesso.

3.1 Responsabilidades

O projeto pode ser dividido em duas grandes etapas: a primeira, na qual nos dedicamos ao estudo das tecnologias e conhecimento do domínio da aplicação, e a segunda, na qual iniciamos a modelagem e a implementação do sistema.

A divisão do grupo em duas equipes ocorreu apenas na segunda etapa. Uma das equipes, formada pelo Carlos e pelo Pedro, ficou responsável por desenvolver a camada web e extensões, enquanto a outra, formada pelo Cleber e Giuliano, ficou encarregada de desenvolver o núcleo.

Embora o grupo tenha sido dividido a comunicação continuou sendo através de uma lista de e-mail única. Dessa forma, membros de uma equipe poderiam opinar nas discussões da outra equipe.

3.2 Andamento

Num primeiro momento, consideramos um projeto diferente do apresentado nesta monografia. Seria desenvolvido um conjunto de ferramentas para desenvolvimento de intranets empresariais. Esse projeto, todavia, mostrou-se ser por demais genérico, considerando a nossa falta de conhecimento do domínio de aplicação de tal sistema. Optamos, portanto, por desenvolver algo que nos parecesse mais viável dentro da atual situação do grupo. A escolha foi um sistema de Workflow Engines.

Inicialmente dedicamos o nosso tempo ao estudo de algumas ferramentas (Workflow Engines) disponíveis no mercado, para compreendermos melhor o domínio da aplicação. No decorrer do primeiro semestre, também estudamos as tecnologias e padrões de design que pretendíamos utilizar. Algum esforço de modelagem também foi feito mas sem que chegássemos, no entanto, em nada concreto.

A modelagem, que estava prevista para ser finalizada até o final de julho, só foi concluída no final de agosto (e alterada em alguns pontos durante a implementação). Embora a modelagem do sistema parecesse ser uma tarefa simples, foi uma das etapas mais trabalhosas e desgastante, e foi o início do atraso nos prazos que havíamos estipulado.

Até o começo de setembro o grupo permaneceu unido. O objetivo era que todos atingissem praticamente o mesmo nível de conhecimento das tecnologias e ferramentas analisadas, e que fôssemos capaz de decidir quais seriam as melhores opções. Optamos também por realizar toda a modelagem do sistema trabalhando juntos, para que todos opinassem em todas as partes do sistema, e que adquirissem uma visão geral do mesmo. Neste período, fazíamos reuniões mensais, para discutirmos o andamento do projeto, pensarmos em assuntos que não havia ficado muito claro nas discussões que mantinhamos através da lista de discussão por e-mails.

A divisão do grupo em duas equipes ocorreu no início da implementação. Optamos por realizar essa divisão para facilitar o processo de implementação, pois programávamos aos pares sempre que possível. Percebemos, também, que seria difícil a reunião de todos os integrantes do grupo para o desenvolvimento, em virtude da diferença de horário livre entre os participantes do grupo e devido às severas restrições de tempo impostas pelo curso. Neste período, as reuniões com a participação de todos foram mais freqüentes, ocorrendo a cada duas semanas, devido ao aumento da dificuldade no desenvolvimento.

Apesar de todo o esforço, não foi possível terminar o projeto até o início de dezembro. Tanto a programação do núcleo quanto das extensões estão bem adiantadas, e, no final de novembro, já estávamos realizando a integração das 2 partes.

3.3 Ferramentas utilizadas

Segue abaixo uma breve descrição das ferramentas que nos forneceram suporte no desenvolvimento do nosso sistema.

- **Eclipse**: é um ambiente de desenvolvimento (IDE) baseado em princípios de código aberto, que possui uma arquitetura extensível baseada no uso e desenvolvimento de *plugins*. A vantagem é o grande número de *plugins* disponíveis, que permite a integração com outras ferramentas disponíveis no mercado como o Tomcat, JBoss, CVS, Ant, entre outras.

- **CVS**: é um sistema de controle de versão, que nos permite manter as versões mais antigas dos arquivos armazenados (código fonte e outros arquivos) e mantém um log de quem, quando e como as mudanças ocorreram. O CVS opera não somente em um arquivo ou um diretório por vez, mas sim em coleções hierárquicas de diretórios. Ele ajuda a gerenciar as distribuições e a controlar a edição de arquivos por diversas pessoas, que ocorre concorrentemente.

  O CVS foi essencial para automatizar a integração do código que foi desenvolvido pelas equipes. Inicialmente tentamos utilizá-lo através do próprio Eclipse, porém encontramos alguns problemas no *plugin*, e optamos por executá-lo através da linha de comando.

- **Ant**: é uma ferramenta de código aberto, escrita em Java, desenvolvida pela "Apache Software Foundation", que tem como finalidade automatizar o processo de compilação de código. Esta ferramenta é a mais comumente usada para este fim em programas escritos em Java. O Ant é bastante portátil e simples de usar. Ele não depende da plataforma utilizada e nem do ambiente de desenvolvimento utilizado.

4 Experiência pessoal

4.1 Desafios e frustrações

Certamente o maior desafio foi trabalhar em um grande projeto por um longo período. Havia muitos detalhes a serem pensados, e o pouco conhecimento do domínio da aplicação foi o grande desafio inicial.

Outra dificuldade foi o aprendizado de tecnologias muito recentes. O conhecimento dos participantes do grupo não era homogêneo, o que aumentou muito o ritmo dos estudos com objetivo de nivelar este conhecimento.

Vencido esse desafio deparamos com a dificuldade de modelar o sistema. A descoberta da aplicação de Redes de Petri para representação de WF facilitou a modelagem, porém dificultou o desenho da interface do sistema.

Os desafios foram muitos, e a medida que foram aparecendo, foram sendo superados. Durante todo o projeto, também havia o desafio da divisão do tempo entre as responsabilidades do curso, o projeto e o estágio.

Particularmente não tive muitas frustrações. Desde o início do projeto, o principal objetivo do grupo era de aprender bastante, e este certamente foi atingido. Entre as frustrações, posso citar a impossibilidade de cumprir os prazos estipulados na proposta e a não conclusão da implementação.

4.2 Disciplinas do BCC mais relevantes

Muitas disciplinas influenciaram direta ou indiretamente com mais ou menos intensidade o desenvolvimento do projeto. Listarei abaixo apenas as disciplinas que foram mais importantes para sua concretização:

- **MAC110** - Introdução à Computação
  MAC122 - Princípios de Desenvolvimento de Algoritmos
  MAC323 - Estruturas de Dados

Certamente essas disciplinas forneceram a base para todo o curso. Através delas, aprendi as técnicas mais básicas de programação, utilização de estruturas de dados mais complexas e a pensar algoritmicamente na solução de um problema. Foi o primeiro contato com o conceito de algoritmo e a preocupação com sua eficiência.

- **MAC211** - Laboratório de Programação I
  MAC242 - Laboratório de Programação II

  Laboratório de Programação I possibilitou o primeiro contato com um projeto maior, e Laboratório de Programação II introduziu uma linguaguem orientada a objetos: Java. Ambas as disciplinas foram muito importantes para o aprendizado da programação em equipe, divisão das tarefas entre o grupo e cumprimento dos prazos de entrega.

  Em Laboratório de Programação I também aprendemos uma linguagem de *script*, que é extremamente útil para automatização de tarefas rotineiras. Embora o projeto tenha sido desenvolvido em Java, foram escritos *scripts* em Perl que possibilitaram o desenvolvimento do projeto na Rede Linux. Estes *scripts* eram responsáveis por fazer o *download* e configuração de programas como o Tomcat, JBoss, e o banco de dados Postgres, em diretórios temporários.

- **MAC316** - Conceitos Fundamentais de Linguagens de Programação
  MAC332 - Engenharia de Software

  Não poderia deixar de citar disciplinas que forneceram importantes conceitos de programação. Conceitos Fundamentais de Linguagens de Programação apresentou os diversos paradigmas de programação, destacando suas vantagens e desvantagens, e ensinando-nos a escolher o ideal de acordo com a aplicação a ser desenvolvida. Engenharia de Software forneceu o conhecimento necessário para implementar e desenvolver um projeto de forma organizada, além de métodos para testar o sistema.

- **MAC328** - Algoritmos em Grafos
  MAC414 - Linguagens Formais e Autômatos

  Os conceitos apresentados nesta disciplina foram muito úteis para a modelagem de um WF utilizando Redes de Petri.

- **MAC338** - Análise de Algoritmos

  Como o próprio nome já diz, nessa disciplina foi ensinada a análise de algoritmos, detectando seus pontos ineficientes e provando formalmente seu correto funcionamento. O conhecimento adquirido foi utilizado para analisar alguns trechos de código do sistema.

- **MAC426** - Sistemas de Bancos de Dados

  Embora não tenhamos modelado o sistema e escrito consultas SQL, os conceitos aprendidos através desta disciplina foram essenciais para a compreensão e utilização de uma implementação do JDO (*Java Data Object*).

Algumas disciplinas não cursadas, cujo conteúdo estudei sozinho, foram essenciais para o desenvolvimento de um projeto orientado a objetos. Entre as quais posso citar Programação Orientada a Objetos (MAC441) e Tópicos de Programação Orientada a Objetos (MAC413).

4.3 Interação com os membros da equipe

Este projeto não foi muito diferente aos quais estávamos acostumados a enfrentar, quanto a interação entre os membros da equipe. Todos já haviam trabalhado juntos em outros projetos, e a convivência entre os integrantes sempre foi boa.

Além de uma lista de mensagens e reuniões para discutir o projeto, formas de comunicação que já haviam sido utilizadas em outros projetos, usamos o CVS.

A grande vantagem de desenvolver um projeto em equipe é que em qualquer fase do projeto sempre há um membro motivado, que incentiva os outros a trabalharem.

A amizade e a liberdade entre todos os integrantes também foi uma desvantagem, pois muitas vezes foi difícil ordenar ou especificar uma tarefa para uma pessoa, além de não haver uma cobrança muito rígida entre os membros.

4.4 Considerações Finais

Desde o início da graduação em Bacharelado em Ciência da Computação, muitos, inclusive eu, questionaram a *utilidade* de algumas disciplinas muito teóricas, que pareciam não ser importantes.

O conhecimento adquirido nessas disciplinas e muitas outras, só foi notado quando precisei compreender novos conceitos que nunca havia estudado anteriormente. Nos primeiros cinco meses, o projeto resumiu-se em aprender novas tecnologias, e foi fascinante observar o crescimento intelectual que o IME ofereceu.

As disciplinas foram essenciais para aumentar nosso conhecimento dos fundamentos da Computação, desenvolver nossa capacidade criativa, estimular a auto-aprendizagem e o pensamento crítico.

Certamente essa formação sólida foi fundamental para o desenvolvimento deste projeto, além de ter colaborado para meu crescimento intelectual, profissional e pessoal.

5 Referências

Segue abaixo a lista das referências utilizadas no decorrer do desenvolvimento do projeto.

1. CRUZ, Tadeu. *E-Workflow*, CENADEM, 2001.
2. van der Aalst, Wil M.P. *The Application of Petri Nets to Workflow Management*, 1998, em: `http://tmitwww.tm.tue.nl/staff/wvdaalst/Petri_nets/petri_nets.html`.
3. ROOS, Robin.*M. Java Data Objects*, Addison Wesley, 2003.
4. MARINESCU, Floyd. *EJB Design Patterns*, John Wiley & Sons, Inc., 1999.
5. YODER, Joseph W. e JOHNSON, Ralph. *The Adaptive Object-Model Architectural Style*, 2002, em: `http://www.adaptiveobjectmodel.com`
6. *Data Access Object*, em `http://java.sun.com/blueprints/patterns/DAO.html`
7. ROMAN, Ed e AMBLER, Scott e JEWELL, Tyler. *Mastering Enterprise JavaBeans*, 2002.

8. ALUR, Deepak e CRUPI, John e MALKS, Dan. *Core J2EE Patterns*, Prentice Hall PTR, 2001.

9. GAMMA, Erich e HELM, Richard e JOHNSON, Ralph e VLISSIDES, John. *Design Patterns*, Addison Wesley, 1995.

10. DEITEL, Harvey M. e DEITEL, Paul J. *Java: Como Programar*, Bookman, 2003.

11. ECKEL, Bruce. *Thinking in Java*, Prentice Hall PTR, 2000.

12. PRESSMAN, Roger S. *Engenharia de Software*, McGraw-Hill, 2002.

13. FOWLER, Martin e SCOTT, Kendall. *UML Essencial*, Bookman, 2000.

14. *Java Servlet Technology*, em `http://java.sun.com/webservices/docs/1.0/tutorial/doc/Servlets.html`

15. *The Tomcat 4 Servlet/JSP Container*, em `http://jakarta.apache.org/tomcat/tomcat-4.1-doc/index.html`

16. *Velocity User Guide*, em `http://jakarta.apache.org/velocity/user-guide.html`

17. *Velocity Layout Servlet*, em `http://jakarta.apache.org/velocity/tools/view/layoutservlet.html`

18. *Struts User Guide*, em `http://jakarta.apache.org/struts/userGuide/index.html`

19. *Apache Ant 1.5.4 Manual*, em `http://ant.apache.org/manual/index.html`

20. *CVS Manual*, em `http://www.cvshome.org/docs/manual/cvs-1.11.10/cvs.html`

21. *Java Persistent Objects - JPOX*, em `http://jpox.sourceforge.net/`