Ciclo de Empréstimo do Colméia

André Guerra da Silva
Fernando Waitman
Ricardo Lazaro de Oliveira

Universidade de São Paulo
Instituto de Matemática e Estatística

MAC 499 - Trabalho de Formatura Supervisionado
Orientador: Prof. Dr. João Eduardo Ferreira

3 de dezembro de 2007

Sumário

Lista de Figuras

1 Introdução

1.1 O Sistema Colmeia

O Ciclo de Empréstimo integra o sistema Colméia, que é um sistema gerenciador de biblioteca. Ele vem sendo desenvolvido no IME-USP desde 2002 pelos alunos das disciplinas Laboratório de Programação eXtrema e Laboratório de Banco de Dados, tutelados pelos professores Eduardo Colli, Fábio Kon e João Eduardo Ferreira.

O Colméia visa a princípio suprir as necessidades básicas da Biblioteca do IME-USP. Uns dos focos do projeto são a gestão do acervo, movimentações - como aquisição, empréstimo, reserva e etc. - tanto de livros aperiódicos como também revistas assinadas pelo IME, teses e dissertações, cadastro de pessoas físicas e jurídicas que se relacionam com a biblioteca, além de geração de relatórios que facilitem a auditoria dessas movimentações.

Atualmente o Colméia já está em estado avançado de desenvolvimento. Seu banco de dados possui uma modelagem que já contempla a maioria dos dados a serem inseridos, e a aplicação em si já oferece muitas das funcionalidades a que o projeto se propõe. Cadastro de funcionários, usuários de diversos tipos e obras, bem como consulta destas são exemplos de funcionalidades já implementadas.

1.2 O Ciclo de Empréstimo

Para que o Colméia possa executar plenamente sua tarefa de gerenciamento de biblioteca, há algumas funcionalidades que devem obrigatoriamente ser implementadas. Algumas dessas funcionalidades já estavam implementadas, como foi dito na seção anterior. Entretanto, as funcionalidades que lidam com a movimentação dos exemplares, ou seja, empréstimo, devolução, reserva e renovação de empréstimo, ainda não haviam sido implementadas.

Atualmente essas movimentações são realizadas manualmente, com um controle baseado em fichas de papel que são colocadas nos livros e em anotações

nas fichas cadastrais de cada usuário. Esse método, apesar de funcionar e ser usado sem grandes mudanças desde que a biblioteca foi inaugurada, tem alguns defeitos. Auditorias, estatísticas e principalmente confiabilidade dos dados não são aspectos proporcionados facilmente por este sistema.

Assim, o projeto de implementação do módulo de Empréstimo no Colméia faz-se vital para que o sistema seja um gerenciador completo para a biblioteca. Com a informatização dessas funcionalidades de movimentação, espera-se que se tornem muito mais fácil o levantamento de dados sobre os livros e usuários, além de tornar essas movimentações mais ágeis no balcão.

2 Tecnologias de Apoio

A implementação do Ciclo de Empréstimo se apóia em diversas tecnologias e ferramentas que ajudam a torná-la mais flexível, coesa, e simples. Esta seção explica quais são, para que serviram no projeto e como foram utilizadas.

2.1 Álgebra de Processos

A NPDL utiliza como princípio a Álgebra de Processos, que trata da especificação e manipulação de termos de processo. Esses termos são representações algébricas dos grafos de processos, estes, por sua vez, são representações de modelos abstratos de negócio. Assim, com uma representação algébrica desses modelos, é possível manipulá-los por meio de de operadores estebelecidos, que serão enumerados a seguir.

Uma álgebra de processos é composta por um conjunto de símbolos de ações (ou eventos), um conjunto de operações e um conjunto de axiomas descrevendo as propriedades dos operadores. O conjunto de axiomas podem também especificar quando dois processos são considerados iguais.

Existem 2 operadores na *Álgebra de Processos Básica*, ou APB:

- o operador " + ", ou composição alternativa. Se os termos t_1 e t_2

representam os processos p_1 e p_2 respectivamente então o termo $t_1 + t_2$ representa o processo que executa p_1 ou p_2.

- o operador " $\bullet$ ", ou composição sequencial. Se os termos t_1 e t_2 representam os processos p_1 e p_2 respectivamente então o termo $t_1 \bullet t_2$ representa o processo que executa p_1 e em seguida p_2.

A NPDL utiliza ainda um terceiro operador, " $\|$ ", conhecido como entrelaçamento. O termo $p_1 \| p_2$ representa o processo que executa p_1 e p_2 em paralelo. Esse operador foi introduzido na *Algebra of Communicating Processes*, ou ACP. Outro conceito introduzido na ACP utilizado pela NPDL é o conceito de expressões recursivas. Todas as definições, regras e axiomas a desta subseção foram retirados da dissertação de mestrado de Kelly Rosa Braghetto, disponível na sua *página online*

2.2 NPDL

A *Navigation Plan Definition Language* (NPDL) é uma linguagem formal com embasamento algébrico que possibilita a definição de expressões que representam modelos de negócio. Seu principal objetivo é viabilizar o controle de processos de negócio em um banco de dados relacional, implementada como uma extensão da linguagem SQL. Ela é uma alternativa para a representação de *workflows*, e utiliza a álgebra de processos como arcabouço formal.

Assim como na álgebra de processos, processos em NPDL são definidos por expressões algébricas. Em NPDL a expressão algébrica de um processo é construída a partir do conjunto **A** de ações atômicas, de operadores NPDL e do conjunto **P**, sendo **P** o conjunto de todos os processos definidos. A NPDL contém os operadores mais comuns da álgebra de processos e também define operadores adicionais que modelam comportamentos frequentes em processos de *workflow*.

Uma ação atômica NPDL é equivalente a uma transação que respeita as propriedades ACID (atomicidade, consistência, independência e durabilidade) em um banco de dados: quando executada, ela é realizada na íntegra ou é totalmente desfeita em caso de problemas na execução.

Na NPDL, existem ainda dois elementos adicionais que possuem a característica de uma ação atômica: a regra e a função.

Uma regra representa uma ação atômica que retorna um valor booleano - verdadeiro ou falso. Uma regra na NPDL pode ser entendida como uma condição, avaliada em tempo de execução, para a execução de um trecho do fluxo.

Uma função representa uma ação atômica que retorna um número inteiro, positivo e não nulo. Na NPDL, uma função torna possível, juntamente a um operador específico, a repetição de um trecho do fluxo um número de vezes definido em tempo de execução.

A NPDL reconhece diversos operadores. Os de principal importância para a definição do Ciclo de Empréstimo foram os seguintes:

- Composição Seqüencial (operador "."): a expressão A . B significa que inicialmente somente o termo A está habilitado para execução. O termo B será habilitado para execução somente após o fim da execução do termo A;
- Composição Alternativa (operador "+"): a expressão A + B significa que inicialmente ambos termos A e B estarão habilitados para execução, porém apenas um deles será executado;
- Composição Paralela (operador "||"): a expressão A || B significa que inicialmente ambos termos estão habilitados para execução, e que os eles poderão ser executados paralelamente ou em uma ordem qualquer entre si;
- Execução Condicional (operador "%r", sendo r é uma regra): a expressão %R1 A significa que o termo A será habilitado para execução se o valor de retorno da regra R1 for verdadeiro;
- Execução Condicional Negativa (operador "%!r", sendo r é uma regra): a expressão %!R1 A significa que o termo A será habilitado para execução se o valor de retorno da regra R1 for falso.

Os comportamentos dos operadores de execução condicional não são representados em álgebra de processos, entretanto os seus comportamentos podem ser representados com alguma combinação de operadores da álgebra de processos.

Para a criação, alteração e exclusão de processos, ações, regras e funções NPDL, a linguagem possui características semelhantes a comandos SQL. Esses comandos podem ser processados por meio de uma das aplicações web da ferramenta NPWS, apresentada adiante nesta monografia.

A criação de um processo chamado P1 é realizada por meio do comando "CREATE PROCESS P1;". Para a criação de uma regra R1, temos o comando "CREATE RULE R1;", assim seguindo o mesmo padrão para a criação de ações ("CREATE ACTION") e funções ("CREATE RULE").

Para que um processo P1 (previamente criado) seja definido, a sintaxe é "SET P1=[EXPRESSÃO];". O termo [EXPRESSÃO] pode ser qualquer expressão válida na linguagem NPDL ([EXPRESSÃO] deve ser uma string formada por nomes de processos, ações, regras e funções anteriormenmte criadas, além de operadores que definem o controle do fluxo). Dessa forma, é possível realizar a criação, definição e inserção dos processos desejados no banco de dados da ferramenta.

Mais informações sobre a linguagem NPDL podem ser encontradas na dissertação de mestrado da aluna Kelly Rosa Braghetto, disponível em: *http://www.vision.ime.usp.br/∼kellyrb/nptool/krbraghetto_dissertacao.pdf*.

2.3 NPTool

A *Navigation Plan Tool* (NPTool) - é uma biblioteca de funções que provê mecanismos para a manutenção de ações e processos em bancos de dados relacionais e para o controle de instanciação e execução desses processos. Ela é composta por três serviços: o Interpretador NPDL, o Serviço de Instanciação de Processos e o Serviço Monitor de Execução de Instâncias de Processos.

O interpretador realiza a análise léxica, sintáxica e semântica de expressões NPDL. Caso identifique um comando como válido, esse é guardado em um banco de dados. Já o serviço de instanciação possibilita a criação

de instâncias de um processo. Instância no contexto da NPTool tem o mesmo sentido que em orientação a objeto em geral, isto é, um processo é uma descrição geral de um modelo de negócios - como uma classe - e uma instância desse processo representa uma requisição deste processo. Assim, com os serviços disponibilizados por esta ferramenta, é possível realizar o controle de fluxo do seu modelo de negócio de maneira independente do aplicativo em si.

A Navigation Plan Tool foi desenvolvida em Java e interage com qualquer banco de dados compatível com a definição padrão da linguagem SQL, tornando assim a NPDL uma extensão independente de sistema gerenciador de banco de dados.

2.4 NPWS

A *Navigation Plan Web Services* (NPWS) é um conjunto de serviços web que encapsulam as funcionalidades da ferramenta denominada NPTool, descrita anteriormente.

A principal característica da ferramenta NPWS é disponibilizar serviços web que permitem a definição, a instanciação e o controle de execução de processos de negócio. Ao oferecer esses serviços, ela torna possível que qualquer aplicação implementada em uma linguagem que possibilite comunicação web e leitura XML - o que quase todas as linguagens modernas possibilitam - possa utilizar o controle de fluxo proporcionado pela NPTool.

Mais informações sobre a ferramenta NPWS podem ser encontradas no documento "NPWS - Serviços Web para a NavigationPlanTool", disponível em: *http://www.ime.usp.br/∼chui/artigos/sicpg2007.pdf*.

2.5 Velocity

O Projeto Apache Velocity é um engine baseado na linguagem Java para criação de templates em páginas web, permitindo que se use uma linguagem para referenciar objetos definidos em Java. Assim torna-se possível a manipulação de estruturas complexas, isto é, qualquer objeto não-primitivo,

dentro do código da página. Seu principal objetivo é facilitar a separação entre aplicação e apresentação, ou seja, a implementação do padrão MVC (model-view-controller).

O projeto Colmeia usa o Velocity junto com o framework Struts (mais informações na seção 2.6). Todo o mecanismo que liga o Velocity ao Struts e tudo isso à aplicação em si é descrito na seção 6.4.

2.6 Struts

O Apache Struts é um framework para desenvolvimento de aplicações web Java EE. Ele estende a Java Servlets API de maneira a incentivar o uso do padrão MVC (model-view-controller).

O framework possibilita separar de maneira bastante intuitiva o modelo (a aplicação em si), da visão (no caso deste projeto as páginas Velocity), do controlador. O Struts provê o controlador com o ActionServlet, mas deixa que o modelo seja implementado pelo desenvolvedor. O arquivo struts-config.xml liga esses três elementos num arquivo central de configuração. Assim, no projeto Colmeia, uma requisição de uma página Velocity comunica-se com o framework Struts, que por sua vez manipula as classes do modelo, que estão escritas em Java, de maneira a implementar de maneira transparente o padrão MVC.

2.7 Tomcat

O Tomcat é um servidor de aplicações Java para web. É distribuído como software livre e desenvolvido como código aberto dentro do conceituado projeto Apache Jakarta e oficialmente endossado pela Sun como a Implementação de Referência (RI) para as tecnologias Java Servlet e JavaServer Pages (JSP). O Tomcat é o servidor que atualmente roda o sistema Colméia, promovendo um ambiente estável, robusto e de fácil escalabilidade para acomadar a produção do Colméia.

2.8 Jboss

JBoss é um servidor de aplicação de código fonte aberto baseado na plataforma J2EE e implementado completamente na linguagem de programação Java. Como é baseada em Java, pode ser usado em qualquer Sistema Operacional que suporte Java. Sua utilização no projeto Colméia é rodar o NPWS, que é necessário para encapsular como serviços web as funcionalidades do gerencidor de processos NPTOOL.

2.9 Subversion

Subversion (também conhecido por SVN, o nome da sua ferramenta de linha de comando) é um sistema de controle de versão desenhado especificamente para ser um substituto moderno do CVS, devido ao fato de este último possuir algumas deficiências que dificultam a sua utilização.

Com o controle de versões podemos gravar o histórico de todos os arquivos de um projeto. Isso é extremamente útil quando, eventualmente, verifica-se a existência de um problema no projeto gerado por alguma alteração realizada anteriormente no código de um arquivo. Nesse caso, o controle de versões permite restaurar facilmente as versões antigas para ver exatamente qual mudança gerou tal problema.

Ele também permite a criação de contas de acesso, de forma que repositórios possam ser acessados remotamente de forma segura por pessoas autorizadas (desde que essas tenham o programa Subversion instalado em seus computadores).

Uma característica importante do programa é que, diferentemente do simples armazenamento de todas as versões de todos os arquivos que criamos, o Subversion armazena todas as versões de uma forma inteligente, guardando apenas as diferenças entre as versões, o que deve ser muito mais vantajoso.

Com a utilização do Subversion, é possível realizar o controle de versão para todo tipo de arquivo. Não apenas para arquivos de codigo fonte, mas para qualquer tipo de arquivo (inclusive arquivos binarios).

As principais funcionalidades oferecidas pela ferramenta são:

- svn update: Atualiza a cópia local do repositório, caso tenha ocorrido alguma mudança.
- svn add [nome_arquivo]: Adiciona o arquivo [nome_arquivo] ao repositório.
- svn log: Exibe as alterações feitas ao repositório.
- svn commit: Atualiza o repositório com todas as alterações realizadas localmente.

Por meio de um *plug-in*, pode-se contar com todas as funcionalidades do programa Subversion no IDE Eclipse. Dessa forma, é possível visualizar de forma prática as situações de todos os arquivos do seu projeto se comparadas com o repositório, além de realizar atualizações e adicionar arquivos a ele facilmente.

2.10 O IDE Eclipse

IDE, do inglês *Integrated Development Environment*, é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo.

Geralmente os IDEs facilitam a técnica de RAD (*Rapid Application Development*), que visa a maior produtividade dos desenvolvedores.

Em geral, algumas das características encontradas nas IDEs são:

- Editor: edita o código-fonte do programa escrito na(s) linguagem(ns) suportada(s) pela IDE;
- Compilador: compila o código-fonte do programa, editado em uma linguagem específica e a transforma em linguagem de máquina;
- *Linker*: liga os vários "pedaços"de código-fonte, compilados em linguagem de máquina, em um programa executável que pode ser executado em um computador ou outro dispositivo computacional;

- Depurador: auxilia no processo de encontrar e corrigir erros no código-fonte do programa;
- Distribuição (*deploy*): auxilia no processo de criação do instalador do software, ou outra forma de distribuição desse;
- Testes Automatizados: facilita a criação de testes automatizados no software que auxiliam na análise do impacto das alterações no código-fonte;

O Eclipse é um ambiente de desenvolvimento integrado de código aberto para a construção de projetos de software. Trata-se do IDE Java mais utilizado no mundo.

Ele possui como características marcantes o uso da SWT e não do Swing como biblioteca gráfica, a forte orientação ao desenvolvimento baseado em *plug-ins* e o amplo suporte ao desenvolvedor com centenas de *plug-ins* que procuram atender as diferentes necessidades de diferentes programadores.

Projetos Eclipse fornecem arcabouços e ferramentas para todas as etapas do desenvolvimento de software, incluindo modelagem, desenvolvimento, ferramentas para *deployment*, geração de relatórios, manipulação de dados, testes e *profiling*. Essas ferramentas e arcabouços tem seu foco no desenvolvimento de serviços e aplicações web JEE, mas também fornecem suporte para outras linguagens como C/C++, PHP, entre outras.

2.11 PostgreSQL

PostgreSQL é um sistema gerenciador de banco de dados objeto relacional (SGBDOR), desenvolvido como projeto software livre, e que conta com recursos como:

- consultas complexas
- chaves estrangeiras
- integridade transacional

- controle de concorrência multi-versão

O PostgreSQL é o gerenciador de banco de dados que atualmente o Colméia utiliza, e este conta com uma grande quantidade de stored procedures escritas em plsql para a realização não apenas das persistências mas também das consistências de dados. Este também é o banco de dados usado pelas ferramentas NPWS e NPTOOL para o projeto Colméia, nunca sendo testadas com outro SGBD.

3 Caso de uso: A NPDL no módulo de empréstimo

O Empréstimo foi implementado utilizando-se de tecnologias que possibilitaram a definição de um processo de negócio que representasse esse comportamento, além do controle e execução deste processo. Este caminho foi tomado por se tratar de um módulo que apresenta um comportamento mais dinâmico do que os módulos Pessoa e Acervo do Colméia (esses últimos puramente cadastrais, por isso podem ser fácilmente representados por um modelo entidade-relacionamento), .

Estas tecnologias são baseadas na linguagem NPDL, portanto, o módulo de Empréstimo usa extensivamente esta linguagem. Isso porque, além do uso da linguagem para a definição do seu modelo de negócios, módulo de empréstimo também utiliza os modelos criados por meio da NPDL para o controle de execução dos processos.

Chamamos de *Ciclo de Empréstimo* a representação do modelo de negócios criada a partir da NPDL para controlar as atividades existentes no módulo de empréstimo do Colméia.

Utilizando-se de um arcabouço Java para suporte NPWS, a implementação da lógica de negócios do Ciclo de Empréstimo no sistema Colméia foi totalmente orientada pela definição dos fluxos NPDL. Para cada passo (ação, regra e função) NPDL, temos uma classe Java que implementa seu valor semântico. Por meio de uma outra classe, é realizada a vinculação de cada

passo NPDL com seus valores semânticos (classes Java). Uma outra classe java cuida da execução do processo NPDL automaticamente, isto é, cuida para que o código que implementa o valor semântico de cada passo seja executado quando este for habilitado para execução.

Essas características trouxeram uma série de vantagens ao sistema, entre elas:

- Reaproveitamento do código: um mesmo passo pode fazer parte de dois ou mais ciclos e, dessa forma, o código da classe que o implementa será reaproveitado;
- Encapsulamento do código: o desenvolvimento quase sempre é voltado para as classes que implementam os valores semânticos dos passos NPDL. Isso facilita a divisão de tarefas dentro de uma equipe;
- Facilitação da manutenção do código: na ocorrência de uma mudança em algum passo ou subprocesso do modelo de negócios, a estrutura existente facilita a sua manutenção;
- Clareza do código: a estrutura existente torna o código mais claro, uma vez que cada passo do fluxo está implementado em uma classe.

Além de todas essas características, a estrutura criada promove a separação explícita entre o modelo de negócios e a implementação, o que pode ser um grande facilitador em alguns tipos de mudanças.

4 Modelagem

O Modelo de negócios do Ciclo de Empréstimo é constituído pelos seguintes processos:

- P_USUARIO: realiza a verificação dos dados de um usuário que deseja realizar um empréstimo;

- P_EXEMPLAR: realiza a verificação dos dados de um exemplar que está sendo requisitado para empréstimo;
- P_EMPRESTIMO: fluxo controlador da rotina de empréstimo de um exemplar;
- P_USUARIO_FLEX: realiza a verificação dos dados de um usuário, exceto as informações sobre cota e atraso, que deseja realizar um empréstimo;
- P_EMPRESTIMO_FLEX: fluxo controlador da rotina de empréstimo flexível. Esse fluxo permite a realização do empréstimo para um usuário que não possua cota ou que esteja em atraso com a devolução de algum exemplar;
- P_DEVOLUCAO: fluxo controlador da rotina de devolução de um exemplar;
- P_USUARIO_RENOV: realiza a verificação dos dados de um usuário que deseja realizar uma renovação;
- P_RENOVACAO: fluxo controlador da rotina de renovação do empréstimo de um exemplar;
- P_USUARIO_RESERVA: realiza a verificação dos dados de um usuário, exceto a informação sobre cota, que deseja realizar um empréstimo;
- P_RESERVA_DE_OBRA: realiza a verificação dos dados de uma obra do acervo que está sendo requisitada para reserva;
- P_RESERVA: fluxo controlador da rotina de reserva de uma obra.

Pelo fato de a NPDL não aceitar encadeamento de regras na expressão que define um mesmo processo, foram necessários vários subprocessos para as definições dos processos do Ciclo de Empréstimo.

Abaixo, seguem os fluxogramas que representam cada processo citado, seguidos do significado de cada passo que os compõem e das suas expressões em NPDL:

4.1 Processo P_USUARIO

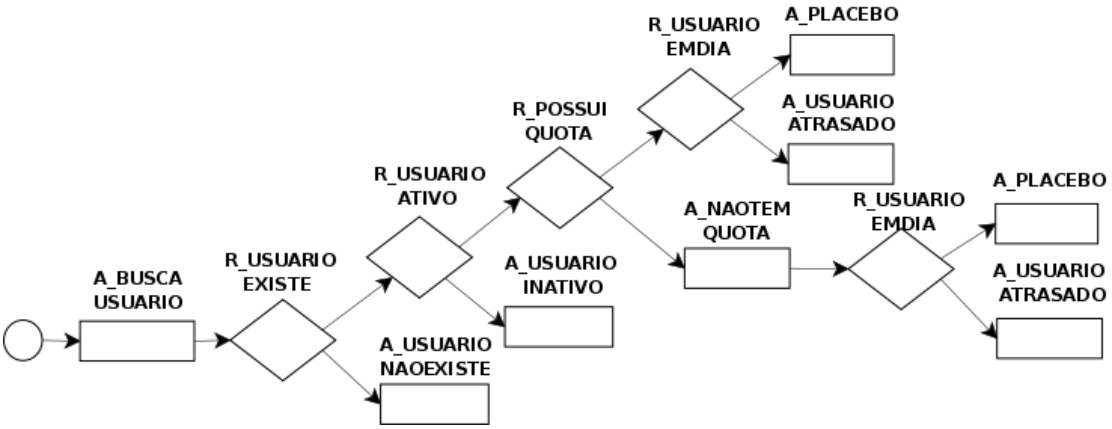

Figura 1: Processo P_USUARIO

P_USUARIO = A_USUARIO_BUSCA. (%!R_USUARIO_EXISTE A_USUARIO_NAOEXISTE + %R_USUARIO_EXISTE P_RULE_USUARIO_ATIVO)

P_RULE_USUARIO_ATIVO = %!R_USUARIO_ATIVO A_USUARIO_INATIVO + %R_USUARIO_ATIVO P_RULE_USUARIO_TEMQUOTA

P_RULE_USUARIO_TEMQUOTA = (%!R_USUARIO_TEMQUOTA A_USUARIO_NAOTEMQUOTA. P_RULE_USUARIO_EMDIA + %R_USUARIO_TEMQUOTA P_RULE_USUARIO_EMDIA)

P_RULE_USUARIO_EMDIA = (%!R_USUARIO_EMDIA A_USUARIO_ATRASADO + %R_USUARIO_EMDIA PLACEBO)

Ações:

- A_USUARIO_BUSCA: busca um usuário no banco de dados baseado no tipo e número do documento;

- A_USUARIO_NAOEXISTE: rotina de erro - usuário inexistente;
- A_USUARIO_INATIVO: rotina de erro - usuário inativo;
- A_USUARIO_NAOTEMQUOTA: rotina de erro - usuário não possui cota para empréstimo;
- A_USUARIO_ATRASADO: rotina de erro - usuário não está em dia com a devolução dos seus empréstimos;
- PLACEBO: não possui valor semântico. Criada apenas para que o fluxo possa continuar caso o processo não encontre erros.

Regras:

- R_USUARIO_EXISTE: retorna true caso o usuário exista;
- R_USUARIO_ATIVO: retorna true caso o usuário esteja ativo;
- R_USUARIO_TEMQUOTA: retorna true caso o usuário possua cota para empréstimo;
- R_USUARIO_EMDIA: retorna true caso o usuário esteja em dia com a devolução dos seus empréstimos.

4.2 Processo P_EXEMPLAR

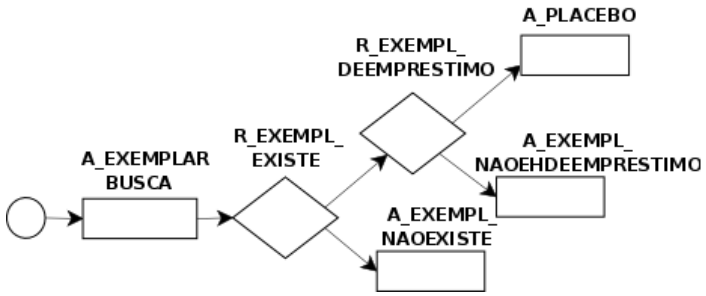

Figura 2: Processo P_EXEMPLAR

P_EXEMPLAR = A_EXEMPL_BUSCA. (%!R_EXEMPL_EXISTE
A_EXEMPL_NAOEXISTE + %R_EXEMPL_EXISTE
P_RULE_EXEMPL_DEEMPRESTIMO)

P_RULE_EXEMPL_DEEMPRESTIMO =
%!R_EXEMPL_DEEMPRESTIMO
A_EXEMPL_NAOEHDEEMPRESTIMO +
%R_EXEMPL_DEEMPRESTIMO PLACEBO

Ações:

- A_EXEMPL_BUSCA: busca um exemplar no banco de dados baseado no código de barras;
- A_EXEMPL_NAOEXISTE: rotina de erro - exemplar inexistente;
- A_EXEMPL_NAOEHDEEMPRESTIMO: rotina de erro - exemplar não é de empréstimo;
- PLACEBO: mesma do P_USUARIO.

Regras:

- R_EXEMPL_EXISTE: retorna true caso o exemplar exista;
- R_EXEMPL_DEEMPRESTIMO: retorna true caso o exemplar seja de empréstimo.

4.3 Processo P_EMPRESTIMO

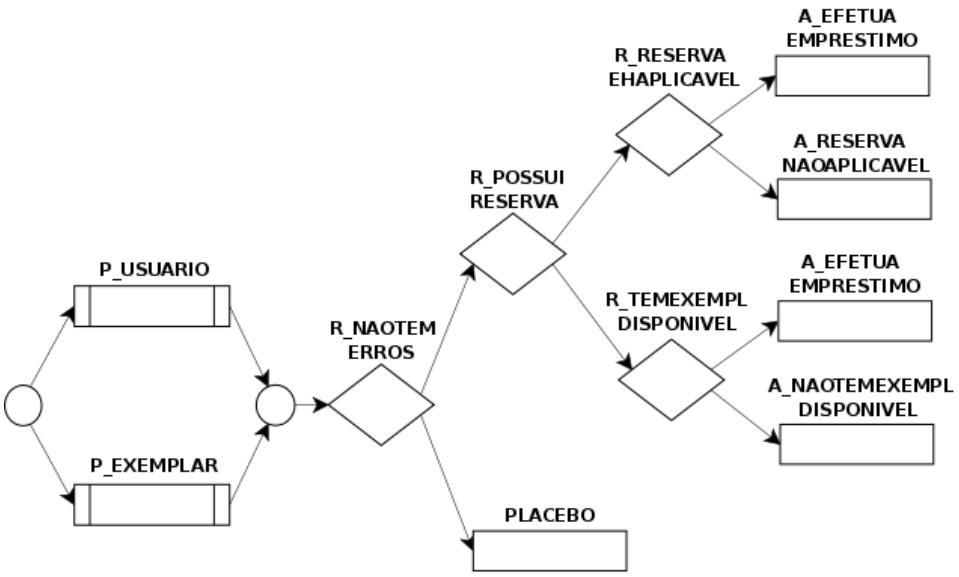

Figura 3: Processo P_EMPRESTIMO

Ações:

- A_EMP_RESERVANAOAPLICAVEL: rotina de erro - a reserva que o usuário possui não é aplicável para a realização do empréstimo nesse momento;
- A_EMP_NAOTEMEXEMPLDISPONIVEL: rotina de erro - não existe exemplar disponível para empréstimo;
- A_EMP_EFETUAEMPRESTIMO: registra o empréstimo no banco de dados;

- PLACEBO: mesma do P_USUARIO.

Regras:

- R_EMP_NAOTEMERROS: retorna true caso o processo não tenha gerado nenhum erro até o momento;
- R_EMP_POSSUIRESERVA: retorna true caso o usuário possua reserva para empréstimo da obra;
- R_EMP_RESERVAEHAPLICAVEL: retorna true caso a reserva que o usuário possui seja aplicável para a realização do empréstimo nesse momento;
- R_EMP_TEMEXEMPLDISPONIVEL: retorna true caso exista exemplar disponível para empréstimo no momento.

Expressão NPDL:

P_EMPRESTIMO = (P_USUARIO || P_EXEMPLAR).
(%R_EMP_NAOTEMERROS P_RULE_EMP_POSSUIRESERVA +
%!R_EMP_NAOTEMERROS PLACEBO)

P_RULE_EMP_POSSUIRESERVA = %R_EMP_POSSUIRESERVA
P_RULE_EMP_RESERVAEHAPLICAVEL +
%!R_EMP_POSSUIRESERVA
P_RULE_EMP_TEMEXEMPLDISPONIVEL

P_RULE_EMP_RESERVAEHAPLICAVEL =
%R_EMP_RESERVAEHAPLICAVEL A_EMP_EFETUAEMPRESTIMO
+ %!R_EMP_RESERVAEHAPLICAVEL
A_EMP_RESERVANAOAPLICAVEL

P_RULE_EMP_TEMEXEMPLDISPONIVEL =
%R_EMP_TEMEXEMPLDISPONIVEL
A_EMP_EFETUAEMPRESTIMO +
%!R_EMP_TEMEXEMPLDISPONIVEL
A_EMP_NAOTEMEXEMPLDISPONIVEL

4.4 Processo P_USUARIO_FLEX

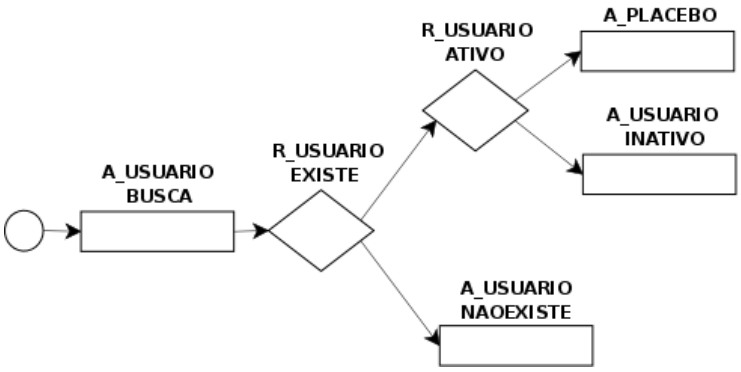

Figura 4: Processo P_USUARIO_FLEX

Todas as ações, regras e subprocessos utilizados já definidos no P_USUARIO.

Expressão NPDL:

P_USUARIO_FLEX = A_USUARIO_BUSCA. (%!R_USUARIO_EXISTE
A_USUARIO_NAOEXISTE + %R_USUARIO_EXISTE
P_RULE_USUFLEX_ATIVO)

P_RULE_USUFLEX_ATIVO = (%!R_USUARIO_ATIVO
A_USUARIO_INATIVO + %R_USUARIO_ATIVO PLACEBO)

4.5 Processo P_EMPRESTIMO_FLEX

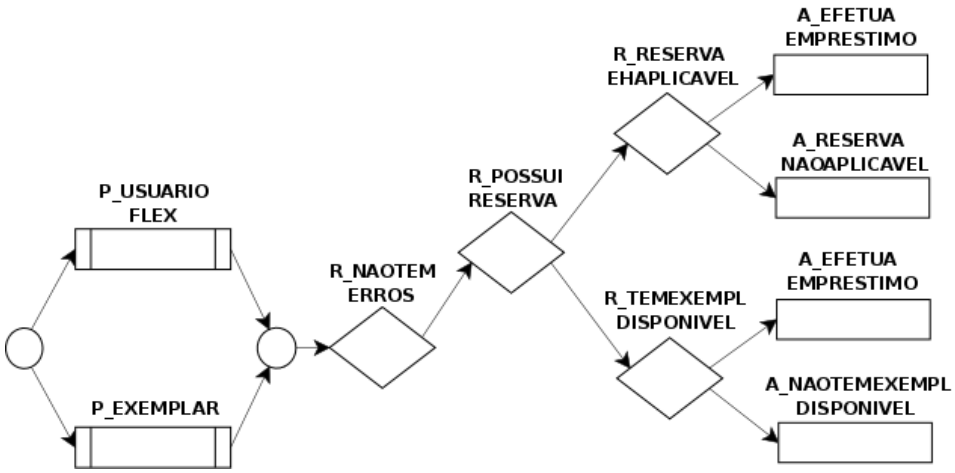

Figura 5: Processo P_EMPRESTIMO_FLEX

Todas as ações, regras e subprocessos utilizados já definidos no P_EMPRESTIMO.

Expressão NPDL:

P_EMPRESTIMO_FLEX = P_USUARIO_FLEX || P_EXEMPLAR.
(%R_EMP_NAOTEMERROS P_RULE_EMP_POSSUIRESERVA +
%!R_EMP_NAOTEMERROS PLACEBO)

4.6 Processo P_DEVOLUCAO

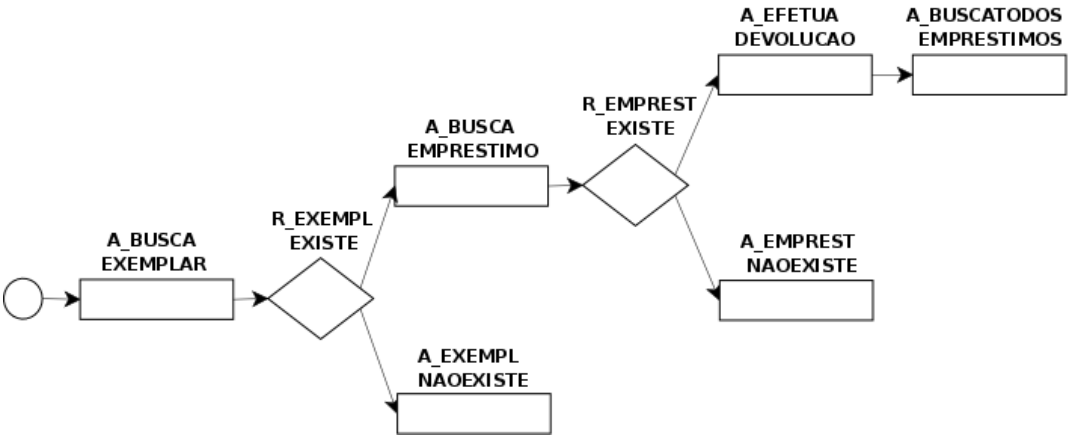

Figura 6: Processo P_DEVOLUCAO

Ações:

- A_DEV_BUSCAEXEMPLAR: busca um exemplar no banco de dados baseado no código de barras;
- A_DEV_BUSCAEMPRESTIMO: busca um empréstimo no banco de dados;
- A_DEV_BUSCATODOSEMPRESTIMOS: busca todos os empréstimos vinculados a um usuário no banco de dados;
- A_DEV_EXEMPLNAOEXISTE: rotina de erro - exemplar inexistente;
- A_DEV_EMPRESTNAOEXISTE: rotina de erro - empréstimo inexistente;
- A_DEV_EFETUADEVOLUCAO: registra a devolução no banco de dados.

Regras:

- R_DEV_EXEMPLEXISTE: retorna true caso o exemplar exista;
- R_DEV_EMPRESTEXISTE: retorna true caso o empréstimo exista.

Expressão NPDL:

P_DEVOLUCAO = A_DEV_BUSCAEXEMPLAR.
(%!R_DEV_EXEMPLEXISTE A_DEV_EXEMPLNAOEXISTE +
%R_DEV_EXEMPLEXISTE P_RULE_DEV_EMPRESTEXISTE)

P_RULE_DEV_EMPRESTEXISTE = A_DEV_BUSCAEMPRESTIMO.
(%!R_DEV_EMPRESTEXISTE A_DEV_EMPRESTNAOEXISTE +
%R_DEV_EMPRESTEXISTE A_DEV_EFETUADEVOLUCAO.
A_DEV_BUSCATODOSEMPRESTIMOS)

4.7 Processo P_USUARIO_RENOV

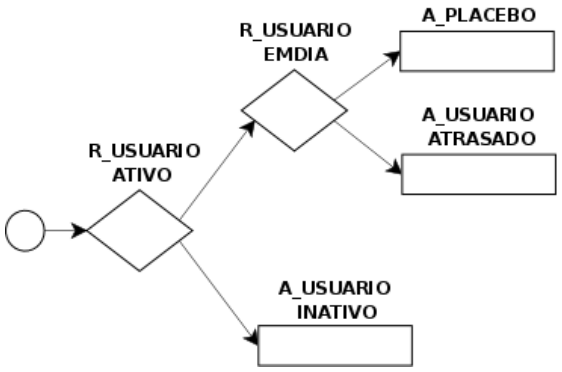

Figura 7: Processo P_USUARIO_RENOV

Todas as ações, regras e subprocessos utilizados já definidos no P_USUARIO.

Expressão NPDL:

P_USUARIO_RENOV = %R_USUARIO_ATIVO
P_RULE_USUARIO_EMDIA + %!R_USUARIO_ATIVO
A_USUARIO_INATIVO

4.8 Processo P_RENOVACAO

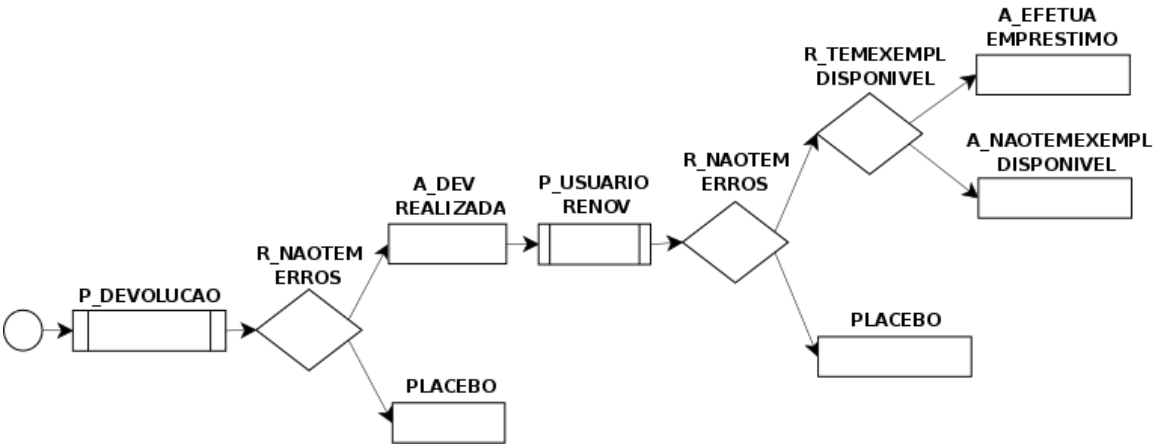

Figura 8: Processo P_RENOVACAO

Ações:

- A_RENOV_DEVREALIZADA: indica que a devolução do exemplar já foi registrada no banco de dados, independentemente do restante da operação de renovação;
- PLACEBO: mesma do P_USUARIO.

Regras:

- R_EMP_NAOTEMERROS: mesma do P_EMPRESTIMO;
- R_DEV_EMPRESTEXISTE: retorna true caso o empréstimo exista.

Expressão NPDL:

P_RENOVACAO = P_DEVOLUCAO. (%R_EMP_NAOTEMERROS
(A_RENOV_DEVREALIZADA. P_USUARIO_RENOV.
P_RULE_RENOV_NAOTEMERROS) +
%!R_EMP_NAOTEMERROS PLACEBO)

P_RULE_RENOV_NAOTEMERROS = (%R_EMP_NAOTEMERROS
P_RULE_EMP_TEMEXEMPLDISPONIVEL +
%!R_EMP_NAOTEMERROS PLACEBO)

4.9 Processo P_USUARIO_RES

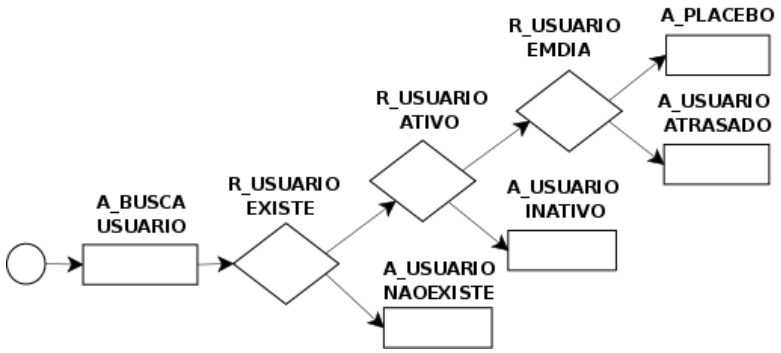

Figura 9: Processo P_USUARIO_RES

P_USUARIO_RES = A_USUARIO_BUSCA. (%!R_USUARIO_EXISTE A_USUARIO_NAOEXISTE + %R_USUARIO_EXISTE P_RULE_USUARIO_ATIVO)

Todas as ações, regras e subprocessos utilizados já definidos no P_USUARIO.

4.10 Processo P_RESERVA_DE_OBRA

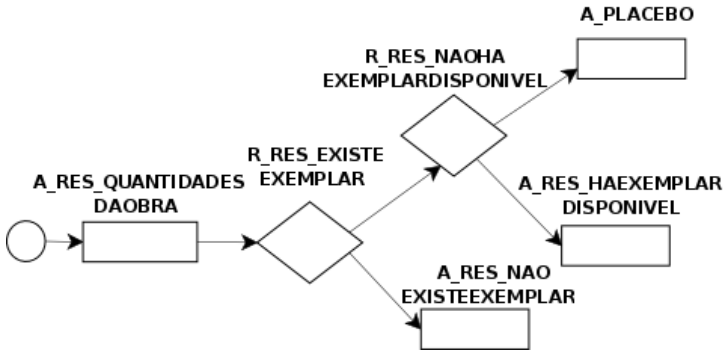

Figura 10: Processo P_RESERVA_DE_OBRA

P_RESERVA_DE_OBRA = A_RES_QUANTIDADESDAOBRA.
%R_RES_EXISTEEXEMPLAR P_DISPONIBILIDADE +
%!R_RES_EXISTEEXEMPLAR A_RES_NAOEXISTEEXEMPLAR)

P_DISPONIBILIDADE = %R_RES_NAOHAEXEMPLARDISPONIVEL
PLACEBO + %!R_RES_NAOHAEXEMPLARDISPONIVEL
A_RES_HAEXEMPLARDISPONIVEL

Ações:

- A_RES_QUANTIDADESDAOBRA: busca as informações sobre quantidade de exemplares de empréstimo da obra e quantidade de exemplares disponíveis para empréstimo no momento;
- A_RES_HAEXEMPLARDISPONIVEL: rotina de erro - existe exemplar disponível para empréstimo na biblioteca;
- A_RES_NAOEXISTEEXEMPLAR: rotina de erro - não existe exemplar de empréstimo da obra desejada;
- PLACEBO: mesma do P_USUARIO.

Regras:

- R_RES_EXISTEEXEMPLAR: retorna true caso exista pelo menos um exemplar de empréstimo da obra;
- R_RES_NAOHAEXEMPLARDISPONIVEL: retorna true caso não exista exemplar disponível para empréstimo na biblioteca.

4.11 Processo P_RESERVA

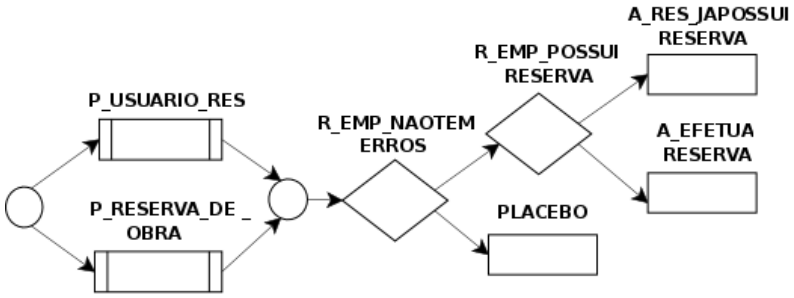

Figura 11: Processo P_RESERVA

P_RESERVA = (P_USUARIO_RES || P_RESERVA_DE_OBRA).
(%R_EMP_NAOTEMERROS P_RULE_RES_POSSUIRESERVA +
%! R_EMP_NAOTEMERROS PLACEBO)

P_RULE_RES_POSSUIRESERVA = %!R_EMP_POSSUIRESERVA
A_RES_EFETUARESERVA + %R_EMP_POSSUIRESERVA
A_RES_JAPOSSUIRESERVA

Ações:

- A_RES_JAPOSSUIRESERVA: rotina de erro - reserva duplicada;
- A_RES_EFETUARESERVA: registra a reserva no banco de dados;

- PLACEBO: mesma do P_USUARIO.

Regras:

- R_EMP_NAOTEMERROS: mesma do P_EMPRESTIMO;
- R_EMP_POSSUIRESERVA: mesma do P_EMPRESTIMO.

5 Preparação do Ambiente

5.1 Instalação das ferramentas NPWS e NPTool

Todos os procedimentos necessários para a instalação das ferramentas NPWS e NPTool, incluindo detalhes sobre instalação e configuração das ferramentas relacionadas (servidor de aplicação JBoss, banco de dados Postgres, etc..) podem ser encontrados no documento manual_servidor.pdf, que se encontra no endereço *http://www.linux.ime.usp.br/∼chui/NPWS/manuais/*.

5.2 Carregamento dos fluxos

Para que o módulo de Empréstimo funcione adequadamente, devem ser incluídos no banco de dados NPDL todos os processos listados na seção **Modelagem**.

Esses procedimentos foram realizados por meio de duas aplicações desenvolvidas com a tecnologia Java Servlet e com a linguagem HTML. Acessíveis como quaisquer páginas dinâmicas encontradas na Internet, as aplicações funcionam totalmente com chamadas aos serviços web da NPWS e são controladas manualmente por um usuário, mas apresentam diferentes escopos. Essas aplicações foram desenvolvidas pelo criador da NPWS (*http://www.linux.ime.usp.br/∼chui*).

A primeira aplicação é voltada exclusivamente à execução de comandos NPDL, de forma a possibilitar a definição de passos e processos. O usuário

digita os comandos em um local da tela e solicita a execução, recebendo os resultados (informações sobre a execução). Isso destaca o fato de o NPWS estar disponível para qualquer aplicação capaz de se conectar com os seus serviços web.

A segunda aplicação é uma área de controle para processos e passos definidos no NPWS. O usuário pode visualizar, em uma interface amigável, as listas de todos os processos, passos e instâncias, bem como filtrar o que deseja ver em cada lista de acordo com as opções fornecidas pela NPTool e obter diversas informações sobre cada item. O aspecto mais importante é a instanciação de processos e a execução das instâncias: o usuário tem à disposição os possíveis passos de uma instância de processo e escolhe o que deseja executar. Quando o passo é finalizado, as novas opções são exibidas e assim sucessivamente, até que a execução da instância termine. Dessa forma, essa aplicação pode ser utilizada para instanciar e controlar a execução de processos existentes.

A primeira coisa a ser feita é a criação de todos os passos listados na seção **Modelagem**. As ações devem ser criadas por meio do comando "CREATE ACTION [nome_da_ação]"e as regras por meio do comando "CREATE RULE [nome_da_regra]".

Depois disso, devem ser criados todos os processos e subprocessos existentes por meio do comando "CREATE PROCESS [nome_do_processo]".

Finalmente, os processos criados devem ser definidos por meio do comando "SET [nome_do_processo] = [EXPRESSÃO]", atentando para que os subprocessos sejam definidos antes dos processos que os referenciam.

5.3 Instalação do Colméia

O Colméia é um projeto atualmente em desenvolvimento, e seu código (código aberto) encontra-se num repositório SVN. Todos os procedimentos para a sua instalação (incluindo a configuração do servidor de aplicações, geração de chave RSA, etc..) podem ser encontrados no site *http://colmeia.incubadora.fapesp.br*, nas partes referentes a Documentação/Instalação.

6 Desenvolvimento

6.1 Abordagem de Implementação

Como discutido anteriormente, um dos objetivos desse projeto foi usar a NPDL para definir os processos de negócio do Ciclo De Empréstimo, cuja gerência de execução dos processos será de responsabilidade da NPTOOL. A abordagem de implementação do Ciclo De Empréstimo foi intrinsecamente concebida na tentativa de contemplar os seguintes itens:

- Traduzir facilmente em código java, dentro do Colméia, as ações, regras e funções que definem os processos em NPDL;
- Tornar o gerenciamento desses processos genéricos suficientes para acomodar todos os processos do Ciclo De Empréstimo;
- Contribuir efetivamente para o desenvolvimento do projeto Colméia como um todo, não apenas com o Ciclo De Empréstimo, mas provendo suporte necessário para a implementação de outros módulos que se utilizem, pelos motivos anteriormente discutidos, de modelos de negócios definidos por processos NPDL.

Para se entender esses itens propostos, bem como as decisões de implementação tomadas, se faz indispensável um entendimento razoavelmente bom sobre o que é um processo definido em NPDL e como funciona a execução desse processo pelo interpretador NPTOOL (ver *http://www.vision.ime.usp.br/∼kellyrb*).

Tendo esses objetivos priorizados, foi então que se mostrou necessária uma implementação de um framework, ainda que seu código fosse implementado dentro do Colméia, provendo uma razoavelmente fácil implementação dos módulos propostos.

6.2 Um framework Java para a NPDL

Embora um framework seja externo a qualquer aplicação, toma-se aqui a liberdade de tratar a implementação realizada no Colméia como tal, pelo simples motivo de que todas as suas classes podem ser exportadas e incluídas em qualquer outra aplicação que queira ter seu modelo de negócio expressado por um processo NPDL. Embora as sessões que se seguem possuam um cunho mais técnico, o objetivo não será simplesmente mostrar como foi feita a implementação desse framework em si, mas sim discutir a natureza dos problemas identificados na implementação de um framework java para a NPDL, e explicar as soluções propostas e que se encontram codificadas no repositório do Colméia. Espera-se então, que a leitura dessas sessões se mostre útil para futuras implementações de módulos no Colméia ou de outros aplicativos java que usem a NPDL para representar seus processos de negócio.

6.2.1 O que é um framework Java para a NPDL

Um framework NPDL[1] é um conjunto de classes escrito na linguagem de programação Java que objetiva facilitar a implementação de aplicativos em Java cujos modelos de negócio estão representados por processos NPDL, e cujo gerenciamento da execução desses processos é realizado pela NPTOOL.

6.2.2 Fundamentos da execução de um processo NPDL

Antes de entrarmos em maiores detalhes técnicos do framework, é importante nos atermos aos pontos principais da execução de um processo em NPDL.
Em NPDL, temos as ações, as regras e funções, que juntamente com os operadores, definem um processo. Essas ações, regras e funções são de maneira generalizada, denominadas passos. A execução da instância de um processo NPDL, nada mais é do que uma sequência de execução de passos que vão sendo habilitados para execução; em outras palavras, se uma instância de um processo estiver finalizada, isso quer dizer que mais nenhum passo será

[1]Quando usados os termos *framework NPDL*, o leitor deve entender esses como um *framework Java para a NPDL*

habilitado para execução.
Visto que tanto as ações, as regras e as funções NPDL são passos, é explicado então a primeira classe:

6.2.3 Classe Passo.java

A classe Passo é a classe que representa o que a aplicação entende ser um passo NPDL. E o que é ser um passo NPDL para uma aplicação?
Como dito anteriormente, a execução de um processo em NPDL consiste em uma sequência de execução dos passos habilitados para execução; porém, as execuções dos passos independem do valor semântico que esses passos têm em alguma aplicação. Por exemplo: quando um processo é definido em NPDL, podemos executar esse processo independente de se ter uma aplicação associada a ele. Em outras palavras, uma aplicação que se utilize desse processo deve conceber aos passos desse processo, um valor semântico baseado no contexto de negócio da aplicação, fazendo com que aquele processo expresse inclusive a lógica do modelo de negócio da aplicação. Em virtude disso, podemos dizer que uma aplicação entende um passo NPDL como sendo algo com um valor semântico e que deve ser consistido, e esse algo precisa ser codificado. Tendo isso em mente, dois métodos dessa classe são destacados a seguir:

1) abstract void avaliaPasso(Object obj);

Esse método é o método que atribui valor semântico ao passo. Ele é um método abstrato pois cada passo na expressão NPDL pode ter um valor semântico diferente na aplicação, e desta forma permite que cada passo da expressão possa (e deva) ter sua própria implementação. O parâmetro *obj* permite que no momento da avaliação do passo, este receba informações necessárias para o cumprimento da avaliação.

2) abstract void executaPasso();

Esse método é o responsável por comunicar a NPTOOL de que o passo foi executado, a fim de que novos passos possam ser habilitados para execução.

Esse método também é implementado como abstrato, pois a informação de execução é diferente entre regras, ações e funções. Por exemplo, para informar a NPTOOL de que uma ação foi executada, basta informar exatamente isso, sem nenhuma outra informação adicional. Porém, se o passo for uma regra, esta pode ser executada como verdadeira ou falsa, o que implica em informação adicional na execução de um passo. Pensando nessas possibilidades é que o método é abstrato, deixando para as especializações de passo (ações, regras e funções) uma implementação própria.

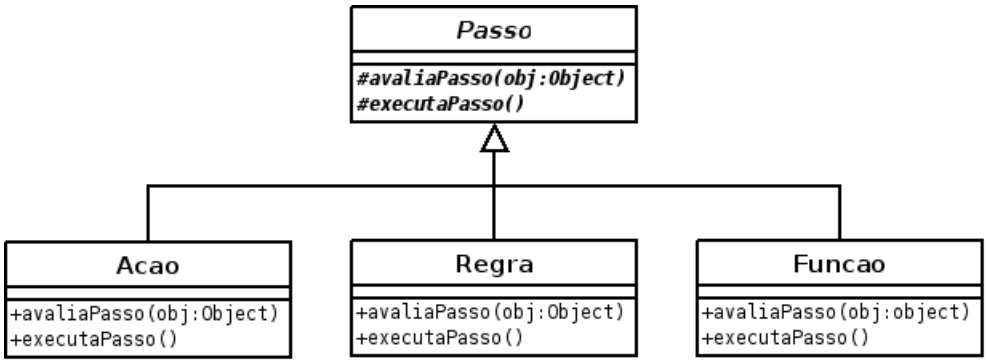

Figura 12: Generalização das ações, regras e funções NPDL

6.2.4 Classe PassoNpdl.java

Mas o que aconteceu com a classe Passo.java, não ficou boa?
Não, não há nenhum engano aqui. Como discutido anteriormente, a classe Passo.java é a classe que representa o que uma aplicação entende ser um passo NPDL. A classe PassoNpdl.java é simplesmente a representação das principais informações que a NPTOOL disponibiliza de um passo. Para a NPTOOL, um passo é constituído, entre outras informações menos relevantes, principalmente pela sua descrição, que é uma string, e pelo seu id, um long. A classe PassoNpdl.java simplesmente concentra essas informações que serão úteis no decorrer da execução da aplicação.

6.2.5 Classe FabricaDePassos.java

Quando um passo é habilitado para execução, as informações que nos são disponibilizadas é tal como explicado acima, a descrição e o id do passo, e são exatamente com esses dados que lidamos em uma aplicação. Então o problema que se verifica aqui é como estabelecer a correspondência entre um passo retornado pela NPTOOL (cujas informações foram armazenadas em um objeto da classe PassoNpdl.java) e o passo na aplicação que o representa. Essa classe se propõe então a realizar essa tarefa. Por meio do método

Passo retornaPasso(PassoNpdl passoNpdl)

que, dado um objeto da classe PassoNpdl.java, retorna o objeto correspondente na aplicação, que é um objeto de uma classe que estende a classe Passo.java, e este objeto então é o passo com valor semântico na aplicação.

Embora o leitor já deva ter concluído o que iremos explicitar, talvez seja um bom momento para esclarecermos que para toda ação, regra ou função retornada pela NPDL, deve existir uma classe que represente esses passos e essa classe deve estender a classe Passo.java, como foi exposto no trecho: *"dado um objeto da classe PassoNpdl.java, retorna o objeto correspondente na aplicação, que é um objeto de uma classe que estende a classe Passo.java"*.

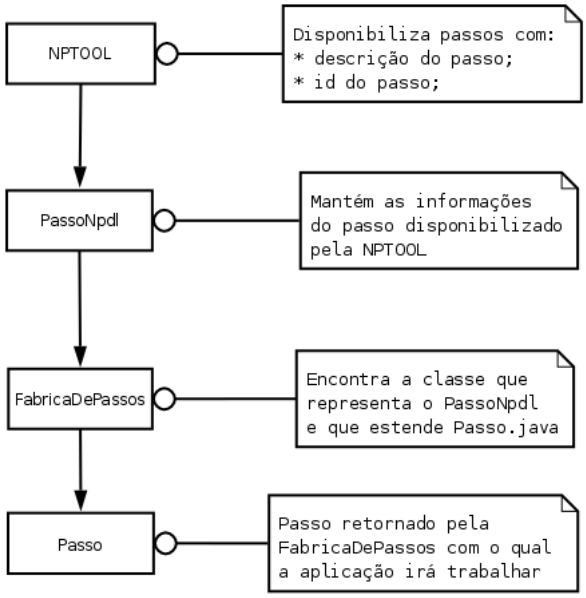

Figura 13: Transformações de um passo NPDL pelo framework
Nota: Para facilitar o entendimento do processo descrito pela figura, foi omitida a camada de comunicação entre o NPTOOL e a aplicação, que é feita pela NPWS

1. Criação de instâncias de processos

Cria de maneira fácil e intuitiva instâncias de processos para serem executadas. Por meio do seu construtor, é possível indicar qual é o processo NPDL definido do qual a aplicação se utilizará. Segue abaixo um exemplo de criação de uma instância para o processo P_EMPRESTIMO, que é o processo definido para empréstimo de exemplares no Colméia, como anteriormente explicado:

ControladorDePassos con;
con = new ControladorDePassos(Ciclos.P_EMPRESTIMO);

O parâmetro Ciclos.P_EMPRESTIMO é uma string de descrição do processo P_EMPRESTIMO, que no Colméia, por exemplo, é a própria string "P_EMPRESTIMO". Realizados esses comandos, temos então um ControladorDePassos 'con' para uma nova instância do processo P_EMPRESTIMO. Essa instância de processo recebe um id da NPTOOL, e esse id é armazenado no ControladorDePassos para as execuções do processo.

2. Controla a execução do processo na aplicação

Controla a execução de um processo definido para a aplicação de forma a ser facilmente manipulável. Seguindo o exemplo do processo P_EMPRESTIMO, temos então a referência 'con' de um ControladorDePassos para um processo P_EMPRESTIMO. Para executar essa instância, basta dispararmos a execução fazendo:

con.iniciaFluxo(obj);

Esse método, *iniciaFluxo(Object obj)*, irá disparar a execução da instância do processo, e sintetizadamente o que ele faz é:

1. Administra a entrada de passos disponíveis para serem executados. Isso é feito pelas chamadas remotas dos métodos da NPTOOL por meio da NPWS, que devolvem passos habilitados para a execução.

2. Força a avaliação dos passos, fazendo com que a semântica de cada passo seja avaliada.

3. Força a comunicação de execução dos passos para a NPTOOL, novamente por meio de chamadas remotas.

Esse método fica muito mais interessante quando verificamos que se esses três itens acima descritos fossem colocados em *looping*, teríamos então a execução inteira de um processo, e é exatamente isso o que é feito para a execução de um processo. Abaixo está uma ilustração desse *looping*:

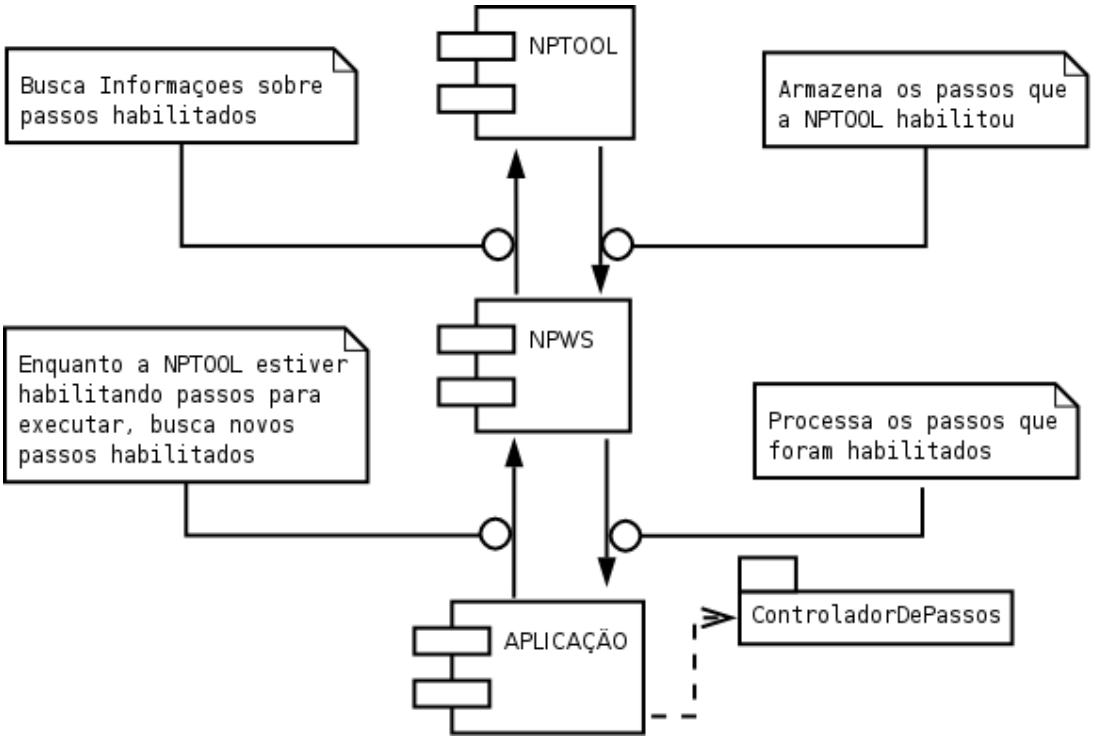

Figura 14: Execução de um processo usando o ColtroladorDePassos
Nota: Esse diagrama mostra também a interação com a ferramenta NPWS, embora isso seja transparente dentro do ControladorDePassos

Durante a execução de um processo, às vezes se faz necessário parar a execução e posteriormente retomá-la. Um caso onde isso ocorre, por exemplo, é no processo de empréstimo (P_EMPRESTIMO) do Colméia. Nesse processo, há uma parada de execução do processo para que a tela do usuário do sistema seja atualizada (embora isso pudesse também ser uma ação na expressão NPDL) e posteriormente, é retomada a execução do ponto de parada. Essa funcionalidade pode ser realizada por meio da chamada do método *retomaInstancia(idInstancia)*, que retorna uma nova referência do ControladorDePassos, cujo processo que ele está administrando é o processo com

id igual a *idInstancia*, e assim, quando o método *iniciaFluxo(Object obj)* é chamado, a execução dessa instância retoma do ponto de parada.

Para permitir uma fácil parada na execução do processo, sem a necessidade de ser adicionada à expressão NPDL uma nova ação para cada parada de execução prevista, a classe Passo.java tem uma propriedade do tipo booleana que indica se o passo é um passo de parada.
Após a execução de qualquer passo, é verificado se o passo em questão tem essa propriedade indicada como *true* e, caso seja, então a execução do processo é parada, podendo futuramente ser retomada.

Um detalhe que até aqui foi negligenciado, é que estamos falando do método *iniciaFluxo(Object obj)* sem explicar o parâmetro "obj". Como dito anteriormente durante a explicação da classe Passo.java, por vezes os passos precisam de informações adicionais para que suas avaliações semânticas possam se realizar. Esse parâmetro obj é então um parâmetro que será repassado para todos os passos que forem sendo instanciados na aplicação, de tal forma que as avaliações semânticas dos mesmos possam ser realizadas.

6.2.6 Pontos fortes e fracos do framework

Nos tópicos acima, discorreu-se sobre os pontos principais do processo de criação do framework NPDL, ainda que sem entrar em maiores detalhes do funcionamento do mesmo, buscando, na medida do possível, identificar os problemas na criação de um framework NPDL e explicando as soluções propostas. A leitura desses tópicos consolida-se em um material útil não apenas para o entendimento e desenvolvimento do projeto Colméia, mas de aplicativos outros que se utilizam da NPDL para expressar seus modelos de negócio.

Nessa sessão, serão expostos alguns pontos fortes e fracos de uma implementação nos moldes que esse framework foi concebido, objetivando que os leitores não fiquem doutrinados nessa implementação.

Da forma como foi implementado, esse framework provê uma maneira ágil de implementarmos aplicativos que usam NPDL. Ágil principalmente porque não há trabalho adicional para ser realizado na estrutura do framework,

fazendo com que os esforços de implementação se concentrem apenas no que é exclusivo da aplicação. A coesão é drasticamente aumentada segundo essa implementação, pois cada passo possui uma tarefa específica dentro da expressão NPDL, e essa especificidade é também concebida ao código.

A manutenção do código é muito facilitada, pois quase toda estruturação lógica é de responsabilidade da expressão, e esta não é codificada dentro da aplicação, fazendo com que mudanças ou correções do modelo de negócio, não demandem uma dispendiosa procura entre as possíveis muitas classes da aplicação, pelo trecho de código que deverá ser alterado, pois se sabe que a funcionalidade do sistema que deve ser alterado está codificada em algum passo, e basta identificar esse passo pela própria expressão NPDL.

A clareza no código é uma outra característica desse framework, pois utilizando-se apenas duas linhas de código, uma para instanciar um ControladorDePassos para um determinado processo e outra para disparar a execução do fluxo, podemos executar processos complexos cuja implementação não está misturada com outras partes e componentes do sistema.

Essas são apenas algumas das vantagens que foram identificadas durante desenvolvimento do Ciclo de Empréstimo do Colméia ao usar esse framework construído. Para o projeto Colméia, esse framework funcionou de forma muito satisfatória, e assim será para a maioria das aplicações. Porém, vale ressaltar alguns pontos importantes, mais especificamente algumas limitações impostas pelo framework. Essas limitações das quais falaremos são devidas a certas restrições de usabilidade da NPTOOL pelo framework. Por exemplo, quando é feita uma busca por passos habilitados para execução, não apenas um único passo pode estar habilitado. Da forma como esse framework funciona, é eleito um passo por vez para ser executado, quando na verdade mais do que um já poderia ser executado. Dependendo da aplicação, se esta tiver uma expressão NPDL muito complexa e que exija rapidez na execução do processo, seria muito interessante que pudéssemos executar todos os passos habilitados de uma só vez. Para aqueles que se identificam com esse problema, ainda assim é muito aconselhável que se utilizem desse framework como base de desenvolvimento, pois nele está facilmente verificável quais os métodos que poderiam ser alterados para que esse framework suprisse essa

necessidade.

Outra limitação imposta, e talvez a mais importante, é que para esse framework, todo passo na expressão NPDL é uma classe java, quando para a NPDL, um passo pode ser qualquer ação que se queira representar, uma chamada de método, a execução de um programa, etc. Outras funcionalidades também não estão implementadas, como acompanhar de maneira simplificada um log de execução de um processo, que dependendo da aplicação poderá ser importante.

Como anteriormente dito, para a maioria das aplicação java, essa abordagem de implementação servirá de maneira muito satisfatória para a utilização da NPDL, assim como foi para o projeto Colméia, sendo, entretanto, o intuito dessa sessão, alertar sobre o fato de que dependendo do aplicativo que será construído, pode ser interessante partir para uma implementação própria de um framework, o que seria o caso mais extremo, ou usar este framework como base de desenvolvimento e incrementá-lo com funcionalidades mais específicas.

6.3 As Telas do Ciclo de Empréstimo

Como foi apresentado na seção 2.5, o Apache Velocity é responsável pela renderização das telas no Colméia, que foram implementadas diretamente pelos desenvolvedores, sem auxílio de qualquer ferramenta. As telas do Ciclo de Empréstimo foram implementadas seguindo o padrão adotado por todo o Colmeia, desde a sua aparência até a organização dentro do código fonte.

O código Velocity aproxima-se bastante de HTML, como demonstra o código a seguir, retirado de uma das páginas do Ciclo de Empréstimo:

```
<table align="center" class="in" style="width: 400px">
  <tr>
    <td align="left" style="width: 130px">
      $text.get("mov.emprestimoCampoExemplar")
    </td>
    <td align="left" style="width: 130px">
```

```
      $requisitante.getNome()
    </td>
  </tr>
  <tr>
    <td align="left" style="width: 130px">
      $text.get("mov.emprestimoCampoDocumento")
    </td>
    <td align="left" style="width: 100px">
      #cortaNome($exemplarDeEmprestimo.getTitulo())
    </td>
  </tr>
</table>
```

Este trecho de código Velocity ilustra alguns dos recursos utilizados no Ciclo de Empréstimo. Primeiro nota-se a semelhança com código HTML, com tags como `<table> e <td>`. As telas do Ciclo de Empréstimo são todas implementadas como uma grande tabela. Na primeira linha fica o menu de seleção de seção, o Ciclo de Empréstimo fica dentro da seção Movimentações. A segunda linha é dividida em duas colunas. Na coluna da esquerda fica o menu de navegação na seção de Movimentações e na da direita fica outra tabela (é possível manter esta estrutura recursiva) que contém os forms e dados pertinentes a cada operação.

Também notamos no código referências a objetos complexos. $requisitante é uma referência direta a um objeto Java, instância da classe Pessoa. Esse objeto é colocado na sessão por meio do do método sessao.setAttribute(), que recebe como parâmetros o objeto a ser colocado e o nome com q ele será manipulado dentro desta. Para mais informações sobre como os objetos são mapeados vide seção 6.4.

Com os objetos corretamente mapeados podemos manipulá-los como faríamos dentro do código Java, ou seja, a partir dos seus métodos de acesso. Neste exemplo pode-se observar que a tela exibe o nome do requisitante do empréstimo por meio do método $requisitante.getNome().

Além disso, o Struts nos fornece a referência $text que possibilita que o texto a ser exibido na tela seja retirado de arquivos (no caso do Colmeia são

eles *ApplicationResources.properties* e *ApplicationResources_pt.properties*) a partir do método $text.get(). Isso permite que o texto seja modificado dependendo do estado da aplicação. No Colméia este recurso é utilizado para que as páginas fiquem disponíveis tanto em inglês quanto em português, dependendo da preferência do usuário. Ainda notamos a macro #cortaNome, que é basicamente uma tag que substitui um pedaço de código Velocity. Nesse caso, #cortaNome limita o número de caracteres a ser exibido na tela, com o código:

```
#if($nome.length() > 30)
  $nome.substring(0, 30) [...]
#else
  $nome
#end
```

Este código substitiu a tag #cortaNome($exemplarDeEmprestimo.getTitulo()) diretamente no código mais acima. Aqui podemos notar o uso do par lógico `#if ... #else`, outro recurso apresentado pelo Velocity utilizado no Ciclo de Empréstimo. Os statements booleanos que são usado como parâmetros suportam comparação de inteiros, strings e objetos.

São disponibilizados além de recursos lógicos como estes, loops por meio do comando `#foreach ... #end`, que percorrem listas (arrays, collections ou maps) e também qualquer recurso disponível em HTML, como forms, headings e etc. Estes recursos amarrados ao arcabouço Struts (melhor explicado nas seções 2.6 e 6.4) possibilitaram a implementação de todas as telas do trabalho.

6.4 Struts e Velocity: como são integrados e utilizados

Esta seção visa explicar como é feita a amarração entre o modelo, o controlador e a visão, relembrando que a visão usa o Velocity, o controlador é provido pelo framework Struts e o modelo são as classes Java que implementam o modelo de negócios proposto. Vale lembrar que o texto se concentrará no Ciclo

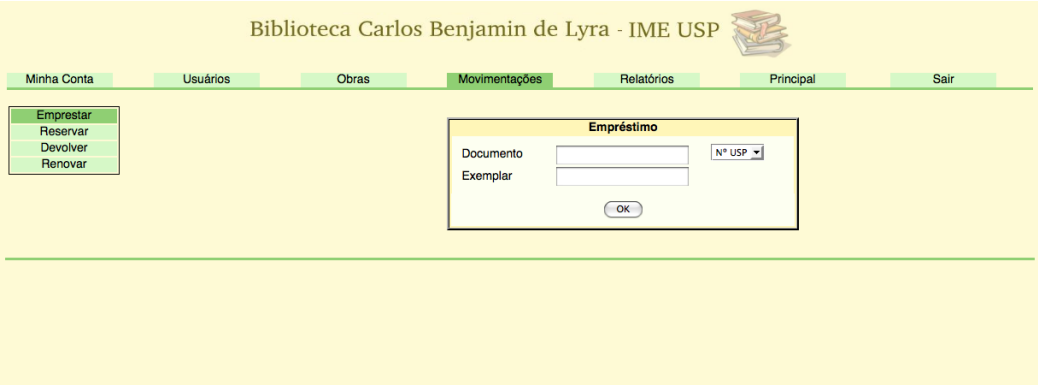

Figura 15: Tela inicial do Ciclo de Empréstimo

de Empréstimo, pois nem todas as partes do Colmeia (como o Zumbido e a busca) usam esta arquitetura.

Para ilustrar como é feita essa amarração será dado o exemplo do documento que o usuário usa para fazer o empréstimo. Este documento pode ser o número USP do aluno, ou um CPF, RG, RNE ou CRB. Independentemente do tipo de documento, para o Ciclo de Empréstimo, é simplesmente um número.

Este número é entrado pelo usuário que deseja emprestar o exemplar, junto com o tipo de documento e o código de barras do exemplar. Esses são os dados necessários para que o empréstimo seja realizado.

O documento do usuário é recebido em um form Velocity, que é apresentado aqui de maneira simplificada:

```
<form name="emprestimoForm"
      action="$link.setAction("/emprestimo")"/>
<input type=text name="documento"/>
#selectOptions("doctipo" $enumDocGen.getMapInstances()
                         $enumDocGen.defaultDocumentTypeId 110 "")
<input type=text name="cbarras"/>
```

```
<input type=submit value="  OK  "/>
</form>
```

Aqui podemos observar que temos os dois inputs que foram citados: *documento*, *doctipo* e *cbarras*, referentes ao documento do requisitante, ao tipo do documento e ao código de barras do exemplar a ser emprestado, respectivamente. Além disso, temos indicados o nome do form, no caso *emprestimoForm* e a ação que deverá ser realizada quando o o form for enviado, neste caso */emprestimo*.

Mas como o aplicativo sabe a que se referem cada um desses dados? O Struts constrói esta relação por meio do arquivo de configuração *struts-config.xml*. Os trechos de código XML abaixo são referentes ao form *emprestimoForm*, onde se encontra o documento do requisitante, retirados do *struts-config.xml*:

```
<form-bean name="EmprestimoForm"
  type="br.usp.ime.colmeia.movimentacao.forms.EmprestimoForm" />
```

Este trecho de código informa ao Struts qual classe Java se relaciona àquele determinado form. Olhando para o código do construtor da classe *EmprestimoForm*, encontramos as variáveis a que se referem os dados dentro do form:

```
public EmprestimoForm() {
  doctipo="";
  cbarras="";
  documento="";
}
```

Aqui encontram-se as variáveis que se relacionam diretamente com as encontradas no form na página Velocity. Agora é possível manipular estes dados que foram retirados da página diretamente da aplicação.

Além de extrair estes dados do form Velocity, há a ação que é realizada quando este é submetido. Examinando a tag do *emprestimoForm*, verificamos que ele realiza a ação */emprestimo*. Olhando outro pedaço do código no arquivo *struts-config.xml*, mais uma vez simplificado, encontramos qual classe Java é executada:

```
<action path="/emprestimo"
   input="/movimentacao/emprestimo/emprestimo.vm"
   name="EmprestimoForm"
   type="br.usp.ime.colmeia.movimentacao.actions.EmprestimoAction">
</action>
```

Fica indicada a execução da classe *EmprestimoAction*. O código que estiver dentro do método *execute()* nesta classe será executado quando o usuário clicar no botão que envia o form.

Este processo se repete ao longo de todo o Ciclo de Empréstimo e seus subciclos. Dessa maneira, os papéis definidos no MVC ficam claros dentro da aplicação. O modelo são as classes Java como a *EmprestimoAction*, que são executadas quando um form é submetido, juntamente com as como a classe *EmprestimoForm*, que retém as informações referentes aos forms. O controlador é o próprio framework Struts, que gerencia a comunicação entre visão e modelo. E por fim, a visão fica a cargo do engine Velocity, que renderiza as páginas.

Uma grande vantagem do uso do framework Struts é que se torna possível trocar o engine de renderização das páginas sem que se precise alterar qualquer outro trecho no código.

7 Conclusão

Com a realização desse trabalho, o Colméia conta hoje com uma estrutura pronta para acomodar as funcionalidades que envolvem o Ciclo de Empréstimo, que cobre o empréstimo de exemplares, devolução de

exemplares, renovação de empréstimos realizados e reserva de obras. Toda a interface com o usuário está implementada, e esta foi cuidadosamente implementada para ser de fácil usabilidade tanto para os funcionários da biblioteca quanto para os alunos, professores e outros associados.

Com relação aos processos que definem o Ciclo De Empréstimo, todas as ações, regras e funções que o governam estão implementados segundo a arquitetura do framework construído, e devido a essa abordagem, com todos os benefícios já explicados, sendo talvez o mais importante, a facilidade com que futuros ciclos possam ser implementados.

As funcionalidades do Ciclo De Empréstimo, que hoje são realizadas manualmente pelos funcionários da biblioteca, muito em breve, quando o sistema for posto em produção, serão feitas via sistema, oferecendo uma melhora tanto dos serviços propriamente dito, quanto na administração do acervo do IME. Como exemplo de melhora de serviços, uma imediata vantagem que será observada, é que a reserva não será mais feita em um exemplar específico, e sim na obra, como de fato deveria ser, além de poder ser feita por um aluno ou professor, sem a necessidade de se dirigir à biblioteca.

Também se poderá ter um controle melhor sobre a situação das pessoas que estão retirando exemplares na biblioteca, pois a cada empréstimo renovação ou reserva que se queira realizar, o sistema poderá de maneira muito mais abrangente do que é feito hoje, consistir a situação do cadastro dessas pessoas e indicar possíveis irregularidades que anteriormente eram negligenciadas, devido à realização manual dessas atividades.

Esses foram apenas alguns exemplos entre os muitos exemplos que fazem com que esse sistema seja de muita importância para a biblioteca do IME.

8 Próximos Passos

O Ciclo de Empréstimo, apesar de se encontrar completamente implementado e funcionando corretamente, encontra ainda tarefas pela frente até que possa ser liberado para fase de produção. Primeiramente, o Empréstimo e seus

subciclos (Devolução, Reserva e Renovação) dependem de conexões com o banco de dados, a chamada camada de persistência, que ainda não foram implementadas pois fugiam do escopo do projeto, que era a implementação do Ciclo utilizando o controle de fluxo externo. Assim, é necessário ainda implementar diversas stored procedures no banco e funções de controle no Colméia. Algumas delas já foram feitas, e as demais devem ser parecidas com essas.

Ainda dentro do Empréstimo, a entrada dos códigos de barras dos exemplares é feita manualmente. O objetivo é que se use um leitor de código de barras, mas por falta de um disponível para o grupo integralmente para desenvolvimento e teste, não foi possível verificar o funcionamento de um destes periféricos. Apesar disso, para que o sistema funcione com um, serão necessário somente pequenos ajustes no código. Há ainda a questão da Reserva, que no momento só pode ser realizada por funcionários e que, no futuro, deve poder ser realizada por qualquer usuário do sistema. Além disso, como o Ciclo de Empréstimo foi a primeira aplicação real a usar a NPTool, muitos dos problemas e detalhes que foram observados durante o processo, além das vantagens que foram descobertas ao longo do processo (vide seção 7) serão compilados em um documento que será traduzido para a língua inglesa.

Por fim, há o Ciclo de Aquisição que deve ser parcialmente implementado ainda neste semestre na disciplina de *Laboratório de Banco de Dados*. Ele deve seguir os moldes do Ciclo de Empréstimo, utilizando as ferramentas de controle de fluxo e o framework Java desenvolvido neste projeto.

Referências

[1] Eclipse. url *http://eclipse.org*.

[2] Jboss. url *http://labs.jboss.com*.

[3] PostgreSQL. url *http://www.postgresql.org.br*.

[4] Struts. url *http://struts.apache.org*.

[5] Subversion. url *http://subversion.tigris.org*.

[6] Tomcat. url *http://tomcat.apache.org*.

[7] Velocity. url *http://velocity.apache.org*.

[8] Kelly Rosa Braghetto. Padrões de Fluxos de Processos em Banco de Dados Relacionais. *Dissertação de Mestrado*, Instituto de Matemática e Estatística, Universidade de São Paulo, 2006. url *http://www.vision.ime.usp.br/∼kellyrb/nptool/krbraghetto_dissertacao.pdf*.

[9] Kelly Rosa Braghetto. A *Navigation Plan Definition Language* e A *NavigationPlanTool*. *Relatório Técnico*, Instituto de Matemática e Estatística, Universidade de São Paulo, 2006. url *http://www.vision.ime.usp.br/∼kellyrb/nptool/npdl_nptool.pdf*.

[10] Maurício Chui Rodrigues, Kelly Rosa Braghetto, Marcos Eduardo Bolelli Broinizi e João Eduardo Ferreira. Encapsulando a *NavigationPlanTool* como Serviços Web. *Relatório Técnico*, Instituto de Matemática e Estatística, Universidade de São Paulo, 2007. url *http://www.ime.usp.br/∼chui/artigos/sicpg2007.pdf*.

9 Parte Subjetiva

9.1 Desafios e frustrações encontrados

Nessa sessão descreveremos os desafios e as frustrações encontradas durante o desenvolvimento do projeto, e que foram comuns entre os participantes.

O primeiro grande desafio com o qual nos deparamos foi entrar em um projeto do porte do Colméia sem grande experiência em desenvolvimento de sistemas. Durante toda nossa vida acadêmica, nunca realizamos desenvolvimento de sistemas propriamente ditos, sendo a nossa experiência em programação consolidada basicamente em desenvolvimento de exercícios-programa, que geralmente têm muito pouco em comum com desenvolvimento de sistemas comerciais. Em virtude disso, dispendeu-se uma boa quantidade de tempo para, primeiramente, filtrar quais das muitas classes do Colméia seriam alteradas pela nossa implementação, e em segundo, absorver um conhecimento mínimo de todas as tecnologias que o Colméia utiliza. Basicamente, a única tecnologia que tínhamos conhecimento de fato era Java genericamente falando, o que não era suficiente para sair escrevendo linhas de código. O processo de "absorver um conhecimento mínimo"embora não pareça muito difícil devido ao grande número de materiais disponíveis na internet, pode se tornar bastante complicado quando são muitas as tecnologias que se tem de aprender e ainda mais quando são todas ao mesmo tempo.

Algo que não faltou durante o nosso desenvolvimento foram problemas técnicos entre as tecnologias que adotamos, que acarretaram em grande quantidade de tempo "ocioso"no desenvolvimento. Os problemas variavam desde problemas mais simples como versão do driver do Postgres que fazia com que a NPTOOL não se comportasse da maneira esperada, e que ocasionaram atrasos de uma a duas semanas, até problemas de conflitos entre as bibliotecas do Colméia com as do NPWS, que ocasionaram em torno de um mês de desenvolvimento comprometidos e mais de cinqüenta emails trocados com o Maurício Chui (criador da NPWS) na tentativa de sanar o problema de rodar o NPWS no Colméia. Devido a todos esses e outros problemas, acreditamos que a nossa maior frustração no desenvolvimento do Colméia foi não conseguir realizar a parte de persistência do Ciclo

de Empréstimo, pois tivemos que optar em nos concentrarmos mais nas camadas de interação com usuário e lógica do sistema, ainda fornecendo interfaces bem definidas para a implementação de maneira fácil da camada de persistência.

9.1.1 André Guerra

O projeto do Ciclo de Empréstimo foi um grande aprendizado, tanto técnico como social. Muitos conceitos que antes do trabalho de conclusão era bastante etéreos se tornaram concretos, à medida em que eram analisados com mais profundidade mostravam suas características, tanto boas quanto ruins. Foi o caso de diversos padrões de projeto que são utilizados no projeto, como *Factory, Model View Controller, Singleton* e por aí afora, além de técnicas como *Stored Procedures* e tecnologias como *Tomcat, JBoss* e muitas outras, além da própria linguagem Java, que a cada classe implementada mostrava suas fraquezas (tipagem, hierarquias muitas vezes confusas) e virtudes (principalmente o poderio de suas inúmeras bibliotecas).

Socialmente o aprendizado também foi grande. Aprendemos a como lidar uns com os outros dentro da equipe, como conciliar, negar e defender idéias, opiniões tanto de implementação propriamente dita como também de design e arquitetura, conciliar horários muitas e muitas vezes incompatíveis, e ter um olhar crítico sobre tudo que foi feito por você e pelos outros membro do grupo. Além disso, aprendemos também a lidar com pessoas de fora do grupo, que têm outros interesses além do nosso projeto e não podem priorizá-lo como nós. Aprendemos também a lidar com um cliente, como tentar olhar através dos olhos da pessoa que vê o produto como uma caixa preta, espera certas funcionalidades dele e raciocinar como essas funcionalidades deverão ser implementadas.

Obviamente todos esses desafios de aprendizado nos levaram a muitos becos sem saída, erros e frustrações. A preparação do ambiente para desenvolvimento foi uma destas grandes frustrações. Diversas incompatibilidades entre bibliotecas, drivers e outros detalhes frequentemente nos impediam de continuar a implementação. Uma dessas incompatibilidades em

particular fez com que o projeto ficasse quase parado durante praticamente um mês, até que com bastante trabalho o Ricardo conseguiu resolvê-la e conseguimos seguir em frente.

No geral foi uma ótima experiência, de grande aprendizado. A única ressalva que pode ser feita é quanto ao estado atual da implementação do Colméia. Com o passar de vários anos sendo desenvolvido por diversas equipes, o sistema tornou-se uma grande colcha de retalhos. Muitas tecnologias, algumas que se sobrepõem umas às outras, e outras que não se justificam dentro do sistema, compõem o código. Há um grande inchaço, muito código para um sistema que não precisaria ser tão complexo.

As seguintes disciplinas foram as que considero mais terem contribuído para que eu conseguisse desempenhar meu papel nesse projeto:

- MAC110 - Introdução à Computação: Trouxe uma primeira introdução ao que é programação, indispensável para qualquer outro curso
- MAC122 - Princípio de Desenvolvimento de Algoritmos: Introduz as primeiras técnicas reais de programação e primeiras estruturas de dados, também indispensável.
- MAC 211/242 - Laboratórios de Programação: Proporcionou um contato maior com projetos de maior porte, além de introduzir novas linguagens.
- MAC426 - Sistemas de Bancos de Dados: Introduziu conceitos cruciais para nós, que durante uma parte do projeto ficamos em contato direto com o banco de dados
- MAC342 - Laboratório de Programação eXtrema: Aqui houve um primeiro contato com um grupo maior dentro de um mesmo projeto (no meu grupo haviam 10 pessoas) e também um primeiro contato com um sistema real, que deveria ser entregue em um prazo e seria de fato utilizado fora do IME.
- MAC441 - Programação Orientada a Objetos: Primeiro contato com conceitos importantes e com padrões de projeto.

9.1.2 Fernando Waitman

Esse trabalho permitiu uma experiência na prática com boa parte dos assuntos com os quais eu me deparei no decorrer da graduação.

Por meio dele, tive contato com muitos dos conceitos que envolvem o desenvolvimento completo de uma aplicação web.

Pude conviver com a realidade de, junto a uma equipe, estar à frente da implementação de um módulo num sistema já existente, o que foi extremamente diferente do que eu imaginava.

Percebi que esse tipo de trabalho nos leva muito além do que se imagina a pincípio. Para que a implementação seja aceitável, é preciso adequar a sua parte a conceitos e estratégias que o sistema utiliza, o que pode significar boa parte do tempo de desenvolvimento do projeto.

Outra experiência extremamente boa pra mim foi o desenvolvimento de um projeto junto a uma equipe. Conheci na pática algumas dificuldades, assim como técnicas e tecnologias que facilitam e permitem a "sincronização"do desenvolvimento em conjunto.

Dentre uma série situações difíceis no decorrer da implementação do ciclo de empréstimo, as dificuldades para a preparação de um ambiente adequado para o desenvolvimento foi o que mais marcou. Em muitos momentos, ficamos reféns de alguns problemas e não pudemos prosseguir com o projeto.

O grande número de tecnologias e conceitos diferentes que compõem o sistema Colméia foi um outro grande desafio na minha opinião. Quase toda a implementação foi precedida de um estudo prévio para conhecer as ferramentas e conceitos relacionados. A resolução de alguns problemas encontrados durante a implementação também acabou sendo bastante dificultada devido a essa característica do sistema.

As matérias mais importantes que serviram de base para o desenvolvimento deste trabalho foram:

- MAC110 - Introdução à Computação: Fundamentos para a programação. Uma base indispensável.

- MAC122 - Princípio de Desenvolvimento de Algoritmos: Técnicas e conceitos ligados a programação. Também essencial.
- MAC211 - Laboratório de Programação I: Primeira experiência com o desenvolvimento de projetos. Nessa matéria foram introduzidas as idéias de ambiente de programação, organização, modularização e depuração de código, entre outros conceitos muito utilizados no desenvolvimento desse trabalho.
- MAC328 - Algoritmos em Grafos: Idéia de fluxogramas e grafos orientados, base da linguagem NPDL.
- MAC426 - Sistemas de Bancos de Dados: De extrema importância, pois o trabalho foi baseado no banco de dados do Colméia e os conceitos transmitidos por essa matéria possibilitaram o entendimento desse e também a criação de novas tabelas e *stored procedures* referentes ao Ciclo de Empréstimo.
- MAC332 - Engenharia de Software: Aprofundamento nos conceitos que envolvem o desenvolvimento de um projeto.

Se eu fosse continuar na área, meu primeiro passo seria aprofundar meus conhecimentos em muitos dos conceitos e tecnologias que o projeto Colméia envolve. Acredito que, mesmo com a conclusão do trabalho, em muitos casos o contato com as tecnologias utilizadas foi muito superficial, sendo que isso torna o desenvolvimento muito limitado.

Para me familiarizar mais com a área de desenvolvimento web, acho que me engajaria na leitura de tutoriais diversos e também num maior contato com as tecnologias relacionadas, assim aumentando a minha capacidade para produzir coisas melhores.

Outra coisa que faria seria estudar uma forma de melhorar ainda mais o arcabouço Java para comunicação com NPWS criado com o nosso trabalho, de forma que a performance fosse melhorada e a sua reutilização facilitada.

9.1.3 Ricardo Lazaro

A realização desse trabalho foi sem dúvida nenhuma uma grande cruzada para mim, tendo que conciliar com outras disciplinas no IME (incluindo MAC0300) e com meu estágio.
Muitas foram as noites em que eu sonhei com a mensagem "Unable to createEndPointReference Provider", que era a mensagem de erro devido ao conflito de bibliotecas entre o Colméia e a NPWS, e que nem o google resolvia. Esse erro foi muito desgastante no projeto, pois ninguém (inclusive pessoas mais experientes em programação de aplicativos web) conseguia dar soluções claras para o problema. Então tive que fazer trabalho de "chinês preso"pra descobrir onde estava o problema, como criar aplicativos testes, migrar o Colméia para outros servidores como o Jboss, jetty, glassfish... e nada de funcionar. Em um certo momento, eu já estava pronto para falar com o Jef que não conseguiriamos implementar o Ciclo de Empréstimo do Colméia usando a NPDL, quando, em uma tarde ensolarada de domingo, após um mês de frustrações, funcionou, e como na maioria das vezes, o problema parece óbvil depois que se descobre.
Acredito que tive um grande amadurecimento em programação ao fazer esse projeto, uma porque tive que aprender muitas tecnologias novas e outra, vivenciar na prática o desenvolvimento de um sistema em que muitas pessoas já mexeram, e que é claro no código do Colmeia que muitas pessoas mexeram.
Na minha opinião, para que o Colmeia seja colocado de fato em produção, a forma como ele está sendo desenvolvido deveria ser repensada. Hoje, é claro que ele está muito mais complexo do que ele realmente deveria ser, e isso se deve ao fato de que muitas pessoas com diferentes níveis em programação estão fazendo módulos sem um rigor de qualidade, sem uma supervisão rigorosa dos módulos, e isso é fundamental quando há pessoas sem experiência em desenvolvimento de sistemas, como nos grupos da disciplina de LabXP. Não estou dizendo que não deveriam usar o Colmeia nas disciplinas do IME, apenas alertando sobre fazer uma supervisão mais rigorosa sobre o que é feito no projeto, e para isso, uma equipe (e não apenas uma pessoa como era o caso do Igor, que já não trabalha mais no Colmeia) deveria estar sempre supervisionando o projeto.

A seguir, vou citar as disciplinas que foram mais relevantes para o desenvolvimento do projeto:

- MAC0110 - Introdução à Computação: é a primeira disciplina de programação do curso e os conceitos apresentados aqui são fundamentais para compreender a lógica de programação.
- MAC0122 - Princípio de Desenvolvimento de Algoritmos: Além de se ter um contato com algoritmos mais complexos, acredito que o mais importante pra mim foi o contato com boas práticas de programação.
- MAC0211 - Laboratório de Programação I: nessa disciplina, tive a oportunidade de desenvolver aplicativos maiores, mesmo que ainda nada comparado a sistemas como o Colmeia.
- MAC0426 - Sistemas de Bancos de Dados: essa disciplina foi importante para entender o banco de dados usado no Colmeia, provendo fundamentação para a armazenagem de dados do Colmeia.
- MAC0441 - Programação Orientada a Objetos: O projeto inteiro é feito em Java, que é uma linguagem orientada a objetos, e essa disciplina contribuiu com conceitos importantes em orientação a objetos que ajudaram em uma implementação de melhor nível no projeto.
- MAC0342 - Laboratório de Programação Extrema: Com essa disciplina tive a oportunidade de trabalhar em uma equipe relativamente grande de programadores, e tive o meu primeiro contato com o projeto Colmeia, contribuindo para a implementação dos módulos de busca de obras.

10 Agradecimentos

Em primeiro lugar, gostaríamos de agradecer ao professor João Eduardo Ferreira, que aceitou ser nosso orientador e nos motivou a realizar esse

trabalho, além de contribuir no planejamento de desenvolvimento do projeto. A ele, desejamos que suas expectativas tenham sido satisfeitas com o nosso trabalho.

Agradecemos muito à aluna Kelly, que esteve sempre disposta a sanar nossas dúvidas em relação à NPDL/NPTOOL, e que nos ajudou também no projeto como um todo, dando dicas valiosas no decorrer do desenvolvimento do projeto.

Ao Maurício, que nos ajudou a sanar problemas na execução da NPWS, tendo paciência de responder aos inúmeros emails decorrentes dos problemas que tivemos.

A todos eles um muito obrigado!