Projeto Colméia - Zumbido

Leandro Inácio de Oliveira Bororo
Rodrigo Di Lorenzo Lopes
Orientador: Prof. Dr. Fabio Kon

26 de Novembro de 2007

Sumário

1 Introdução

O projeto Colméia surgiu com o intuito de informatizar e integrar as atividades realizadas em uma biblioteca universitária. Utilizando-se de padrões e ferramentas de software livres, o sistema pretende atender às necessidades dos estudantes, funcionários, professores e membros da comunidade externa.

O projeto vem sendo desenvolvido com a linguagem de programação Java e poderá ser acessado através da rede WEB, visando a atender usuários de diferentes plataformas e localidades.

A coordenação do Colméia está a cargo dos professores Eduardo Colli (Coordenador da Comissão de Biblioteca), Fabio Kon e João Eduardo Ferreira (ambos do Departamento de Ciência da Computação). Desde 2002 alunos das disciplinas de Programação Extrema, ministrada pelo professor Fabio Kon, e Laboratório de Programação, ministrada pelo professor João Eduardo Ferreira, trabalham no desenvolvimento do software. Isso faz com que os conhecimentos adquiridos nessas disciplinas possam ser aplicados de modo a gerar um benefício mútuo entre alunos e instituição, incrementando a relevância e pertinência do projeto.

O principal requisito do Colméia é atender as necessidades da biblioteca do IME[1], como Por exemplo, o cadastro de obras existentes no acervo da biblioteca, o cadastro de usuários e o controle de aquisições e empréstimos.

Dentro deste contexto, no início de Março de 2007 durante a disciplina de Programação Extrema, foi oferecido aos alunos, como opção de projeto para aplicação dos métodos da disciplina, o desenvolvimento de um módulo que atendesse ao requisito de exportação de dados bibliográficos existentes no sistema atual da biblioteca do IME.

O projeto foi então iniciado e recebeu o nome de Zumbido. O nome do projeto é uma metáfora que remete ao meio de comunicação entre abelhas, de modo que, assim como as abelhas se comunicam através do som utilizando seu zumbido, o módulo Colméia-Zumbido deve prover um meio e uma linguagem padrão para a comunicação com outros sistemas bibliográficos automatizados.

Um arquivo contendo informações à respeito das obras e exemplares que têm cadastro no sistema atual foi entregue aos alunos que deveriam, a partir de então, desenvolver um módulo que implementasse as seguintes funcionalidades:

- Cadastro de registros de obras, no banco de dados do sistema, descritos no formato MARC, contidos em um arquivo fornecido através da interface WEB do sistema.

[1] Instituto de Matemática e Estatística da USP

- Cadastro de registros de obras, no banco de dados do sistema, descritos no formato MARC, fornecidos através de uma área de texto da interface WEB do sistema.

- Exportação de dados referentes a obras ou exemplares de obras, selecionadas a partir de algum tipo de filtro, para um arquivo de registros no formato MARC.

- Implementação do protocolo Z39.50

Tendo em vista a implementação das funcionalidades acima, fez-se necessário um estudo do formato MARC e do protocolo Z39.50. Adotou-se como fonte de pesquisa a Internet, uma vez que, esta contém referências para as instituições que mantêm o MARC e o Z39.50 e fornecem material contendo suas especificações.

Pode-se dizer que se denomina MARC o conjunto de formatos padrão para a representação bibliográfica e informações relacionadas que podem ser compreendidos por sistemas computadorizados. Já o protocolo Z39.50 é uma especificação para troca de mensagens entre um cliente e um servidor que habilita o cliente a realizar buscas por registros armazenados em um servidor segundo um determinado critério.

2 O formato MARC

2.1 O que é MARC?

O MARC surgiu em 1968 como resultado de uma série de conferências e experiências lideradas pela LoC[2], que buscava, desde meados de 1950, a automatização de suas operações. MARC é o acrônimo para *MAchine Readable Record*, ou seja, é um formato ou especificação para registros que podem ser interpretados por sistemas computadorizados. A partir de sua formulação inicial, muitas extensões foram criadas para atender requisitos específicos, algumas dessas extensões são o USMARC, utilizado nos Estados Unidos, o CANMARC, utilizado no Canadá, e o MARC21, que surgiu da tentativa de união entre o USMARC e o CANMARC.

O MARC fornece um mecanismo através do qual computadores podem trocar, utilizar e interpretar dados bibliográficos, geralmente é distribuído no formato binário e já é adotado na maioria dos sistemas de catálogo.

[2]Library of Congress - Biblioteca do Congresso Americano

2.2 Estrutura de um registro MARC

Um registro MARC é formado por três elementos básicos:

- Líder (*Leader*): Contém informações que podem ser usadas para o processamento do registro. Consiste de 24 caracteres que determinam individualmente ou em subconjuntos *meta dados* do registro.
- Diretório (*Directory*): Contém o comprimento e a posição de início de todos os campos de dados do material, indexados por marcadores que os identificam.
- Campos de Variáveis (*VariableFields*): Campos que contêm os dados do material, são indexados por marcadores armazenados no Diretório e devem, obrigatoriamente, terminar com o caractere ASCII[3] *1E* hexadecimal. Subdividem-se em:
  - Campos de Variáveis de Controle (*VariableControlFields*):
    São campos indexados por marcadores do tipo 00X. Não podem ser modificados por indicadores ou conter subcampos.
  - Campos de Variáveis de Dados (*VariableDataFields*):
    São campos indexados por marcadores do tipo 01X-9XX. Cada um dos índices (marcadores) corresponde a um tipo de informação a respeito do registro, como por exemplo, o autor da obra ou o seu título, e alguns desses campos são passíveis de repetição.

As seções subseqüentes trazem exemplos ilustrativos do Líder e dos Campos de Variáveis, além de explicar sua semântica de maneira mais detalhada.

2.2.1 Líder (*Leader*)

De maneira geral, o líder traz metadados do registro que está sendo processado. Consiste de 24 caracteres que trazem as seguintes informações:

- 00-04 Uma cadeia de caracteres numéricos que especificam o comprimento do registro.
- 05 Indica através de uma letra minúscula, a relação entre o registro e o arquivo. Por exemplo, se o registro é uma correção de um registro pertencente a um arquivo previamente gerado.

[3] American Standard Code for Information Interchange

- 06 Indica através de uma letra minúscula, características e/ou conteúdo do registro. Por exemplo, o caractere f indica que o registro refere-se a um material cartográfico manuscrito.

- 07 Indica através de uma letra minúscula, o nível bibliográfico do material. Por exemplo, se o material é parte de uma monografia ou se o material é parte de uma série de materiais.

- 08 Indica se o registro está arquivado, através do caractere 'a', ou não, através do caractere ' '.

- 09 Indica se a codificação do registro é do tipo MARC8, através do caractere ' ', ou UCS/Unicode, através do caractere 'a'.

- 10 Indica o número de posições reservadas para indicadores em cada campo de variável.

- 11 Indica o número de posições reservadas para a especificação de um subcampo.

- 12-16 Uma cadeia de 5 caracteres numéricos que especificam a posição do primeiro caractere do primeiro campo de variáveis.

- 17 Indica através de uma letra minúscula, a forma de descrever o catálogo do material. Por exemplo, o caractere 'i' indica que o catálogo do material foi feito segundo especificações da ISBD[4].

- 19 Indica se é possível gerar, a partir dos campos de variáveis indexados pelo conjunto de marcadores 76X-78X, uma nota contendo informações básicas sobre o material, como Por exemplo, a entrada principal do título.

- 20-23 São quatro caracteres numéricos que indicam a estrutura de cada entrada no diretório. Estes campos correspondem à seguinte cadeia de caracteres '4500'.

Um exemplo em formato de texto de um campo Líder:

$$\underset{\text{marcador}}{\underset{\Uparrow}{LEADER}\ 00733nam} \quad \underset{\text{cadeia de 24 caracteres}}{\underset{\Uparrow}{2200241Ia}\ 4500}$$

[4]International Standard Bibliographic Description

Assim, as informações contidas no líder permitem que um programa de computador interprete ou crie um arquivo binário no formato MARC.

2.2.2 Campos de Variáveis (*VariableFields*)

- Variáveis de Controle (*VariableControlFields*)

Campos de Variáveis de Controle contêm, de forma geral, informações fornecidas no momento em que o registro foi gerado. Essas informações podem ser utilizadas em processamentos posteriores do registro. Esses campos contêm números e informações codificadas de controle do registro, fornecidas pela instituição que gerou o arquivo MARC. Os marcadores utilizados variam de 001 a 008 e não pode haver mais de um campo com o mesmo marcador.

Tabela 1: Campos de variável de controle

Marcador	Informação do campo
001	Número de controle fornecido pela instituição que gerou o registro
002	Não é mais utilizado
003	Código que identifica a instituição que gerou o registro
004	Não é mais utilizado
005	Contém a data da última transação envolvendo o registro
006	Características adicionais do material
007	Descrição Física
008	Aspectos especiais do material
009	Não é mais utilizado

- Variáveis de Dados (*VariableDataFields*)

Campos de Variáveis de Dados contêm informações a respeito do material registrado. Assim como os campos de variáveis de controle, são identificados por marcadores. Os marcadores para este tipo de campo variam de 01X a 9XX e cada um deles determina um tipo de informação diferente para a obra cujo registro foi feito.

Tabela 2: Exemplos de campos de variável de dados

Marcador	Informação do campo
010	LCCN[5]
020	ISBN[6]
100	Entrada principal de nome pessoal (autor)
245	Informação de título (inclui título e subtítulos)
250	Informações a respeito da edição do material
260	Informações a respeito da publicação do material
300	Descrição física do material
440	Série ou coleção à qual o material pertence
520	Anotação ou notas de sumário
650	Assuntos associados
700	Entrada adicional para nome pessoal (autor secundário, tradutor etc)

Os marcadores para este tipo de campo seguem uma regra geral estabelecida em termos de intervalos de marcador, ou seja, determinados intervalos de marcadores são utilizados para armazenamento de informações relacionadas, como pode ser visto na tabela abaixo.

Tabela 3: Regras gerais para marcadores

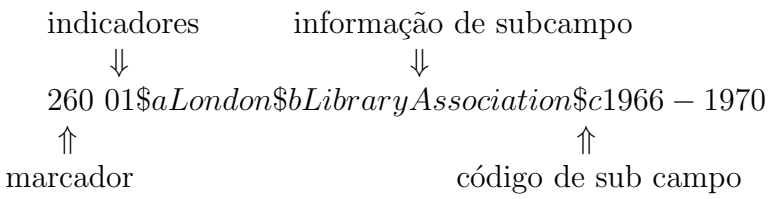
Marcadores	Tipo de informação
0XX	Informação, números e códigos de controle
1XX	Entrada principal
2XX	Título, declaração de responsabilidade, edição, e publicação
3XX	Descrição física
4XX	Informação a respeito da série ou coleção
5XX	Notas
6XX	Entradas adicionais de assunto
7XX	Entradas adicionais diferentes de assuntos ou séries
8XX	Entradas adicionais de séries
9XX	Entradas definidas localmente

Os campos de variáveis de dados podem conter *indicadores* ou *códigos de subcampo* que adicionam informação para a interpretação de um campo.

Os indicadores são dois caracteres que ocupam alguma posição dentro do campo de acordo com o que foi especificado no Líder. Devem ser compostos por letras minúsculas ou números e seus caracteres devem ser interpretados de maneira independente. Caso um dos indicadores não seja necessário em um determinado campo, sua posição deve conter o caractere ' '. Por motivos didáticos os indicadores também são referenciados como primeiro indicador e segundo indicador. Para cada tipo de campo, indexado por um dos marcadores, os indicadores adicionam um tipo de informação diferente.

Os códigos de subcampo são dois caracteres que servem para dividir a informação contida em um campo em elementos de dados. Cada código de subcampo é definido por dois caracteres, um delimitador de caractere (ASCII *1F*) seguido de uma letra minúscula ou de um número. Um mesmo código de campo pode aparecer em diferentes campos dividindo-os de diferentes maneiras.

Um exemplo em formato de texto de um campo de variável de dado:



Marcadores, indicadores e códigos de subcampo são chamados, nos materiais de referência disponíveis, de *designadores de conteúdo*. Ao contrário dos marcadores que devem ser determinados no Líder do registro, indicadores e códigos de subcampos dependem do material cujo registro foi gerado e, portanto, sua ocorrência é determinada em tempo de processamento.

2.3 Conclusões

O formato MARC estabelece um padrão para armazenamento de informações bibliográficas que possibilita processamento de um registro por sistemas computadorizados. A partir da informação existente no Líder e no Diretório, um programa é capaz de processar um registro, uma vez que, esses campos fornecem dados estruturais importantes para a leitura do arquivo. Além disso, a interpretação dos dados existentes em um registro pode ser realizada através de seus *designadores de conteúdo* e das especificações de campo fornecidas pelas instituições que mantêm o MARC.

3 O padrão Z39.50

3.1 O que é Z39.50?

Z39.50 é uma especificação que estabelece um protocolo do tipo cliente-servidor que possibilita consulta e recuperação de informação armazenada em um banco de dados remoto. O Z39.50 foi proposto em 1984 para suprir a necessidade de intercâmbio de informações bibliográficas que existia na época.

Em 1988, a primeira versão foi lançada e, em 1990, um grupo chamado ZIG [7] foi criado para cuidar de sua implementação e especificação, o que resultou no lançamento, em 1992, da segunda versão do Z39.50, sendo a esta adicionadas novas características que culminaram no lançamento de uma nova versão em 1995. A versão de 1988 tornou-se obsoleta e as versões de 1992 e 1995 têm diferenças significativas.

O protocolo estabelecido a partir do Z39.50 especifica formatos e procedimentos para a troca de dados entre um cliente e um servidor, permitindo que um cliente realize buscas e consultas, que atendam a um critério por ele estabelecido, em um banco de dados localizado em um servidor remoto. O cliente pode enviar requisitos no papel de usuário, comunicando-se via protocolo Z39.50 com uma aplicação de um servidor.

Além de simplificar o trabalho de busca, o protocolo Z39.50 permite trabalhar com uma grande quantidade de dados e comunicar com múltiplos sistemas através de interface única.

De modo geral, o Z39.50 permite que um cliente envie uma requisição de consulta, em um ou mais bancos de dados remotos, e pode ou não receber registros como resposta, de acordo com a parametrização da consulta. O Z39.50 permite ao cliente:

- Conhecer previamente o tipo de arquivo recuperado, o tamanho e o custo, por exemplo, antes de fazer sua solicitação.
- Especificar a quantidade de registros que deseja receber como resposta por requisição.
- Especificar a sintaxe dos registros, como por exemplo, o MARC21 ou o USMARC.
- Especificar um rótulo para o conjunto de registros retornados.

Além disso, no servidor (server-side), o Z39.50 permite :

- Controlar o acesso de determinados usuários, que podem ser realizadas por meio de autenticação.
- Controlar o acesso a determinados recursos, reportando ao usuário o status de sua consulta.
- Suspender processos de consulta correntes.

[7] Z39.50 Implementors Group

3.2 Utilização do protocolo

Como dito anteriormente, o protocolo Z39.50 especifica um formato para os dados que devem ser enviados e recebidos, portanto, fica a cargo das aplicações, do servidor e do cliente, a tradução e organização dos dados que devem ser trocados. As mensagens trocadas entre as aplicações são transmitidas em unidades de dados chamadas de PDU[8] que contêm a informação necessária para que a transmissão das mensagens seja realizada.

O processo de troca de mensagens ocorre da seguinte maneira:

- Estabelecimento de uma sessão de comunicação entre as duas aplicações envolvidas no processo, que pode ser feita, por exemplo, através da Internet.
- Envio de uma PDU contendo os parâmetros do serviço, isto é, um conjunto de regras que deverão ser respeitadas por ambas as aplicações durante o processo de requisição e resposta. O resultado deste passo é a criação de uma associação Z39.50 (*Z-association*). Neste contexto, o cliente é chamado de origem da associação e o servidor de alvo. Dentro de uma mesma associação não é permitida uma troca de papéis entre as aplicações, ou seja, origem será sempre a aplicação do cliente e o alvo será sempre a aplicação do servidor. Entretanto, uma aplicação pode estabelecer inúmeras associações com outra, atuando como alvo ou origem em cada uma delas.
- Envio, a partir da origem, de PDUs contendo consultas para o alvo.
- Envio, a partir do alvo, de uma PDU notificando os resultados obtidos pela consulta para a origem.
- Envio, a partir da origem, de uma PDU contendo uma requisição de resposta ou de exclusão para o alvo.
- Caso a origem tenha enviado uma requisição de resposta ao alvo, o resultado da consulta é enviado à origem. Caso contrário, os resultados da busca são desprezados pelo alvo.

3.2.1 Atributos de busca

Os atributos referentes a uma busca devem pertencer a um dos conjuntos de atributos que fazem parte da especificação do protocolo e, portanto, é de responsabilidade da origem da associação atender à especificação. A

[8]Protocol Data Units

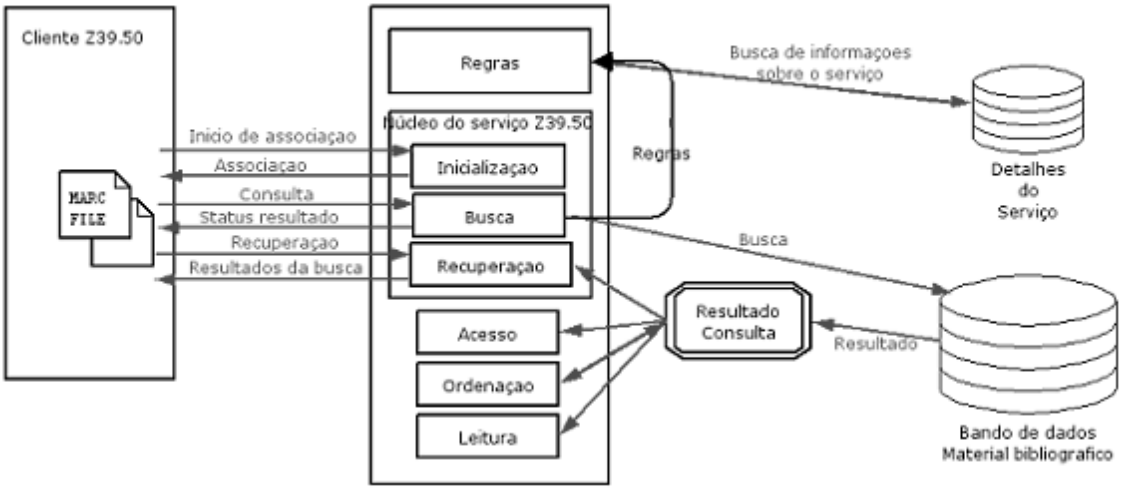


Figura 1: Uma visão geral do processo de troca de mensagens entre origem e alvo

versão 3 do Z39.50 possibilita à origem a utilização de diferentes conjuntos de atributos dentro de uma mesma associação, porém as versões anteriores estabelecem que apenas um conjunto de atributos, escolhido na inicialização da associação, pode ser utilizado. Além disso, as consultas podem ser de dois tipos:

Tipo 0: Requer um acordo privado entre os dois sistemas, que estabelece a forma da consulta e o conteúdo dos resultados que podem ser obtidos.

Tipo 1: Estabelece que as consultas devem ser realizadas em notação polonesa reversa.

3.2.2 Resultados da busca

Os resultados provenientes de uma busca atendem ao formato MARC. O protocolo Z39.50 permite que a origem especifique que extensão do formato MARC deseja-se obter como resposta de uma consulta. Algumas das extensões disponíveis são USMARC, UKMARC e o CANMARC. O protocolo prevê também, a possibilidade de cadastro de novas extensões.

Os resultados obtidos em uma busca podem, dependendo das regras de associação estabelecidas entre origem e alvo, conter códigos de diagnóstico que fazem parte da especificação do protocolo. Alternativamente, também é

possível o cadastro de novos códigos. Para que isso seja possível, deve haver um acordo entre as aplicações envolvidas.

3.3 Conclusões

O protocolo Z39.50 possibilita a busca por dados existentes em bancos de dados remotos, independentemente da plataforma da aplicação do cliente (que realiza a consulta) e da aplicação do servidor (que mantém o banco de dados). O processo de consulta ocorre de modo transparente para os usuários das aplicações, por meio de uma interface única.

4 Implementação

4.1 Tecnologias e bibliotecas utilizadas

Esta seção é uma breve introdução às principais tecnologias utilizadas durante a implementação.

4.1.1 Linguagem Java

Java é uma linguagem de programação desenvolvida inicialmente pela Sun Microsystems, ela é baseada no paradigma de Programação Orientada a Objetos e tem sintaxe similar à sintaxe utilizada em linguagens procedurais. Além da sintaxe, a linguagem Java contém um conjunto de bibliotecas para solução de problemas mais comuns como o tratamento de dispositivos de entrada e saída, estruturas de dados, programação paralela e concorrente, tratamento de erros etc.

Além da linguagem, a Sun Microsystems criou uma plataforma composta de um compilador e uma máquina virtual (JVM), para a qual a linguagem foi projetada. Essa plataforma permite o isolamento entre o programa e o sistema operacional sobre o qual o programa é executado, ao mesmo tempo que é mais eficiente do que seria com o uso de interpretadores.

4.1.2 Hibernate

O Hibernate é um projeto aberto, desenvolvido sob a plataforma Java, que provê um serviço de persistência de dados e suporte a consultas em um banco de dados qualquer[9]. O serviço permite que os desenvolvedores implementem suas classes dentro do paradigma da orientação a objetos e

[9]Na verdade, qualquer banco de dados que implemente um *driver jdbc*, pois o hibernate utiliza as interfaces da especificação jdbc para comunicação com o banco de dados.

realiza a persistência de objetos utilizando-se de um mapeamento entre as classes da aplicação e as entidades que as representam no banco de dados. É, portanto, um arcabouço para o mapeamento objeto-relacional.

O Hibernate também oferece ao desenvolvedor uma linguagem de consultas chamada HQL que permite ao desenvolvedor lidar apenas com as classes por ele definidas, sem nunca fazer referência a tabelas do banco, não necessita de servidor de aplicação e não exige que o objeto que será persistido possua nenhuma característica especial. Para ser persistido um objeto deve fornecer *getters* e *setters* para seus atributos mapeados e um construtor que aceite como parâmetros estes atributos.

É parte do trabalho do desenvolvedor ao utilizar o arcabouço:

- Criar um arquivo xml de configuração que deve indicar quais são as classes da aplicação que representam entidades da base de dados, fornecer informações para a conexão com o a base de dados e definir parâmetros que serão utilizados pelo Hibernate, como por exemplo, a utilização ou não de cache.
- Representar as tabelas que serão utilizadas pela aplicação através de classes.
- Definir o mapeamento entre classes e tabelas do banco.
- Definir e mapear as relações existentes entre as classes.

Existem duas maneiras de realizar o mapeamento entre classes e objetos para possibilitar sua persistência: mapeamento com a utilização de arquivos xml e o mapeamento utilizando a tecnologia *Annotations*, disponível a partir da especificação Java 5, que permite a introdução de meta dados nas classes das quais deseja-se realizar persistência.

4.1.3 Tomcat

O Apache Tomcat é um contêiner para servlet, desenvolvido e mantido pela fundação Apache e é compatível com as tecnologias Java Servlet[10] e Java Server Pages[11]. O Tomcat é licenciado sobre os termos de licença Apache e muito utilizado no mercado. Entre os casos de uso do Tomcat temos o projeto JBoss[12] (que vem com o Tomcat como seu web contêiner padrão).

[10] Aplicação que pode ser executada em um servidor WEB
[11] Tecnologia para desenvolvimento de aplicações WEB
[12] Servidor de aplicações WEB

4.1.4 Biblioteca marc4j

A biblioteca marc4j é um projeto aberto que visa a oferecer aos desenvolvedores de diferentes domínios de aplicações uma interface de programação para trabalhar com arquivos binários no formato MARC e arquivos xml[13] no formato MARCXML dentro da plataforma Java. A biblioteca foi implementada para trabalhar com arquivos que atendam ao formato de algumas extensões do MARC, como Por exemplo, o MARC21 e o UNIMARC.

A biblioteca marc4j inclui:

- Uma interface que possibilita o manuseio de um grande número de registros MARC e MARCXML.
- Objetos que possibilitam a leitura e a escrita de arquivos MARC e MARCXML.
- Suporte para conversão entre arquivos MARC e arquivos MARCXML.

4.1.5 JUnit

O JUnit é um arcabouço para a criação de testes automatizados desenvolvido sob a plataforma Java.

4.2 Modelagem do Banco de Dados do Colméia

Como parte do processo de implementação, foi necessário a realização de um estudo da modelagem do banco de dados do sistema Colméia uma vez que, o banco de dados seria o destino dos dados contidos nos registros MARC fornecidos. Quando o projeto foi iniciado, o banco já havia sido modelado e sofreu mudanças durante todo o semestre. Segue uma descrição da estrutura utilizada pelo módulo Zumbido neste trabalho. As tabelas do banco são referenciadas dentro de parênteses.

Um item do acervo (ITEM_ACERVO) é a base para a inserção de cinco tipos de entidades:

- Livros (LIVRO)
- Teses (TESE)
- Coleções de livros
- Exemplar de periódicos (EXEMPLAR_DE_PERIODICO)

[13] EXtensible Markup Language

- Exemplar de não periódicos (EXEMPLAR_DE_NAO_PERIODICO)

Um item do acervo deve possuir como atributos uma editora (EDITORA), que o publicou, uma língua (LINGUA), na qual este foi editado, um departamento (DEPARTAMENTO), ao qual ele está relacionado, e um assunto principal (ASSUNTO). Além disso ele pode possuir assuntos secundários, subtítulos (SUBTITULO) e um ou mais identificadores (IDENTIFICADOR) que podem ser de dois tipos: ISBN[14] ou ISSN[15]. Eventualmente itens do acervo podem estar relacionados com ocorrências de eventos (OCORRENCIA_EVENTO), com algum tipo de observação (OBSERVACAO) ou com algum tipo de nota (NOTA).

Além de itens do acervo, existem na base de dados mais dois tipos de entidades importantes: artigos (ARTIGO) e materiais multimídia (MULTIMIDIA). Livros, teses, artigos e materiais multimídia devem obrigatoriamente ter um autor principal e podem ter eventuais autores secundários.

Existem no banco também entidades que servem para representar sinônimos de outras entidades do banco, como por exemplo, o sinônimo de uma editora (SINONIMO_EDITORA). Este recurso foi utilizado para que fosse evitada a inserção de duas tuplas numa tabela que representassem o mesmo objeto do mundo real.

A inserção de exemplares depende da periodicidade no qual estes são publicados e do tipo de material ao qual se referem. Exemplares de teses são sempre inseridos na tabela EXEMPLAR_DE_PERIODICO, enquanto exemplares de livros dependem de seu atributo periodicidade.

Os identificadores utilizados para inserção de tuplas nas tabelas (ids) podem ser gerados de duas maneiras, dependendo da tabela: ou são gerados a partir de uma tabela de identificadores (TABELAID), que contém uma tupla referenciando cada tabela, ou por seqüências geradas pelo próprio banco.

4.3 Mapeamento objeto-relacional

Para que fosse possível a utilização do Hibernate como provedor de persistência de dados, foi necessária a criação de classes que representam as entidades, ou tabelas, do acervo presentes na base de dados do sistema. Estas classes foram armazenadas no sub pacote do módulo chamado *entidades*.

Além disso foi necessária a criação de mapeamentos entre cada classe e a tabela da qual fez-se a representação. Durante a disciplina de Programação Extrema a solução escolhida foi o uso de mapeamentos via arquivos xml. Sendo assim, para cada classe, que faz parte do modelo da aplicação, foi

[14]International Standard Book Number
[15]International Standard Serial Number

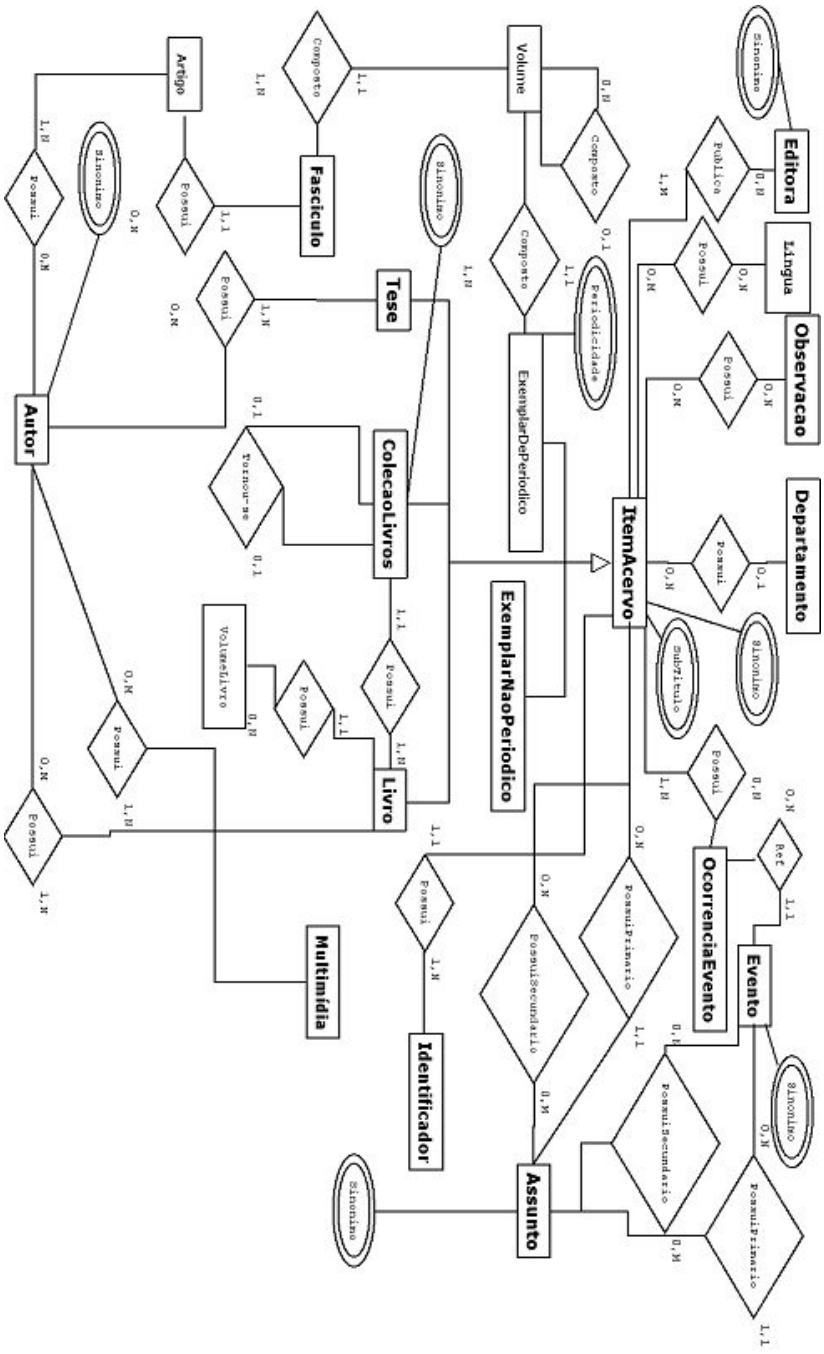


Figura 2: Modelo conceitual do banco do sistema Colméia

criado um arquivo xml contendo o mapeamento objeto-relacional que era usado pelo Hibernate para associar os objetos da aplicação com as tabelas da base de dados. Porém a maneira de realizar este mapeamento foi modificada no segundo semestre. Os arquivos xml foram eliminados e foram adicionados as classes meta dados na forma de Annotations[16], pois desta maneira, quando mudanças ocorressem nas tabelas da base de dados, a manutenção do código seria mais simples estando limitada à modificações apenas no código fonte das classes.

Com o intuito de separar o acesso ao banco do código da aplicação também foi empregado o padrão de desenvolvimento DAO[17]. O emprego do padrão consiste basicamente da criação de classes que contenham métodos capazes de receber objetos do modelo e realizar operações de acesso ao banco utilizando estes objetos como parâmetros. Alguns exemplos de operações são a inserção, remoção e atualização de tabelas contidas na base de dados. Em nosso contexto o emprego do padrão DAO traduziu-se na criação de classes nomeadas da seguinte maneira: NomeDaEntidade+*Home*. Por exemplo, na base de dados do Colméia temos a tabela ITEM_ACERVO que é representada na aplicação pela classe ItemAcervo e têm como classe de acesso ao banco ItemAcervoHome.

4.4 Estrutura do módulo Zumbido

De modo geral o módulo é composto pela biblioteca marc4j, o pacote java br.usp.ime.colmeia.zumbido e o conjunto de bibliotecas que permitem a utilização do arcabouço Hibernate. A maneira como estes subcomponentes se relacionam pode ser visualizada na figura 3.

4.4.1 O pacote br.usp.ime.colmeia.zumbido

O Colméia tem seu código fonte dividido em dois diretórios src/java e o src/test. O diretório src/java contém o código fonte da aplicação e o src/test contém o código fonte dos testes. Esta seção tratará do código fonte da aplicação que foi produzido.

Como já foi dito anteriormente, o código fonte do módulo começou a ser produzido em março de 2007. Foi nesta época que se optou pela utilização da biblioteca marc4j para facilitar a interpretação do arquivo binário no formato MARC com o qual deveria ser realizada a população do banco de dados do Colméia. Porém, um primeiro empecilho surgiu: a biblioteca marc4j era capaz de interpretar de maneira correta apenas arquivos que seguissem

[16] Anotações, disponíveis a partir do Java 5.0
[17] Data Access Object

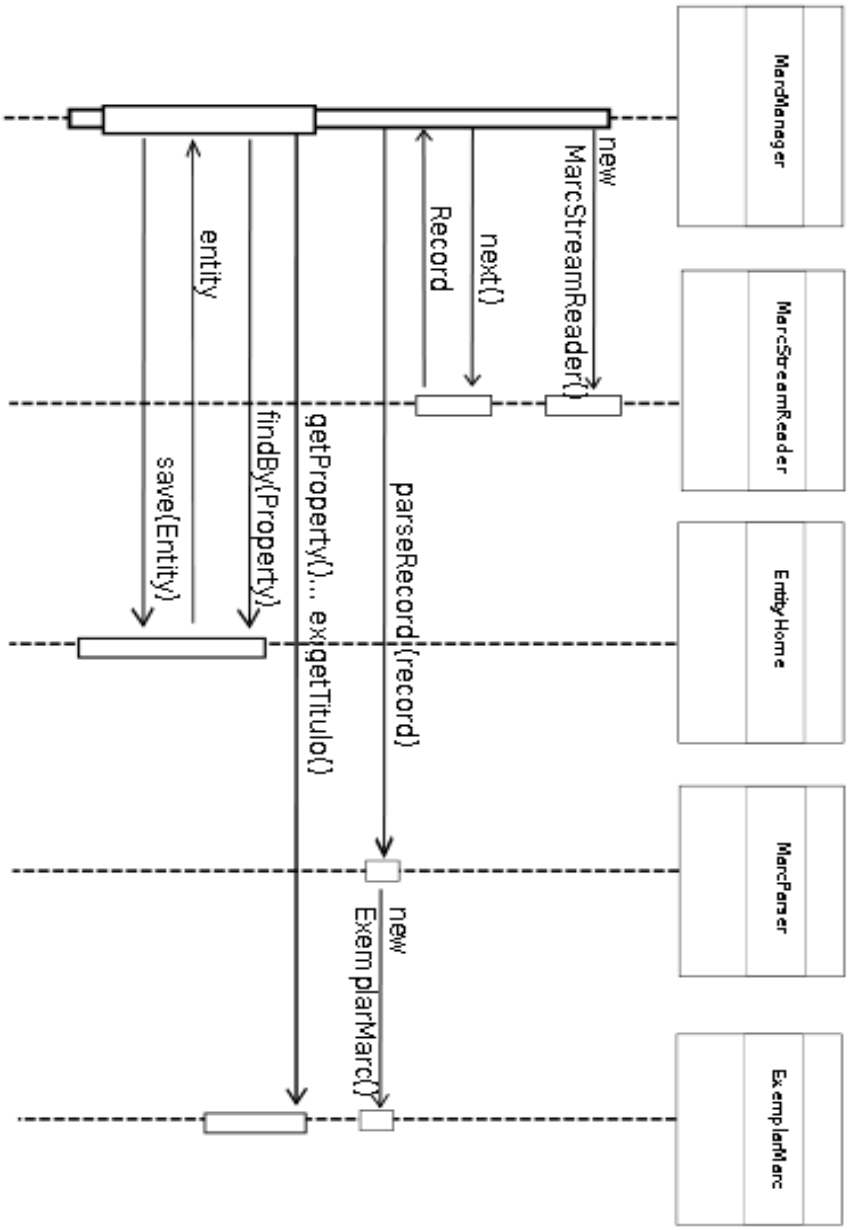


Figura 3: Diagrama de seqüência: importação de dados

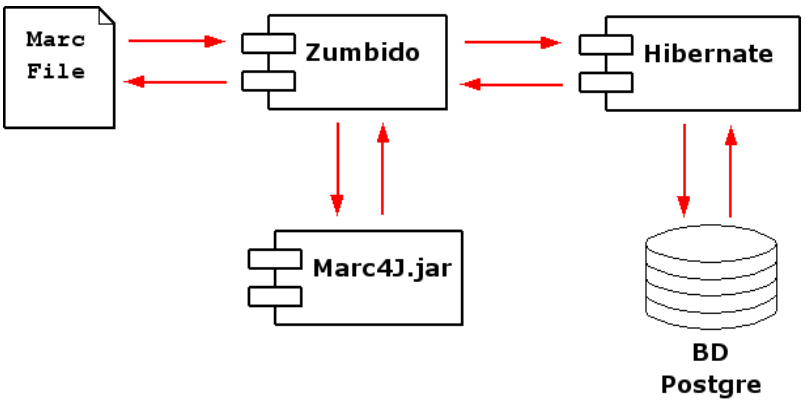


Figura 4: Relacionamento entre subcomponentes

a especificação de um tipo específico de extensão do MARC. Este tipo de extensão deveria ter uma característica existente no formato MARC21, os indicadores existentes nos campos de variável de dados deveriam sempre ocupar a primeira e segunda posição do campo, o que não ocorria com o arquivo fornecido para a população do banco, que tinha, assim como a primeira versão do MARC, a informação da posição dos indicadores no Líder do registro.

Tornou-se necessária então, a criação de uma classe que realizasse essa leitura e esta classe foi chamada de LenientMarcStreamReader. A função desta classe é gerar, a partir de um objeto do tipo InputStream[18], exatamente os mesmos objetos que seriam gerados caso a interpretação do arquivo fosse feita inteiramente pela biblioteca marc4j. Assim o papel da biblioteca marc4j na aplicação é fornecer objetos que encapsulem os dados contidos em um arquivo MARC e forneçam uma interface pública de manipulação dos dados. Quatro tipos de objetos existentes na biblioteca assumiram papel importante dentro do módulo, os objetos Record, DataField, ControlField e o objeto Subfield.

Para que fosse possível a utilização do Hibernate como provedor de persistência de dados, foi necessária a criação das classes que representariam as entidades alvo do banco de dados e um mapeamento entre elas e suas respectivas tabelas. Durante a disciplina de Programação Extrema a solução escolhida foi o uso de mapeamentos via arquivos xml. Sendo assim, para cada tabela do banco foi criada uma classe, que é parte do modelo da aplicação, um arquivo xml com o mapeamento objeto-relacional e uma classe DAO[19] nomeada com o sufixo *Home*. As classes DAO fornecem uma interface de persistência para a classe modelo correspondente, contendo métodos de acesso

[18]pacote java.io.InputStream
[19]Data Access Object

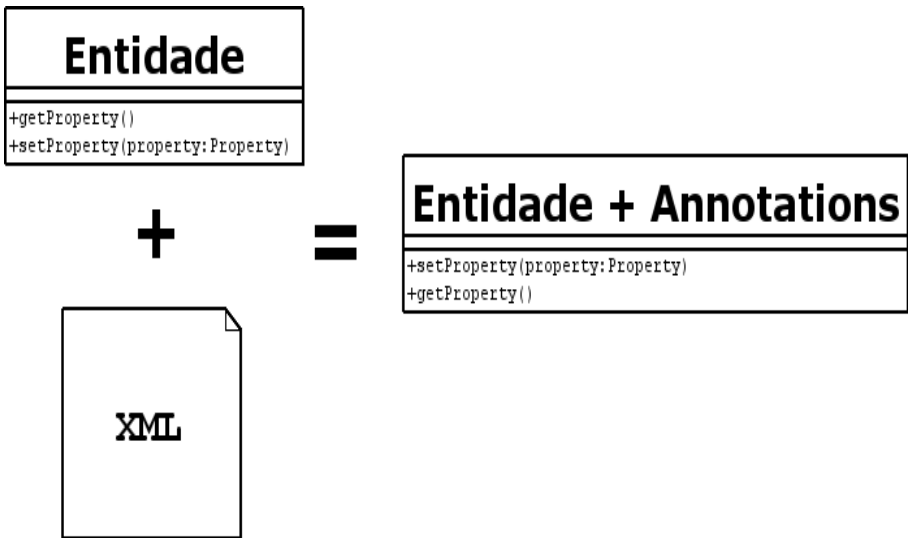


Figura 5: Equivalência entre os tipos de mapeamentos

ao banco de dados. Esta arquitetura sofreu modificação no segundo semestre. Os arquivos xml foram eliminados e foram adicionados às classes meta dados na forma de annotations[20], o que facilitou o processo de manutenção do código. As classes do modelo juntamente com as classes de acesso ao banco de dados estão no subdiretório *entidades*.

4.4.2 Importação de dados

Para a realizar a exportação era preciso encontrar as informações, que pertenciam ao modelo do banco de dados, nos registros formatados no padrão MARC. Foi criada a classe MarcParser que é responsável pela análise sintática do conteúdo do arquivo e extração das informações que devem ser armazenadas no banco de dados do sistema. Neste momento novos problemas surgiram, pois não era claro quais campos do arquivo MARC continham os dados necessários para realizar a importação, e além do mais os dados não haviam sido armazenados de maneira idêntica para todos os registros. Por exemplo, os marcadores 490 e 440 são destinados às informações a respeito da série ou coleção a qual um item de acervo pertence. Em alguns registros esta informação aparecia no campo indexado pelo marcador 440 e em outros no campo indexado pelo marcador 490. A partir deste ponto um estudo mais detalhado da estrutura do MARC começou a ser feito e foram necessárias pesquisas dentro da biblioteca do IME com auxílio dos funcionários e a uti-

[20]Disponível a partir do Java 5.0

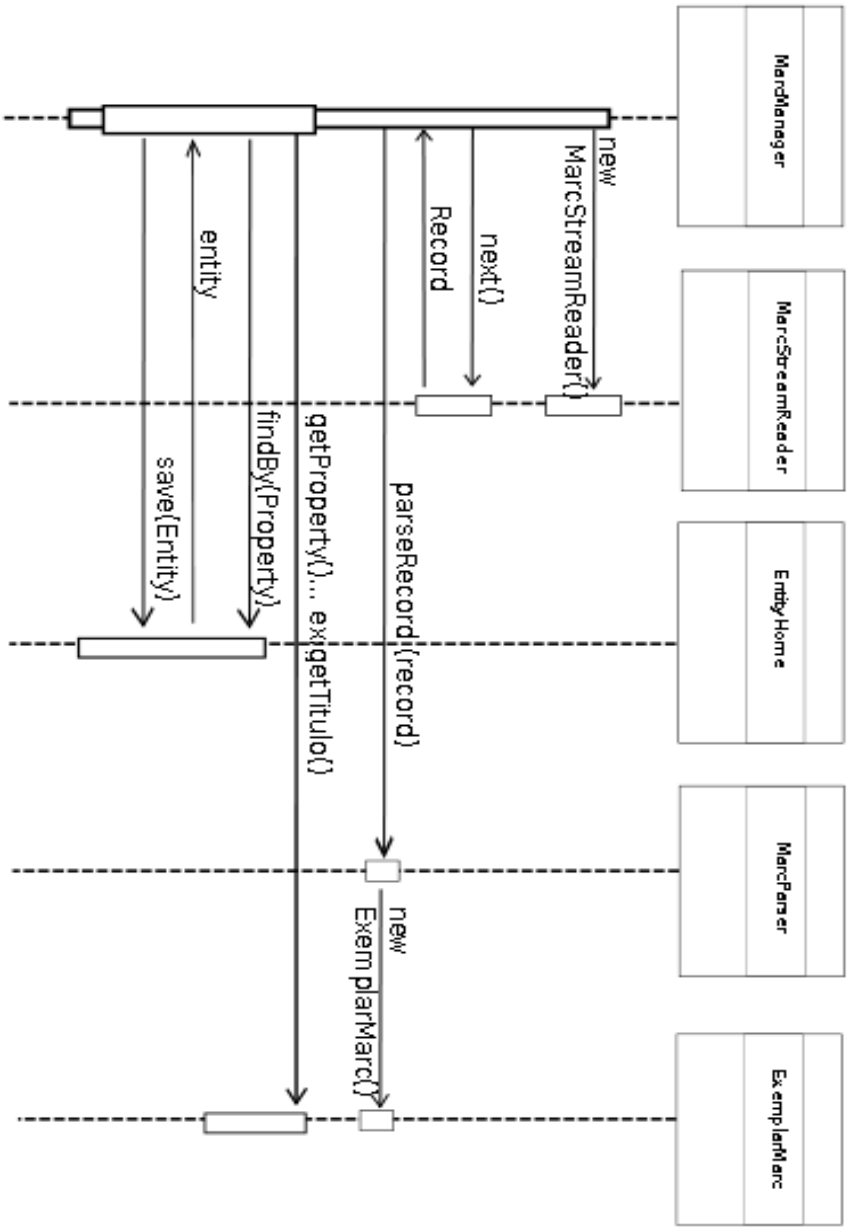


Figura 6: Diagrama de seqüência: importação de dados

lização do sistema Dedalus[21]

Além disso, algumas informações necessárias para inserção de um exemplar, como Por exemplo, o número do exemplar e a localização deste na biblioteca, estavam armazenadas em alguns campos que são definidos localmente pela instituição responsável pelo catálogo do item do acervo e eram subpartes de outras informações, para obter estas informações foi necessário o uso de expressões regulares.

Assim, a importação de registros de obras a partir de um arquivo MARC é constituido pelas classes:

- MarcParser - extração das informações (como explicado anteriormente);
- ExemplarMarc - encapsula uma obra do acervo e todas as suas informações;
- MarcManager - gerencia a criação de todos os objetos que devem ser inseridos a partir de um registro;
- Entidades - classes de mapeamento objeto-relacional;
- EntityHome - classes de fornecem as chamadas para o arcabouço de persistência;

A importação de dados pode ser feita através da interface *WEB* do sistema.

4.4.3 Exportação de dados

Para realizar a exportação foi implementada a classe ExportManager. A classe ExportManager tem como função delegar às classes DAO a busca, no banco de dados, de todas as informações a respeito de uma obra que atenda a um critério de seleção especificado pelo usuário do sistema através da interface web. Por exemplo, a busca por um exemplar não periódico deverá retornar não apenas um objeto do tipo ExemplarNaoPeriodico, como também o livro, editora, assuntos, notas, autores etc, relacionados ao exemplar.

De posse das informações, a classe ExportManager obtém instâncias da classe Record através da classe MarcFactory, que implementa o padrão de *design* Factory. Os objetos do modelo são então utilizados para a criação de registros no formato MARC, que são encapsulados nos objetos Record e finalmente, através da biblioteca marc4j o arquivo MARC é gerado. A exportação de obra para um arquivo binário MARC no Colméia pode ser

[21]Banco de Dados Bibliográficos da USP - Catálogo On-line Global

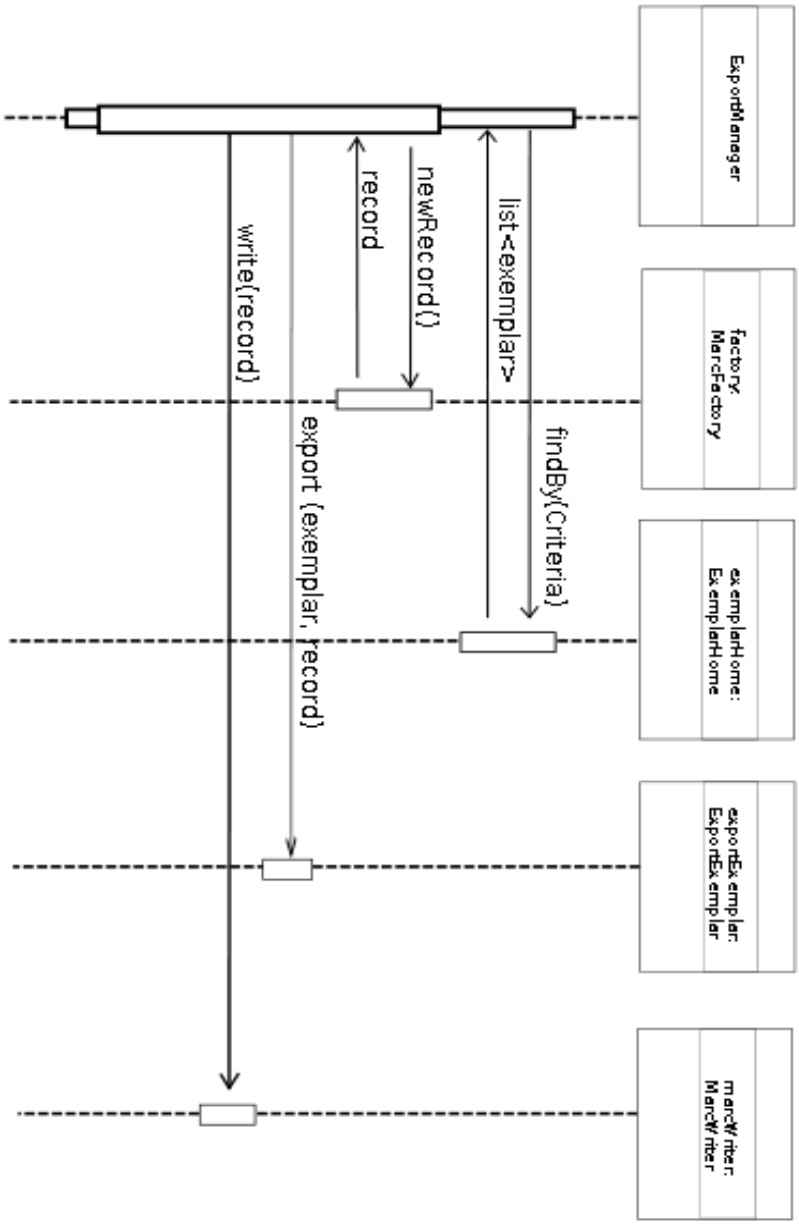


Figura 7: Diagrama de seqüência: exportação de dados

visualizada na figura 6 que apresenta um diagrama de seqüência ilustrando a rotina de exportação de dados.

A exportação de dados pode ser feita através da interface *WEB* do sistema.

4.4.4 Log do sistema

O sistema é capaz de gerar arquivos de log que permitem ao usuário saber como foi a inserção de obras no banco a partir de um arquivo binário MARC ou a exportação de registros provenientes do banco. Após a importação o usuário pode, requisitar pela interface do sistema *WEB* o *download* dos arquivos de log gerados.

4.4.5 Testes unitários

Foram criadas diversas rotinas de testes para o módulo Zumbido. Estas rotinas estão no pacote br.usp.ime.colmeia.zumbido dentro do diretório src/test. Para a criação dos testes foi utilizado o arcabouço de criação de testes automatizados JUnit. As rotinas de testes desempenharam papel fundamental no desenvolvimento do módulo, pois existiam outros sub projetos sendo desenvolvidos concorrentemente com o módulo Zumbido. Esse desenvolvimento de sub projetos paralelos tinha como conseqüência a modificação de entidades e relações entre elas existentes na base de dados do sistema. Os testes unitários permitiram a detecção imediata destas modificações e possibilitaram a manutenção imediata do código.

4.5 Método de desenvolvimento

Como foi dito anteriormente, este projeto surgiu a partir das aulas da disciplina de Programação Extrema, ministrada pelo professor Fábio Kon. Dessa forma, as técnicas desta metodologia foram aplicadas em todo o desenvolvimento do trabalho.

Programação Extrema é uma metodologia de desenvolvimento ágil para equipes de pequeno ou médio porte. A metodologia inclui 12 técnicas principais de desenvolvimento, das quais pode-se aplicar subconjuntos, dependendo do contexto de trabalho e necessidades da equipe de desenvolvimento. As 12 técnicas são:

- Jogo do Planejamento: São realizadas interações entre os desenvolvedores e os clientes. Em cada interação os clientes expõem novas funcionalidades que gostariam que fossem acrescentadas à aplicação, o

desenvolvedor analisa a possibilidade de implementação na nova funcionalidade naquele momento e caso esta seja possível, a nova funcionalidade é descrita em cartões contendo sua descrição na forma de tarefas que o usuário pode realizar. Tarefas podem ser divididas em subtarefas. Os desenvolvedores devem estimar o tempo necessário para a implementação das novas funcionalidades.

- Fases Pequenas: As interações entre clientes e desenvolvedores devem ser constantes, de modo que, a cada interação, novas funcionalidades são apresentadas aos clientes que especificam novas funcionalidades que serão apresentadas na próxima interação. Para que seja possível interação constante, as tarefas especificadas não devem ser muito grandes.

- Metáfora: Metáforas são importantes para aumentar a comunicação entre clientes e desenvolvedores e muitas vezes ajudam o desenvolvedor no planejamento da arquitetura do software.

- Design Simples: A modelagem e arquitetura do sistema devem ser simples. Não é necessária a utilização de uma arquitetura sofisticada se esta não for indispensável ou trazer benefícios para a aplicação.

- Testes: A implementação de uma funcionalidade só está completa depois que o desenvolvedor preparou uma rotina de testes para ela. A criação de testes deve fazer parte do cotidiano do programador XP.

- Refatoração: A refatoração do código deve ser constante e deve ocorrer sempre que necessária. Essa prática permite que o código mantenha-se organizado e que no final do projeto o código da aplicação seja reutilizável e de fácil entendimento.

- Programação Pareada: Os programadores devem trabalhar em pares utilizando um mesmo micro. Esta técnica diminui a chance de criação de código ruim ou errado, além de possibilitar o surgimento de idéias mais elaboradas.

- Propriedade Coletiva: Não existe código de apenas um programador. Uma classe escrita por um programador pode ser modificada por outro. O código deve estar sempre disponível em um repositório e é dever de todos os programadores o envio do código produzido todos os dias.

- Integração Contínua: As mudanças devem ser integradas continuamente, ou seja, não deve haver muita diferença entre a versão da aplicação apresentada ao cliente e a versão sob a qual os programadores trabalham.

- Semana de 40 Horas: Os programadores não devem trabalhar mais do que 40 horas semanais, pois sua produtividade e rendimento podem ser influenciados pelo cansaço e indisposição.

- Cliente junto aos Desenvolvedores: Os clientes devem estar sempre que possível junto aos desenvolvedores para possibilitar as fases pequenas de desenvolvimento.

- Padronização do Código: Uma vez que o código é de propriedade coletiva, é necessário que este siga um padrão mantendo-se organizado e compreensível para todos os membros da equipe.

Após o término da disciplina optou-se pela manutenção das seguintes técnicas aprendidas: design simples, testes, refatoração, integração contínua e padronização de código.

O código fonte foi todo editado dentro do ambiente de desenvolvimento integrado Eclipse, o repositório de código fonte escolhido foi o *Subversion* e como ferramenta de acesso ao banco de dados utilizamos o *PgAdmin* e o *plugin* do *Eclipse SQLExplorer*. A ferramenta de acesso ao banco de dados *PgAdmin* foi muito importante, pois durante o desenvolvimento foi necessário o estudo também das *Stored Procedures* que realizavam inserção de itens dos quais o cadastro é feito pela interface *WEB* do sistema.

5 Conclusões

O módulo Colméia-Zumbido é parte fundamental do projeto Colméia. Ele começou a ser implementado neste ano e deve, nos próximos semestres, crescer muito mais. Alguns dos objetivos esperados foram atingidos, porém há muito trabalho a ser feito. Espera-se que no futuro o módulo seja capaz de gerar arquivos binários no formato Marc com registros mais completos e homogêneos. Para que isso seja possível a modelagem das entidades que representam o acervo da biblioteca deve ser ampliada para que um número maior de informações a respeito dos materiais do acervo possam ser armazenadas. O estudo realizado sobre o formato Marc e o protocolo Z39.50 deixou claro que o sistema Colméia deve, além de automatizar as tarefas realizadas em uma biblioteca universitária, integrar-se com outros sistemas bibliográficos, o que possibilitará ao sistema o intercâmbio de informações e capacidade de atualização de seus dados. O ideal seria que o módulo não apenas oferecesse uma interface de consulta para outras bases bibliográficas, mas que também permitisse que outros sistemas realizassem consultas remotas em seu banco de dados.

Referências

[1] http://colmeia.incubadora.fapesp.br/portal.

[2] http://eclipsesql.sourceforge.net.

[3] http://itsmarc.com/crs/bib1468.htm.

[4] http://marc4j.tigris.org.

[5] http://subversion.tigris.org.

[6] http://www.cbr.washington.edu/ camel/z/z.html.

[7] http://www.eclipse.org.

[8] http://www.hibernate.org.

[9] http://www.ibict.br/cionline/viewarticle.php?id=430.

[10] http://www.libraryhq.com/z3950qa.html.

[11] http://www.loc.gov/marc/.

[12] Kent Beck and Martin Fowler. *Planning Extreme Programming.* Addison-Wesley Professional, October 2000.

[13] Betty Furrie. *Understanding Marc Bibliographic: Machine-Readable Cataloging.* Cataloging Distribution Service, 6th edition, 2001.