UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIA E TECNOLOGIA
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DISSERTAÇÃO DE MESTRADO

DILIFRAME: UM FRAMEWORK PARA O DESENVOLVIMENTO DE BIBLIOTECAS DIGITAIS

ELIANE CRISTINA DE ARAÚJO

Campina Grande – PB
Novembro de 2002

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIA E TECNOLOGIA
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DILIFRAME: UM FRAMEWORK PARA O DESENVOLVIMENTO DE BIBLIOTECAS DIGITAIS

ELIANE CRISTINA DE ARAÚJO

Dissertação apresentada à coordenação de Pós-graduação em informática – COPIN – da Universidade Federal de Campina Grande – UFCG, como requisito parcial para a obtenção do grau de Mestre em Informática.

Orientador: Cláudio de Souza Baptista

Campina Grande – PB
Novembro de 2001

Ficha catalográfica

ARAÚJO, Eliane Cristina
A658D

Diliframe: Um Framework para o Desenvolvimento de Bibliotecas Digitais

Dissertação (Mestrado) - Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande, Paraíba, Novembro de 2002.

198 p. Il.

Orientador: Cláudio de Souza Baptista, Ph.D

Palavras Chaves:

1. Bancos de Dados Bibliotecas Digitais
2. Bibliotecas Digitais
3. Engenharia de Software
4. Framework

CDU - 681.3.07B

"DILIFRAME: UM FRAMEWORK PARA O DESENVOLVIMENTO DE BIBLIOTECAS DIGITAIS"

ELIANE CRISTINA DE ARAÚJO

DISSERTAÇÃO APROVADA EM 29.11.2002

PROF.CLAUDIO DE SOUZA BAPTISTA, Ph.D
Orientador

PROF. MARCUS COSTA SAMPAIO , Dr.
Examinador

PROF. ED PORTO BEZERRA, D.Sc
Examinador

CAMPINA GRANDE – PB

“A imaginação é mais importante que o conhecimento. O conhecimento é limitado. A imaginação abraça todo o mundo”

Albert Einstein

Dedico este trabalho a meus pais: Elizabete e Mário.

Agradecimentos

Aos meus pais e irmãos pela confiança.

Aos meus amigos e amigas do mestrado pelo incentivo.

Ao meu orientador pelo empenho.

Aos novos amigos pelo suporte.

A Harrison e Glaucimar pela ajuda nos mais momentos difíceis.

Sumário

AGRADECIMENTOS III

SUMÁRIO IV

LISTA DE FIGURAS VIII

LISTA DE TABELAS X

RESUMO XI

ABSTRACT XII

CAPÍTULO 1 1

INTRODUÇÃO 1

1.1 OBJETIVO 3

1.2 RELEVÂNCIA 3

1.3 ESTRUTURA DA DISSERTAÇÃO 3

CAPÍTULO 2 5

EMBASAMENTO TEÓRICO 5

2.1 BIBLIOTECAS DIGITAIS 5
2.1.1 Fundamentos e Conceitos 10
2.1.2 Desafios e Questões 11
2.1.2.1 Arquitetura de Bibliotecas Digitais 16
2.1.2.2 Desafios 18
2.2 FRAMEWORKS 21
2.2.1 Fundamentos e Conceitos 24
2.2.2 Desafios e Questões 28

CAPÍTULO 3 32

DILIFRAME – UM FRAMEWORK PARA BIBLIOTECAS DIGITAIS 32
3.1 APRESENTAÇÃO 32
3.2 ASPECTOS CONCEITUAIS 34
3.2.1 Análise dos Requisitos 34
3.2.2 Análise de Casos de Uso 42
3.3 ASPECTOS ARQUITETURAIS 57
3.3.1 Modelo de Metadados 57
3.3.2 Modelo arquitetural 63
3.3.2.1 A arquitetura da Dilib 69
3.3.2.2 A arquitetura da MetadataMaker 75
3.3.2.3 A arquitetura da RepositoryMaker 79

CAPÍTULO 4 83

ASPECTOS DO PROJETO DO DILIFRAME 83
4.1 PROJETO DO DILIFRAME 83
4.1.1 A Dilib 84
4.1.1.1 View Assembler 89
4.1.1.2 Controller 92
4.1.1.3 Collection 101
4.1.1.4 Repository 104
4.1.1.5 User 108
4.1.1.6 Login 113

4.1.1.7 Searcher 116
4.1.1.8 Retriever 124
4.1.2 A MetadataMaker 128
4.1.2.1 Cliente 129
4.1.2.2 MetaMaker 132
4.1.2.3 MetaMakerBD 134
4.1.3 A RepositoryMaker 137
4.1.3.1 Cliente 138
4.1.3.2 RepMaker 140
4.1.3.3 RepMakerBD 142
4.2 ESTRATÉGIA PARA IMPLEMENTAÇÃO 143

CAPÍTULO 5 148

CONCLUSÃO 148
5.1 PERSPECTIVAS 149
5.2 RESULTADOS 150

REFERÊNCIAS BIBLIOGRÁFICAS 151

ANEXO A 163

BREVE ANÁLISE DE PROJETOS EXISTENTES 163
1. New Zealand Digital Library 163
2. Alexandria Digital Library 165
3. Renardus 167
4. Greenstone 168

ANEXO B 170

MODELO PARA DESCRIÇÃO DE CASOS DE USO 170

ANEXO C 175

O PADRÃO DE METADADOS DUBLIN CORE **175**
O que é o Dublin Core? 175
Princípios do Dublin Core e seus elementos 176

ANEXO D **183**

ARQUITETURA DE ALTO NÍVEL DO DILIFRAME **183**

Lista de Figuras

FIGURA 1 - ARQUITETURA SIMPLIFICADA DE BIBLIOTECAS DIGITAIS 17
FIGURA 2 - ATORES NAS BIBLIOTECAS DIGITAIS 18
FIGURA 3 - REUSO NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE 22
FIGURA 4 - REUSO COM BIBLIOTECAS DE CLASSE 22
FIGURA 5 - INVERSÃO DE CONTROLE 28
FIGURA 6 – REPOSITÓRIO DE DADOS E APLICAÇÃO EM UM SÓ SERVIDOR 36
FIGURA 7 – REPOSITÓRIO DE DADOS E APLICAÇÃO EM SERVIDORES DISTINTOS 37
FIGURA 8 – VÁRIOS REPOSITÓRIOS DE DADOS ACESSADOS POR UMA APLICAÇÃO 38
FIGURA 9 - RELACIONAMENTOS: REGISTRO DE METADADOS X COLEÇÕES E RECURSOS 58
FIGURA 10 – CENTRALIZAÇÃO DOS REGISTROS DE METADADOS 59
FIGURA 11 - SUB-APLICAÇÕES DO DILIFRAME 66
FIGURA 12 - DIAGRAMA DE CASOS DE USO DA DILIB 67
FIGURA 13 - DIAGRAMA DE CASOS DE USO DA *METADATAMAKER* 68
FIGURA 14 - DIAGRAMA DE CASOS DE USO DA *REPOSITORYMAKER* 69
FIGURA 15 - MÓDULOS FUNCIONAIS DA DILIB 70
FIGURA 16 - FILTRO DE INTERCEPTAÇÃO 72
FIGURA 17 - RELACIONAMENTO ENTRE OS DADOS DO *USER* 73
FIGURA 18 - MÓDULOS FUNCIONAIS DA *METADATAMAKER* 77
FIGURA 19 - MÓDULOS FUNCIONAIS DA *REPOSITORYMAKER* 81
FIGURA 20 - APLICAÇÃO J2EE MAIS O FRAMEWORK PARA APLICAÇÕES WEB 87
FIGURA 21 - COMPOSIÇÃO ARQUITETURAL DA DILIB 89
FIGURA 22 - PÁGINA FORMADA A PARTIR DA COMPOSIÇÃO DE *VIEWS* 90
FIGURA 23 - SEQÜÊNCIA DE AÇÕES NO FUNCIONAMENTO DO VIEWASSEMBLER 92
FIGURA 24 - CICLO DE ATIVIDADE NA CAMADA WEB 93
FIGURA 25 - DIAGRAMA DE ESTADOS DO WAF 94
FIGURA 26 - DIAGRAMA DE CLASSE HTMLACTION 96
FIGURA 27 - DIAGRAMA DE CLASSE WEBCONTROLLER 98
FIGURA 28 - DIAGRAMA DE INTERAÇÃO WAF 100
FIGURA 29 - REPRESENTAÇÃO GRÁFICA DO WAF 101
FIGURA 30 - DIAGRAMA DE CLASSES DO ENTITY BEAN COLLECTION 103
FIGURA 31 - DIAGRAMA DE INTERAÇÃO DA OPERAÇÃO "CREATE COLLECTION" 104

FIGURA 32 - DIAGRAMA DE CLASSES DAO GENÉRICO DO *REPOSITORY* 105
FIGURA 33 - DIAGRAMA DE INTERAÇÃO DA OPERAÇÃO "CREATE REPOSITORY " 107
FIGURA 34 - DIAGRAMA DE COMPONENTES DO *ENTITY BEAN USER* 108
FIGURA 35 - DIAGRAMA DE CLASSES DO *ENTITY BEAN USERPROFILE* 110
FIGURA 36 - DIAGRAMA DE CLASSES DO *ENTITY BEAN USERDETAILS* 111
FIGURA 37 - DIAGRAMA DE INTERAÇÃO DA OPERAÇÃO "CREATE USER" 113
FIGURA 38 - DIAGRAMA DE ESTADOS DO MÓDULO *LOGIN* 114
FIGURA 39 - DIAGRAMA DE CLASSES DO *SESSION BEAN LOGIN* 116
FIGURA 40 - DIAGRAMA DE CLASSES DO *ENTITY BEAN METADATA* 117
FIGURA 41 - DIAGRAMA DE COLABORAÇÃO *FASTLANEREADER* 120
FIGURA 42 - DIAGRAMA DE COLABORAÇÃO *VALUELISTHANDLER* 121
FIGURA 43 - DIAGRAMA DE CLASSES DO *SESSION BEAN RESOURCE* 125
FIGURA 44 - DIAGRAMA DE INTERAÇÃO DA OPERAÇÃO "*GETRESOURCE"* 127
FIGURA 45 - DIAGRAMA DE INTERAÇÃO DA OPERAÇÃO "*LOGIN"* 129
FIGURA 46 - DIAGRAMA DE INTERAÇÃO DA OPERAÇÃO "*INSEREREGISTRO*" 130
FIGURA 47 - DIAGRAMA DE INTERAÇÃO DA OPERAÇÃO "*ATUALIZAREGISTRO"* 131
FIGURA 48 - DIAGRAMA DE INTERAÇÃO DA OPERAÇÃO "*EXCLUIREGISTRO"* 132
FIGURA 49 - DIAGRAMA DE COLABORAÇÃO DO *METADATAMAKER* 134
FIGURA 50 - DIAGRAMA DE INTERAÇÃO DA OPERAÇÃO "*VALIDATE"* 136
FIGURA 51 - DIAGRAMA DE COLABORAÇÃO DA *REPOSITORYMAKER* 138
FIGURA 52 - DIAGRAMA DE INTERAÇÃO DA OPERAÇÃO DE "*LOGIN"* 139
FIGURA 53 - DIAGRAMA DE INTERAÇÃO DA OPERAÇÃO "*INSERERECURSO"* 140
FIGURA 54 - ARQUITETURA DA ADEPT 166
FIGURA 55 - ARQUITETURA DO RENARDUS 168
FIGURA 56 – MODELO DE APLICAÇÃO CLIENTE/SERVIDOR 185
FIGURA 57 – MODELO DE APLICAÇÃO MULTI-CAMADAS 185
FIGURA 58 – MODELO DE COMPONENTES J2EE 188
FIGURA 59 – EJB IMPLANTADO EM UM CONTAINER 190
FIGURA 60 – EMPACOTAMENTO DE MÓDULOS DE APLICAÇÃO J2EE 192
FIGURA 61 – DEPLOYMENT DESCRIPTOR 193
FIGURA 62 – ARQUITETURA MVC 195
FIGURA 63 – PADRÃO DE PROJETO J2EE: INTERCEPTING FILTER 196
FIGURA 64 – PADRÃO DE PROJETO J2EE: VIEW HELPER 196
FIGURA 65 – PADRÃO DE PROJETO J2EE: COMPOSITEVIEW 197
FIGURA 66 – PADRÃO DE PROJETO J2EE: SESSION FAÇADE 197
FIGURA 67 – PADRÃO DE PROJETO J2EE: BUSINESS DELEGATE 198

Lista de Tabelas

TABELA 1 - BIBLIOTECAS TRADICIONAIS *VERSUS* BIBLIOTECAS DIGITAIS 8
TABELA 2 - EXEMPLOS DE BIBLIOTECAS DIGITAIS 10
TABELA 3 - REUSO NOS FRAMEWORKS 23
TABELA 4 - DOMÍNIOS DOS FRAMEWORKS 29
TABELA 5 - TÉCNICAS DE DOCUMENTAÇÃO 31
TABELA 6 – REQUISITOS FUNCIONAIS 40
TABELA 7 – REQUISITOS NÃO-FUNCIONAIS 42
TABELA 8 – ATORES QUE INTERAGEM COM A BIBLIOTECA DIGITAL 44
TABELA 9 – CASOS DE USO CONSIDERADOS 45
TABELA 10 – CONVENÇÕES DE UTILIZAÇÃO DO DUBLIN CORE NO DILIFRAME 63
TABELA 11 – IDENTIFICADOR DO RECURSO 78
TABELA 12 - VANTAGENS DA UTILIZAÇÃO DE UM FRAMEWORK DE APLICAÇÕES WEB 88
TABELA 13 - AÇÕES HTML E RESPECTIVAS URLS 97
TABELA 14 - EVENTOS DA DILIB 98
TABELA 15 - AÇÕES EJB X EVENTOS RESPOSTA 99
TABELA 16 - FUNCIONALIDADES DA CLASSE *USEREJBACTION* 109
TABELA 17 - OPERAÇÕES DO *METAMAKER* 134
TABELA 18 – AÇÕES DO *REPMAKER* 141
TABELA 19 - CONSIDERAÇÕES NA PRIORIZAÇÃO DE MÓDULOS PARA IMPLEMENTAÇÃO 144
TABELA 20 - DESCRIÇÃO DE CICLOS DE ITERAÇÃO 147
TABELA 21 – MODELO DE CASO DE USO 171
TABELA 22 - CARACTERÍSTICAS DO PADRÃO DUBLIN CORE 176
TABELA 23 - ELEMENTOS DO DUBLIN CORE 177
TABELA 24 – APIS CORPORATIVAS JAVA 186

Resumo

Este trabalho apresenta o Diliframe – um *framework* para o desenvolvimento de bibliotecas digitais. O seu objetivo é facilitar a construção deste tipo de sistema e diminuir o tempo gasto com desenvolvimento através do reuso promovido pelo *framework*. O seu projeto flexível pode adaptar-se às necessidades das aplicações do domínio das bibliotecas digitais. Construído em cima da plataforma Java para aplicações corporativas – J2EE, o Diliframe pode garantir escalabilidade e confiabilidade para as suas aplicações. Além disso, abrange diversos aspectos da criação de bibliotecas digitais tais como: administração de dados, administração de usuários, consulta e recuperação de recursos, criação de repositórios e criação de metadados.

Abstract

This dissertation presents Diliframe: a framework for the development of digital libraries. The aim is to facilitate the development of these systems through reuse achieved with the framework. Diliframe is build upon a flexible design and architecture and can easily suit the needs of applications in digital libraries domain. This framework is based on Java 2 Enterprise Edition Platform – J2EE, and can provide scalability, reliability to its derived applications. Furthermore, this framework covers various aspects of digital libraries development, such as: data administration, users administration, search and resource retrieval, repository creation and metadata creation.

Capítulo 1

Introdução

Cada vez mais a informação digital se torna presente em nossa vida diária. Instituições, centros de pesquisa e universidades estão trabalhando para que seja ampliada a camada da população beneficiada pelas mudanças trazidas pela "Era da Informação". As bibliotecas digitais desempenham um importante papel neste cenário: "Estes sistemas disponibilizam acesso coerente de uma comunidade de usuários a uma grande quantidade de informação e conhecimento organizados em repositórios" [Garcia-Molina, 1995].

Vários países estão adotando estratégias para a criação ou o melhoramento de suas bibliotecas digitais. Dentre outros benefícios, estas iniciativas trazem à população a possibilidade de resgatar obras criadas por artistas e autores de seus países. As bibliotecas representam um símbolo nacional, símbolo de sua história e de sua cultura. Atuando no campo da educação, proporcionam a troca de informações entre universidades, centros de pesquisas e instituições.

A *National Science Foundation* – NSF, agência governamental de fomento a pesquisa norte-americana, coordena o projeto *Digital Library Initiative* naquele país, desde a década de noventa. Diversos estudos e pesquisas vêm sendo desenvolvidos e milhões de dólares investidos ao longo de anos. O maior desafio no qual os pesquisadores estão trabalhando refere-se à interoperabilidade entre bibliotecas digitais [FOX, 1999].

A Fundação Biblioteca Nacional, vinculada ao Ministério da Cultura do Brasil, desenvolve o projeto Biblioteca Digital Multimídia. Como parte desta iniciativa, foi lançado o Portal da Biblioteca Digital Multimídia. O portal engloba outros projetos já existentes, como a Biblioteca sem Fronteiras [BibSFront, 2002]. Através deste, a população tem acesso a inúmeros recursos, incluindo: livros, gravuras, partituras, áudio, vídeo, etc. O projeto inclui, ainda, a criação de uma rede de alta velocidade que fará a interconexão entre

mais de cem bibliotecas. O principal objetivo do projeto é democratizar o acesso à informação [FBN, 2002].

Bibliotecas digitais têm evoluído rapidamente [Arms, 2001]. Há pouco tempo, tais sistemas eram apenas um conjunto de metadados indexados em algum sistema gerenciador de banco de dados, associados a *links* indicando onde determinados recursos poderiam ser encontrados. Ademais, estas bibliotecas eram puramente textuais. Hoje em dia, existe a demanda pelo armazenamento do próprio conteúdo digital e não apenas de seus metadados [deVries 1998].

Bibliotecas Digitais envolvem atores, serviços e recursos. Os atores são os provedores de dados, provedores de metadados, administradores e usuários. Os serviços incluem busca por informação, cobrança, cadastro de usuários, segurança e proteção de direitos autorais, dentre outros. Os recursos são os documentos nas mais diversas mídias.

O desenvolvimento deste tipo de sistema requer pessoal de informática bem qualificado. Bibliotecas digitais são aplicações não triviais, nas quais estão envolvidas diversas variáveis para assegurar seu desempenho, usabilidade e manutenibilidade. Destarte, é bastante salutar o estudo de técnicas que possibilitem facilitar o processo de criação destes sistemas. O enfoque dado neste trabalho para lidar com esta necessidade é a utilização de frameworks.

Os frameworks são essencialmente uma técnica de reuso orientada a objetos [Fayad, 1999]. São aplicações semiprontas em cima das quais novas aplicações, do mesmo domínio, podem ser construídas. Sob a ótica da reusabilidade, utilizando frameworks, atinge-se o reuso em três dimensões:

- Reuso de Análise: Todo o entendimento do problema e a definição de um conjunto de entidades e termos específicos do domínio já foram feitas no framework. As novas aplicações apenas seguirão o que já foi estabelecido.
- Reuso de Projeto: A arquitetura definida, as interações entre os componentes e todo o conjunto de classes projetado para o framework são pontos de partida para a nova aplicação a ser gerada. Até mesmo as interfaces externas dos novos componentes a serem criados, já foram preestabelecidas pelo projeto do framework.

- Reuso de Código: Quando uma nova aplicação é criada através da extensão do framework as subclasses reutilizam o código existente da superclasse. Além disso, alguns frameworks oferecem recursos que podem ser diretamente utilizados pelos desenvolvedores.

1.1Objetivo

O objetivo deste trabalho é desenvolver a arquitetura e o projeto de um framework para a construção de bibliotecas digitais. O framework, denominado Diliframe, deve auxiliar o desenvolvedor na realização de seu projeto, minimizando o tempo de desenvolvimento através do reuso por ele possibilitado.

O público a que se destina o framework é formado por desenvolvedores de bibliotecas digitais, tanto públicas quanto comerciais, que desejam aproveitar-se de um projeto lógico já concebido como base para a construção do seu. O desenvolvedor deverá ser capaz de estender a aplicação através da adição de novos elementos ou através da modificação de elementos existentes, de acordo com as regras de negócio da sua aplicação.

1.2 Relevância

A importância deste trabalho está em propor uma arquitetura reutilizável para o desenvolvimento de bibliotecas digitais que englobe todos os seus processos, tais como: a criação, utilização e manutenção de seus dados.

Além disso, outro ponto considerado relevante é a exploração da plataforma J2EE para a construção do projeto. Esta plataforma define uma arquitetura Java unificada, baseada em componentes, centrada em serviços e habilitada para aplicações multi-camadas. Ela oferece diversas vantagens na construção de aplicações corporativas.

1.3 Estrutura da dissertação

A presente dissertação está organizada da seguinte forma:

- **Capítulo 1**: Faz uma apresentação geral do contexto da dissertação e de seus objetivos;
- **Capítulo 2**: Contempla o embasamento teórico necessário para o bom entendimento do problema discutido neste trabalho;

- **Capítulo 3**: Apresenta a solução oferecida com o Diliframe e o caminho traçado para a sua concepção;
- **Capítulo 4**: Apresenta os aspectos do projeto e da implementação do Diliframe;
- **Capítulo 5**: Apresenta as conclusões obtidas com a avaliação deste trabalho e aponta os caminhos para a evolução do mesmo.

Capítulo 2

Embasamento teórico

Este capítulo tem como objetivo apresentar um embasamento teórico sobre os temas que são discutidos ao longo desta dissertação. Além disso, visa a esclarecer os conceitos e o vocabulário de referência que serão utilizados para a apresentação deste projeto.

Com este intuito, o capítulo divide-se em duas seções. A primeira apresenta o contexto histórico das bibliotecas digitais, seus fundamentos e seus desafios. A segunda aborda a tecnologia dos frameworks, fundamentando a escolha da abordagem para a execução deste projeto.

2.1Bibliotecas Digitais

Os esforços no sentido de utilizar computadores para armazenar informações de bibliotecas remontam à década de sessenta. Entretanto, muitas eram as barreiras tecnológicas enfrentadas para que tais iniciativas obtivessem sucesso. Além do proibitivo custo para o armazenamento e processamento, as interfaces a caracteres não ofereciam maior atrativo que a manipulação dos objetos físicos.

O desenvolvimento tecnológico alcançado na década de noventa proporcionou a derrubada dos obstáculos que impediam a popularização e o desenvolvimento adequado das bibliotecas digitais. Dentre os quais podemos citar: 1) Redução no custo do armazenamento eletrônico; 2) Dispositivos gráficos de apresentação mais agradáveis; 3) Popularização dos computadores pessoais; 4) Redes de computadores de alta velocidade e de amplo alcance; 5) Alto poder de processamento de computadores pessoais.

A possibilidade de viabilizar o acesso mais amplo aos itens da biblioteca foi um dos fatores que mais influenciou no desenvolvimento de pesquisas nesta área. O acesso às informações de qualidade ainda é um privilégio de poucos, não obstante a Internet muito tenha contribuído para a disseminação e ampliação do acesso às informações. Com relação às bibliotecas tradicionais, as digitais ampliam o acesso dos usuários à medida que permite acesso, permanente, aos dados enquanto a maioria das bibliotecas está fechada, quer seja durante a noite, em feriados ou finais de semana; ou mesmo quando se está a vários de quilômetros de distância da biblioteca propriamente dita.

É importante ressaltar que as bibliotecas digitais devem ser entendidas como uma evolução qualitativa e quantitativa das bibliotecas tradicionais. A possibilidade de armazenar grande quantidade de informação em espaços reduzidos e do compartilhamento de coleções entre bibliotecas geograficamente dispersas aumenta de forma quantitativa o acervo disponibilizado. A diversidade de formatos (multimídia) em que pode ser encontrada a informação e a possibilidade do oferecimento de novos tipos de serviços caracterizam o aspecto qualitativo desta evolução.

É insuficiente afirmar que as bibliotecas digitais são as versões eletrônicas das bibliotecas tradicionais. Além de agregar novos conceitos e incorporar novos serviços e capacidades, um público diferente do que era atingido pelas tradicionais é servido pelas bibliotecas digitais. Parte destas pessoas não teria acesso às informações disponibilizadas pela biblioteca devido às suas limitações geográficas. Outra parte pode ser constituída por jovens pesquisadores que já nasceram na era da informação digital e que por comodidade dão preferência à manipulação e pesquisa de recursos digitais. Para fins de análise, será apresentada, a seguir, a evolução cronológica das bibliotecas em termos de gerações [Baptista, 2000].

- **Primeira Geração: Bibliotecas Tradicionais**

Nas bibliotecas da primeira geração, todo o serviço é realizado sem a utilização de sistemas computacionais de informação. O acervo da biblioteca é catalogado manualmente em cartões. A pesquisa é feita de modo manual. Os usuários devem preencher cartões onde estarão contidas suas informações pessoais, e informações sobre seus empréstimos e reservas.

Os recursos disponibilizados pelas bibliotecas tradicionais geralmente são os seguintes: livros, revistas, jornais, jogos, mapas, material de áudio e vídeo. Os serviços oferecidos por elas são: empréstimo, reserva, pesquisas e a estrutura para o acesso físico ao acervo.

Questões sobre propriedade intelectual dos recursos são claramente definidas nas leis de *copyright* específicas para bibliotecas. Para as bibliotecas tradicionais, as noções de empréstimo e de reserva são importantes, pois os recursos são físicos e limitados.

- **Segunda Geração: Bibliotecas com Sistemas de Informação**

Nesta geração, as bibliotecas utilizam sistemas computacionais específicos para a informatização de seus serviços. Os cartões utilizados nas bibliotecas tradicionais são convertidos para meio eletrônico.

Há pacotes de software com o propósito de informatizar a indexação e pesquisa no acervo, largamente utilizada por bibliotecas em todo o mundo, como, por exemplo, o Open Public Access Library - OPAC. Há uma preocupação com a produção de metadados e, mais ainda, para que seja estabelecido um padrão na produção de metadados. Isto permitiria que tais sistemas de indexação pudessem interoperar com outros do mesmo gênero. Diante deste quadro surge o padrão MARC e suas variações [MARC, 2002].

- **Terceira Geração: Bibliotecas Digitais**

As bibliotecas digitais agregam mais uma categoria de serviço às bibliotecas tradicionais. Além da busca e indexação, apresenta-se agora a recuperação da informação. Isto é possível, pois nesta geração o acervo da biblioteca é digital. Assim, após uma busca, a informação é recuperada a partir da própria máquina do cliente. Este tipo de sistema constrói uma idéia de biblioteca bastante distinta daquela apresentada nas gerações anteriores. Conceitos importantes, como reserva e disponibilidade de recursos, não fazem mais sentido na manipulação de recursos digitais.

Alguns estudiosos acreditam que, por alguns anos, coexistirão recursos digitais e recursos físicos nas bibliotecas desta geração. Além disso, é, no mínimo, precipitado declarar a extinção das bibliotecas tradicionais com o advento das bibliotecas digitais. A grande massa de informação produzida pelo homem está catalogada nas coleções existentes

em milhares de bibliotecas espalhadas pelo mundo. A transposição de tudo isso para meio magnético ou digital é indesejável e inviável.

- **Quarta Geração: Bibliotecas Digitais Multimídia**

A quarta geração de bibliotecas agrega à terceira diferentes tipos de recursos tais como: áudio, vídeo, mapas e imagens, além dos documentos tipo texto. A estes diversos formatos nos quais a informação pode se apresentar dá-se o nome de multimídia. O processo de busca e indexação de informação multimídia é diferente do realizado para atributos de metadados textuais. Para as informações multimídia, utiliza-se a técnica de recuperação baseada em conteúdo, a ser detalhada mais adiante. Nesta abordagem, o processo de busca é diretamente dependente do tipo de informação que está sendo manipulado: imagem, som, texto e vídeo.

A Tabela 1, apresenta um resumo comparativo entre as bibliotecas tradicionais e as digitais:

Bibliotecas Tradicionais	Bibliotecas Digitais
Recursos físicos	Recursos digitais
Número limitado de recursos	Número ilimitado de recursos
Instalações físicas	Ambiente virtual
Grande corpo de funcionários	Corpo reduzido de funcionários
Número limitado de usuários	Número ilimitado de usuários
Expediente determinado	Aberta 24 horas
Regras de segurança bem definidas	Regras de segurança difíceis de implementar
Geograficamente limitada	Pode ser globalmente distribuída
Reserva e não recuperação da informação	Recuperação e não reserva da informação

Tabela 1 - Bibliotecas Tradicionais *versus* Bibliotecas Digitais

As iniciativas mais importantes para a construção de bibliotecas digitais tiveram raízes nos Estados Unidos. Em 1994, foi lançado um grande projeto, o *Digital Library Initiative*, envolvendo seis grupos de pesquisa nas seguintes universidades americanas: Universidade da Califórnia (Berkeley e Santa Barbara), Universidade de Standford, Universidade de Illinois e Universidade Carnegie-Mellon. O projeto teve como

objetivo "buscar a evolução dos métodos de captação, armazenamento e organização da informação em formato digital e torná-los disponíveis para busca, recuperação e processamento via redes de comunicação de forma amigável para o usuário" [DLI1, 1994]. Ele foi financiado por alguns órgãos do governo americano sob a coordenação da National Science Foundation – NSF. Hoje, este projeto encontra-se em sua segunda fase, envolvendo várias universidades e com um foco de atuação bem mais abrangente que o seu antecessor preocupando-se com os efeitos sociais trazidos pelas bibliotecas digitais em várias áreas [Fox, 1999].

Atualmente, diversos sistemas de bibliotecas digitais estão em funcionamento pelo mundo. Bons exemplos são as disponibilizadas via Web pela *Association for Computer Machinery* – ACM e pelo *Institute of Eletrical and Eletronic Engineers* – IEEE. O acervo de ambas, restrito a documentos tipo texto, é formado portrata-se de artigos científicos publicados em conferências e revistas destas instituições. Aos usuários, são oferecidos serviços de navegação em toda a coleção, utilizando palavras-chave e restrições de campo. A cobrança pela utilização do material é feita através de uma assinatura anual.

A Tabela 2, lista algumas bibliotecas digitais de variadas áreas de conhecimento acessíveis via Web.

Projeto	Instituição
Informedia Digital Video Library http://www.informedia.cs.cmu.edu/	Carnegie Mellon University
DeLIVer – Desktop link to Virtual Engineering Resources http://dli.grainger.uiuc.edu/	University of Illinois,Urbana-Champaign (UIUC)
Alexandria Digital Earth Prototype http://alexandria.sdc.ucsb.edu/	University of California, Santa Barbara
UC Berkeley Digital Library Project http://elib.cs.berkeley.edu/	University of California, Berkeley
Infobus – Stanford Digital Library Technologies Project http://diglib.stanford.edu/	Stanford University
Digital Library Project http://www.si.umich.edu/UMDL/	University of Michigan
Networked Computer Science Technical Reports Library http://www.ncstrl.org/	Cornell University
Networked Digital Library of Theses and Dissertations	Virginia Polytechnic Institute and State University

http://www.ntdl.org/	
The New Zealand Digital Library Project http://www.cs.waikato.ac.nz/~nzdl/	University of Waikato
ACM Digital Library http://www.acm.org/dl/	Association for Computing Machinery
IEEE Digital Library http://computer.org/dlsearch.htm	Institute of Eletrical and Eletronic Engineereing

Tabela 2 - Exemplos de Bibliotecas Digitais

2.1.1 Fundamentos e Conceitos

Embora os estudos e pesquisas na área de bibliotecas digitais não sejam tão recentes, não há, ainda, uma definição consensual para este conceito. Em [Garcia-Molina, 1995], diz-se que "bibliotecas digitais são sistemas que viabilizam o acesso coerente de uma comunidade de usuários a uma grande quantidade de informação e conhecimento organizada em um repositório". Já [Reddy, 1999] define que "uma biblioteca digital é um conjunto integrado de serviços para capturar, catalogar, armazenar, pesquisar, proteger e recuperar informações".

Além do mais, para cada uma das comunidades em que estão segmentadas as pesquisas e os interesses em bibliotecas digitais, a sua definição assume um enfoque diferente. Para as comunidades da ciência da computação, dentre elas a de bancos de dados e a de recuperação da informação, a preocupação é a construção do sistema. Para a comunidade de ciência da informação, o foco é o conteúdo veiculado e as relações sociais existentes neste contexto. Já a comunidade de negócios enxerga nas bibliotecas digitais a viabilização tecnológica do mercado mundial da informação.

É importante que sejam definidos alguns conceitos básicos, no escopo de bibliotecas digitais, que farão parte do linguajar a ser utilizado no decorrer deste trabalho. O esclarecimento destes conceitos é a chave para o entendimento dos problemas deste domínio. Os conceitos de recursos, coleções, acervo e metadados são os seguintes:

Recursos: Os recursos das bibliotecas digitais são os itens que constituem o seu acervo. Diversos termos são utilizados, em outros trabalhos, indistintamente com a mesma finalidade, como por exemplo: item, material, objeto, documento, etc. Nesta dissertação, um recurso representará um item da biblioteca, independentemente de seu tipo: quer seja uma fotografia, uma gravura, um texto, uma partitura, ou qualquer outro.

Coleções: As coleções são agrupamentos de recursos sob uma característica comum. Um acervo de biblioteca digital é constituído de pelo menos uma coleção. As coleções podem estar centralizadas no servidor da biblioteca ou distribuído entre vários servidores. Neste último caso, as coleções podem ser compartilhadas por diversas bibliotecas.
Acervo: O acervo da biblioteca é definido pelo conjunto que envolve todas as coleções que ela disponibiliza para consulta e recuperação.
Metadados: Nas bibliotecas são armazenados dados e metadados. Os dados são os recursos em forma digital que são disponibilizados pela biblioteca. Os metadados existem para caracterizar os recursos aos quais eles se referem. Dentre as categorias nas quais se enquadram os metadados encontram-se: os metadados descritivos (usados para a descrição dos recursos), metadados estruturais (apresentam informações sobre o formato dos dados) e metadados administrativos (apresentam informações sobre *copyrights* e direitos autorais) [Sheth, 1998]. Metadados são bastante importantes para as bibliotecas digitais, por isso, este tema ainda será abordado com mais detalhes na seção 2.2.

2.1.2 Desafios e Questões

Do ponto de vista do desenvolvimento de sistemas de bibliotecas digitais, vários questionamentos devem ser feitos. As definições assumidas em cada uma destas questões caracterizarão a biblioteca a ser construída. São elas: captação do conteúdo, criação ou extração de metadados, armazenamento, pesquisa e recuperação da informação, proteção do acervo, cobrança, dentre outros.

A arquitetura das bibliotecas digitais mostra como estas questões são implementadas e os níveis de relacionamento entre elas. As definem como se processam as operações do negócio, como os componentes tecnológicos se ajustam no sistema e como eles interagem uns com os outros [Reddy, 1999]. A seguir, é apresentada uma breve discussão sobre alguns destas questões que fazem parte do funcionamento dos sistemas de bibliotecas digitais.

- **Captação do conteúdo**

As bibliotecas digitais podem oferecer uma grande diversidade na formato dos recursos de suas coleções. As informações podem ser: imagens, vídeo, arquivos de áudio, jogos, simulações, documentos tipo texto, páginas Web, etc. A tecnologia das bibliotecas digitais possibilita a distribuição de informação multimídia.

Com relação à captação do conteúdo, uma preocupação presente em alguns projetos é o processo de digitalização do recurso que não "nasceu digital" [1]. Aqui, não se trata apenas de usar um *scanner* para digitalizar imagens ou textos. São decisões tecnológicas que devem ser tomadas de acordo com a missão do sistema. Por exemplo, uma resolução de 640 por 480 pixels de uma imagem de uma coluna grega pode ser adequada para um estudante secundarista, mas não suficiente para um estudante de arquitetura que precisa analisar a imagem em detalhes.

- **Criação ou extração de metadados**

Os metadados, que fornecem informações sobre os dados, desempenham um papel muito importante nos processos de busca e recuperação das informações, principalmente, no contexto das bibliotecas digitais [Sheth, 1998]. Através da criação de metadados, é possível organizar e classificar as coleções, cujos conteúdos podem se apresentar nas mais diversas formas: estruturado ou não, em tamanhos variados e em diferentes formatos.

Há uma tênue divisão entre dados e seus metadados. Por exemplo, o *abstract* de um artigo científico é considerado metadado ou parte do próprio dado? De acordo com [Voisard, 1998], não há uma distinção intrínseca entre dados e metadados, pois, dependendo do contexto, um item pode ser considerado dado ou metadado.

A extração de metadados pode ser feita de pelo menos três formas: manual, automática e semi-automática [Baptista, 2000]. Na extração manual, um profissional especializado é responsável por capturar a informação semântica que descreve o dado. Na extração automática, técnicas de casamento de características (*feature matching techniques)* são usadas para capturar a informação desejada. Por exemplo, nos documentos formatados segundo o padrão SGML ou XML, o DTD (*Document Type Definition*) sob o qual o

[1] O termo "nasceu digital", do inglês *born digitally,* foi popularizado pelos orgãos *Digital Library Federation* e o *Council for Library and Information Resources* (CLIR).

documento é construído representa o seu esquema de metadados. Na extração semi-automática, há a participação da máquina junto com a intervenção humana. Por exemplo, o diagnóstico médico através da análise de tomografia computadorizada.

Metadados podem ser utilizados como um mecanismo para viabilizar a interoperabilidade, ou seja, permitir a troca de informações entre sistemas distintos. Para tanto, diversos padrões de metadados foram propostos e estão sendo utilizados com este objetivo. Podemos citar:

- MARC – amplamente utilizado por bibliotecas tradicionais para a descrição de seus catálogos [MARC, 2002];
- FGDC (*US Federal Geographic Data Committee*) – propõe um grande conjunto de terminologias e definições para a documentação de dados espaciais [FGDC, 2002];
- Dublin Core – Cada vez mais popular, este padrão foi criado para a descrição de objetos digitais dispostos na Web utilizando apenas quinze elementos [DC, 2002];
- MPEG-7 (*Multimedia Content Description Interface*) – lançado em 1996, especifica um conjunto padrão de descritores que pode ser usado para descrever vários tipos de informações multimídia [MPEG7, 2002].

A discussão sobre metadados é crucial para permitir uma evolução das bibliotecas tradicionais (com seu catálogo caro e complexo), passando pela Web (com ausência quase completa de catalogação e metadados), para um ambiente razoável no qual metadados são disponibilizados para todos os tipos de objetos digitais (as bibliotecas digitais) [Fox, 1999b].

- **Armazenamento**

O sistema de armazenamento da biblioteca digital deve ser capaz de suportar uma grande quantidade de dados, nos mais variados formatos e acessá-los tão rápido quanto possível [IEEE, 2000]. Dificuldades adicionais são encontradas no armazenamento de recursos de imagem, áudio e vídeo. Devido ao volume de espaço ocupado por estes recursos, técnicas de compressão devem ser utilizadas. Alguns dos formatos de compressão mais utilizados são: MPEG (para vídeo), MP3 (para áudio) e JPEG (para imagens).

Os recursos disponibilizados pelas bibliotecas digitais podem estar armazenados em bancos de dados orientados a objeto, relacionais ou objeto-relacionais;

podem estar distribuídos ou centralizados ou, ainda, dispersos na Web acessíveis via URL. É desejável que a biblioteca digital possa integrar cada uma dessas abordagens, isto porque são variados os tipos de recursos manipulados e eles podem adequar-se melhor em estruturas diferentes. Por exemplo, os metadados, que são bem estruturados, podem ser armazenados em bancos de dados relacionais. Já a manipulação de mapas e imagens pode ser feita em bancos de dados orientados a objetos ou objeto-relacionais.

- **Pesquisa e recuperação da informação**

As técnicas de recuperação da informação em bibliotecas digitais variam de acordo com o tipo de recurso que está sendo pesquisado. As principais técnicas utilizadas são: pesquisas usando atributos de metadados, recuperação baseada em conteúdo e a recuperação textual.

Nas consultas baseadas em atributos de metadados, a informação é recuperada com base em determinadas características previamente extraídas do recurso. Por exemplo, palavras-chave, título, autor, descrições, etc [Voisard, 1998].

Na recuperação baseada em conteúdo, a abordagem de consulta utilizada para a recuperação da informação é baseada na semelhança, ao contrário da abordagem tradicional que busca a combinação exata da chave de pesquisa. Por exemplo, considera um banco de dados de criminosos suspeitos, cujo conteúdo são fotografias. Em uma consulta, seria lançado o retrato falado de um suspeito. O sistema devolveria uma lista dos registros que mais se aproximariam do padrão pesquisado. Algumas características utilizadas para o reconhecimento de padrões na pesquisa por conteúdo seriam: textura, cor, forma, espaço, tempo, som, dentre outras.

Nos sistemas de recuperação textual, os documentos são armazenados na forma de conjuntos de palavras-chave e oferecem consultas em uma linguagem de consulta baseada em palavras ou partes de palavras com o auxílio de conectores e operadores lógicos [Yu, 1994]. Esta abordagem é adequada para a manipulação de documentos tipo texto.

O sucesso do processo de recuperação da informação pode ser medido pelos parâmetros *recall* e *precision* [Salton, 1988]:

Recall – é a habilidade do sistema de encontrar a maior quantidade possível de informação relevante, dada pela fórmula:

$$Recall = \frac{No_RecursosRelevantesRecuperados \times 100}{No_TotalRecursosRelevantes}$$

Precision – é a habilidade do sistema de selecionar apenas o que é relevante excluindo os itens irrelevantes;

$$Precision = \frac{No_RecursosRelevantesRecuperados \times 100}{No_TotalRecursosRecuperados}$$

Por exemplo, seja uma base de dados que contém 100 documentos. Destes, 50 contêm a palavra "tumor". Se ao fazer uma busca pela palavra tumor, o sistema retornar 30 itens dos quais apenas 20 contêm a palavra tumor, então o *recall* será de 40% (20/50) e a *precision* é de aproximadamente 67% (20/30).

Em um sistema de recuperação ideal, os dois parâmetros tenderiam ao valor de 100%. Entretanto, por serem índices ortogonais, esta situação é impossível. Então, deve-se buscar o equilíbrio entre o *recall* e o *precision* de modo que eles sejam maximizados. Tão maior e equilibrados estejam estes valores, tão mais eficiente e eficaz será considerado o processo de recuperação da informação.

- **Proteção do acervo**

Com relação à proteção do acervo, dois aspectos interdependentes influenciam na pesquisa e desenvolvimento de sistemas de bibliotecas digitais que são: os direitos de propriedade intelectual (*copyright*) e a segurança da autoria da informação [Marchionini, 1998].

Como manter os direitos sobre a propriedade intelectual dos recursos disponibilizados pela biblioteca digital, é um dos maiores desafios para os desenvolvedores destes sistemas. Garantir que a autoria do recurso não será violada. Por exemplo, um usuário apropriar-se de um recurso e assiná-lo como se fosse de sua própria autoria, é outra questão ainda mais complexa. Muitas técnicas vêm sendo desenvolvidas para combater a pirataria no mercado da informação.

As marcas d'água digitais (*digital watermarks*) são utilizadas para este fim. A idéia é marcar o item de forma que facilmente o autor possa identificar a marca d'água e que seja impossível ao usuário removê-la. A técnica de *digital watermarking* é bastante usada em imagens, mas também pode ser aplicada, de forma mais difícil, a textos [Lesk, 1997].

- **Cobrança**

Diversas modalidades de cobrança pela aquisição dos recursos têm sido praticadas: assinatura anual na biblioteca digital, cobrança por recurso adquirido (por exemplo: por artigo ou por imagem), por uso do sistema (por exemplo: por uso de CPU, por tempo de conexão, por pesquisa realizada) [Baptista, 2000].

2.1.2.1 Arquitetura de Bibliotecas Digitais

Os sistemas de bibliotecas digitais são, geralmente, constituídos por módulos, que integrados, desempenham e viabilizam as suas operações. As arquiteturas destes sistemas definem, entre outras coisas, de que forma será feita esta integração a fim de atingir o melhor desempenho possível.

A Figura 1 ilustra, em alto nível, a arquitetura simplificada de uma biblioteca digital em três camadas. A primeira é a interface com o usuário, através da qual ele interage com o sistema a fim de utilizar seus serviços. A segunda representa os módulos responsáveis por viabilizar os serviços oferecidos. A terceira responde pelo armazenamento tanto dos dados quanto dos metadados do sistema.

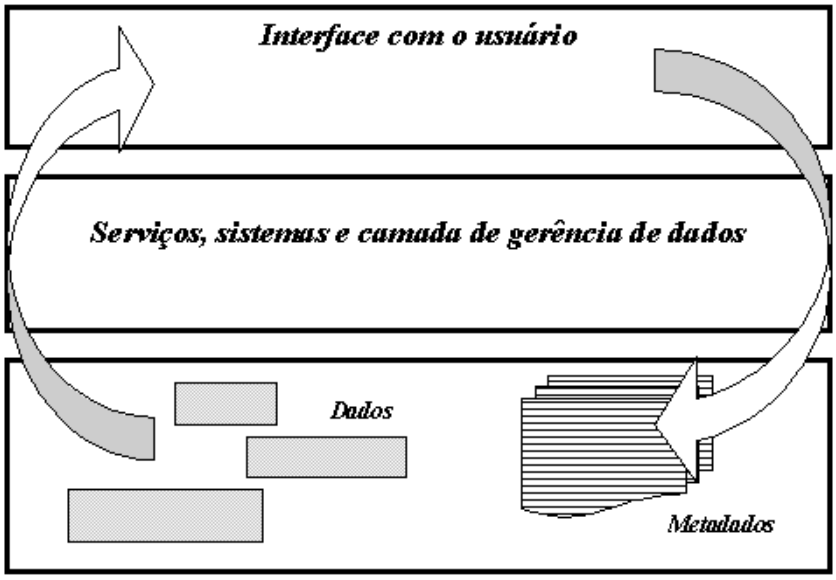


Figura 1 - Arquitetura simplificada de bibliotecas digitais

Com relação aos atores que interagem com o sistema de biblioteca digital, podemos classificá-los em grupos independentes [Vries,1998]:

- **Provedores de conteúdo** – Responsáveis pelos dados a serem distribuídos pela biblioteca digital;
- **Provedores de acesso ao conteúdo ou administradores** – Responsáveis pela gerência da biblioteca digital. Pode existir mais de um grupo de provedores de conteúdo para o sistema;
- **Anotadores ou provedores de metadados** – Responsáveis pela produção de metadados. Como visto anteriormente, eles podem ser atores humanos ou não;
- **Usuários finais** – São aqueles a quem se destina o sistema. Os usuários finais realizam pesquisas, navegam nas coleções e utilizam outros serviços disponibilizados pelo sistema.

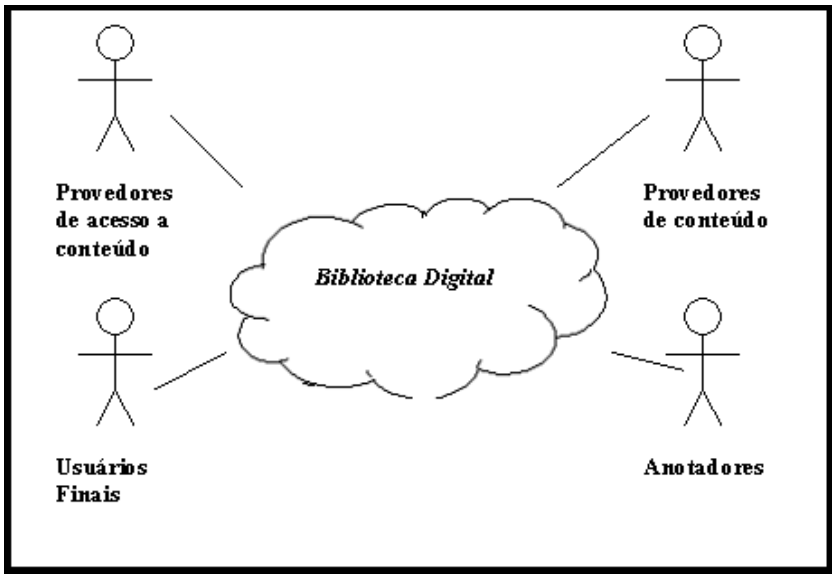


Figura 2 - Atores nas Bibliotecas Digitais

2.1.2.2 Desafios

A natureza dos sistemas de bibliotecas digitais é multidisciplinar e sem limites geográficos definidos [Schäuble,1998]. Estas características também se refletem nas pesquisas para o desenvolvimento destes sistemas. Grandes dificuldades para uma evolução coerente da área devem-se a esta fragmentação das pesquisas em diversas comunidades de estudo e em centros de pesquisas dispersos.

No sentido de minimizar estes problemas e direcionar os esforços dos pesquisadores, para que avanços concretos nos projetos de bibliotecas digitais sejam atingidos, têm sido promovidos encontros internacionais onde são definidas "agendas de pesquisa". Para a produção destes documentos, pesquisadores discutem as recomendações e os desafios encontrados na área. Como resultado, tem-se uma lista com os principais temas em torno dos quais devem-se somar esforços para a obtenção de alternativas para a evolução da área e resolução de problemas.

Abaixo, estão alguns temas recorrentes em agendas de pesquisa para bibliotecas digitais [Garcia-Molina, 1995; Schäuble, 1998]:

- **Interoperabilidade**

No contexto das bibliotecas digitais, interoperabilidade refere-se à capacidade do sistema de usar coleções e serviços de outras bibliotecas digitais de forma transparente para o usuário. Assim, por exemplo, em uma consulta, um usuário poderia recuperar informação de fontes diferentes.

Desde o início das pesquisas em bibliotecas digitais, a interoperabilidade aparece como um dos maiores desafios. As principais questões da área são: desenvolvimento de modelo de informação utilizando metadados, definição de mecanismos para pesquisas distribuídas (incluindo busca em bancos de dados heterogêneos) mantendo a mesma consistência semântica e a escolha do mecanismo de implementação (tecnologia que viabiliza o desenvolvimento de sistemas de computação distribuídos).

Ainda não há soluções definitivas para as questões apontadas, apenas propostas mais ou menos bem sucedidas. Um exemplo que vem sendo alcançado pela comunidade é o *Open Archive Initiative* cujo objetivo é "permitir pesquisa e recuperação de artigos científicos geograficamente dispersos" [Tennant, 2000]. Os principais projetos nesta linha são: *arXive e-Print Archive* [ARXIV, 2002], NCSTRL [NCSTRL, 2002], NTDL [NTDL, 2002] e NASA *Technical Report Server* [NTRS, 2002]. A interoperabilidade entre os diversos arquivos tem sido obtida através do estabelecimento de padrões pelo *Open Archive Initiative*. Tais padrões são: protocolos para extração de metadados, critérios para a seleção dos metadados extraídos e um formato de metadados comum que os arquivos possam usar para responder às requisições. Há também o protocolo Z39.50 da ANSI/ISSO. Este padrão é o mais aceito atualmente nos aplicativos de software específicos para bibliotecas. Dentre os serviços oferecidos por ele estão os seguintes: pesquisa e recuperação da informação, manipulação de mensagens de erro, acesso a informações sobre o conteúdo dos recursos, dentre outros.

- **Aspectos econômicos, legais e sociais.**

As bibliotecas digitais são sistemas de informação inseridos em um ambiente muito maior onde transitam questões legais, sociais e econômicas. Os direitos

de propriedade intelectual, cobrança e privacidade do usuário são, dentre outras, preocupações presentes na definição de modelos econômicos para estes sistemas.

Todavia, os aspectos legais podem tomar conotações diferentes dependendo de qual o objetivo e escopo da biblioteca digital ou das leis de propriedade intelectual do país onde ela está abrigada. De forma a atender tais necessidades, visando à interoperabilidade, a arquitetura destes sistemas deve suportar mecanismos alternativos para a implantação destas políticas, ou seja, deve ser flexível e dinâmica o suficiente para acomodar novas modalidades e tipos de cobrança pela informação.

- **Interfaces com o usuário ou interação homem-computador**

No contexto das bibliotecas digitais, algumas preocupações relativas à usabilidade do sistema são importantes e dizem respeito às seguintes questões: apresentação da informação, visualização e navegação em grandes coleções, mecanismos para facilitar a consulta e ferramentas para análise e manipulação da informação.

O sucesso do sistema da biblioteca digital será determinado pela sua facilidade de uso e de assimilação pelos usuários, e daí a importância estratégica da discussão dessas questões.

- **Acesso internacionalizado à informação**

A troca de informações entre fontes de dados de países e idiomas diferentes é uma necessidade cada vez mais patente no mundo globalizado. Os sistemas devem suprir às necessidades dos usuários superando barreiras culturais e lingüísticas.

A preocupação com o acesso à informação multi-idioma é muito forte nas comunidades de pesquisas em bibliotecas digitais na Europa, muito mais que nos Estados Unidos.

As principais questões relativas a este tema são:

- Troca de dados: Envolve aspectos de codificação dos caracteres, navegação, *display* de fontes. Implica na definição de padrões internacionais para o compartilhamento tanto de dados quanto de metadados.

- Processamento de linguagens: Envolve as tecnologias de processamento de linguagem natural para: processamento de discursos, tradução, recuperação e consulta de informações em múltiplas linguagens, etc.

Recomenda-se o estudo de infraestruturas para bibliotecas digitais flexíveis de modo que, a partir da constante análise das necessidades do usuário, elas possam evoluir.

2.2 Frameworks

A tecnologia, no último século, sofreu avanços inigualáveis a qualquer outro período da história da civilização. A informática, em especial, é cada vez mais onipresente, circundando quase todas as atividades de nossa vida cotidiana.

Diante deste processo, a indústria de software e os desenvolvedores, de maneira geral, vêm se deparando com uma crescente demanda por novos sistemas cada vez mais complexos. Além disso, as pressões econômicas impostas pela estrutura de um mercado globalizado, impõem prazos mais curtos. As empresas devem produzir em *Internet Time* [Cosumano, 1999].

A forma de produzir software não acompanhou o passo da tecnologia. O impacto destas forças nos tradicionais processos de desenvolvimento de software resulta em um produto final com pelo menos um destes problemas: alto nível de falhas *(bugs)*; desacordo com os requisitos do usuário; ou extrapolação prazo estipulado. Com o intuito de aliviar estas conseqüências, os desenvolvedores têm se valido, agora mais do que nunca, do **Reuso**.

As técnicas de reutilização podem ser empregadas em qualquer uma das fases do desenvolvimento de software: análise, projeto ou implementação. A Figura 3, ilustra esta situação.

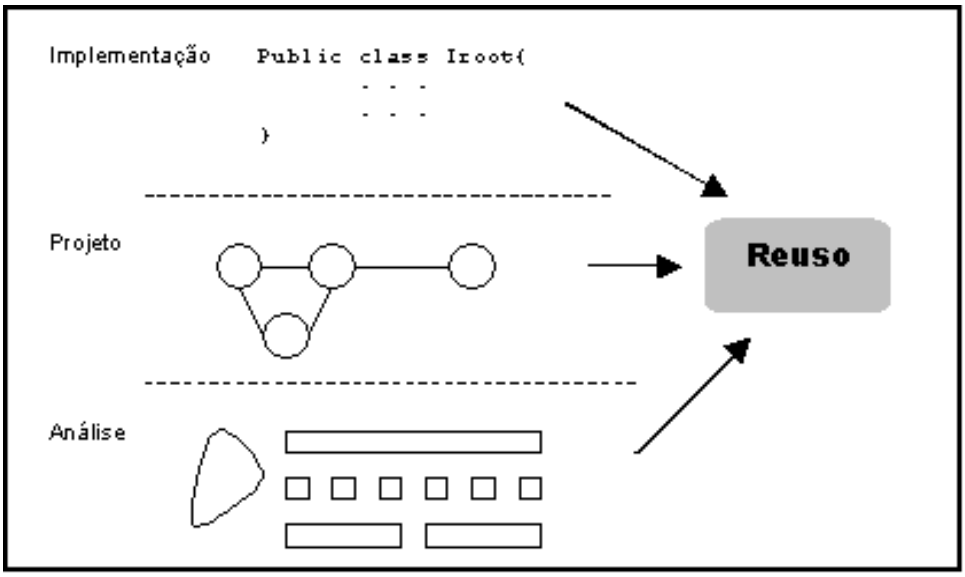


Figura 3 - Reuso no processo de desenvolvimento de software

Diversas técnicas oferecem reuso durante as fases do processo de desenvolvimento. As bibliotecas de classes ou de componentes, por exemplo, viabilizam o reuso na fase de implementação. O código produzido e já testado é menos sujeito a erros, reduz o esforço e o tempo de desenvolvimento. As novas aplicações invocam a biblioteca de classes quando desejam utilizar-se de seus recursos. Estas bibliotecas, em alguns casos são incorporadas, através do *linking,* à aplicação do usuário, com exceção das bibliotecas de *linking* dinâmico (*dll* para o Windows ou *so* para o Unix). A Figura 4, representa graficamente este processo.

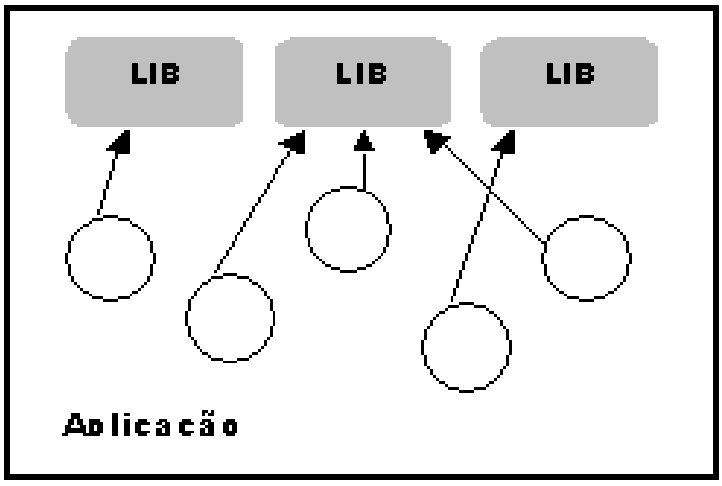


Figura 4 - Reuso com bibliotecas de classe

Os frameworks são essencialmente uma técnica de reuso orientada a objetos [Fayad, 1999]. São aplicações semiprontas em cima das quais novas aplicações, referentes ao mesmo domínio do problema, podem ser criadas. Sob a ótica da reusabilidade, utilizando frameworks, atingimos o reuso em pelo menos três fases do desenvolvimento de software. A Tabela 3, detalha o reuso obtido com o uso dos frameworks nas fases de desenvolvimento.

Reuso de Análise	Todo o entendimento do problema e o seu mapeamento para um conjunto de entidades e termos específicos são feitos na análise dos frameworks. Até mesmo a criação de um vocabulário com conceitos próprios do domínio do sistema. As aplicações criadas a partir do framework reutilizarão a análise para ele empreendida.
Reuso de Projeto	As aplicações geradas com o framework reutilizam o seu projeto tanto no nível arquitetural, que envolve a divisão do sistema em camadas, definição dos padrões de comunicação, definição dos bancos de dados, dentre outras; quanto no baixo nível, que engloba os componentes a serem utilizados, o fluxo de interação entre os módulos, as interfaces que comporão a aplicação e os padrões de projetos utilizados na solução.
Reuso de Implementação	De maneira geral, uma aplicação construída a partir de um framework é, com relação a sua implementação, uma aplicação mista: parte do código foi escrita pelo desenvolvedor e parte do código é o próprio framework. Isto acontece através dos mecanismos de extensão do framework, como, por exemplo, a implementação de uma sub-classe de uma super-classe (já implementada). Além disso, os frameworks podem oferecer recursos em uma biblioteca de componentes a serem utilizados diretamente no código desenvolvido.

Tabela 3 - Reuso nos Frameworks

Cabe ressaltar que nem todo framework implementa classes concretas, já implementadas, ou disponibiliza bibliotecas de componentes prontos para uso. Há frameworks conceituais que não atingem o reuso no nível da implementação.

Destas três dimensões, consideram-se as mais importantes, e que, em longo prazo, compensam os custos e recursos investidos, o reuso de análise e de projeto [Biggerstaff, 1989]. Analisamos, a seguir, alguns motivos para esta afirmativa:

- O processo de análise é custoso, uma vez que, depende de especialistas no domínio, pessoas que conheçam profundamente o negócio e que definam os requisitos do sistema. Normalmente, estes profissionais não são parte da equipe de desenvolvimento. No momento em que se pode reutilizar a análise do problema, a dependência existente destas pessoas é diminuída, trazendo uma maior agilidade ao processo.
- O projeto de um sistema de software de qualidade requer um arquiteto de sistemas bem gabaritado. Projetar sistemas complexos não é uma atividade trivial. O resultado desta atividade influi diretamente sobre o desempenho do sistema, eventuais dificuldades para a sua implementação e manutenção. Padrões de projeto [Gamma *et al*, 1994] são peças fundamentais neste processo, uma vez que através deles é feito reuso de idéias. O reuso de projeto possibilita redução no tempo de desenvolvimento e um aumento das chances de sucesso da aplicação.

2.2.1 Fundamentos e Conceitos

Segundo [Gamma *et al*, 1994] frameworks podem ser definidos como um conjunto de classes cooperativas que compõem um projeto reutilizável para uma classe específica de software. Um framework oferece um guia arquitetural para a partição do projeto em classes abstratas e a definição de suas responsabilidades e colaborações. O desenvolvedor adapta o framework para a sua aplicação particular através da criação de sub-classes e da composição de instâncias das classes dos frameworks.

Outra definição freqüentemente utilizada afirma que "o framework é o esqueleto de uma aplicação que pode ser adaptado por um desenvolvedor de aplicações" [Fayad, 1999]. Estas duas definições não são conflitantes e sim complementares. A

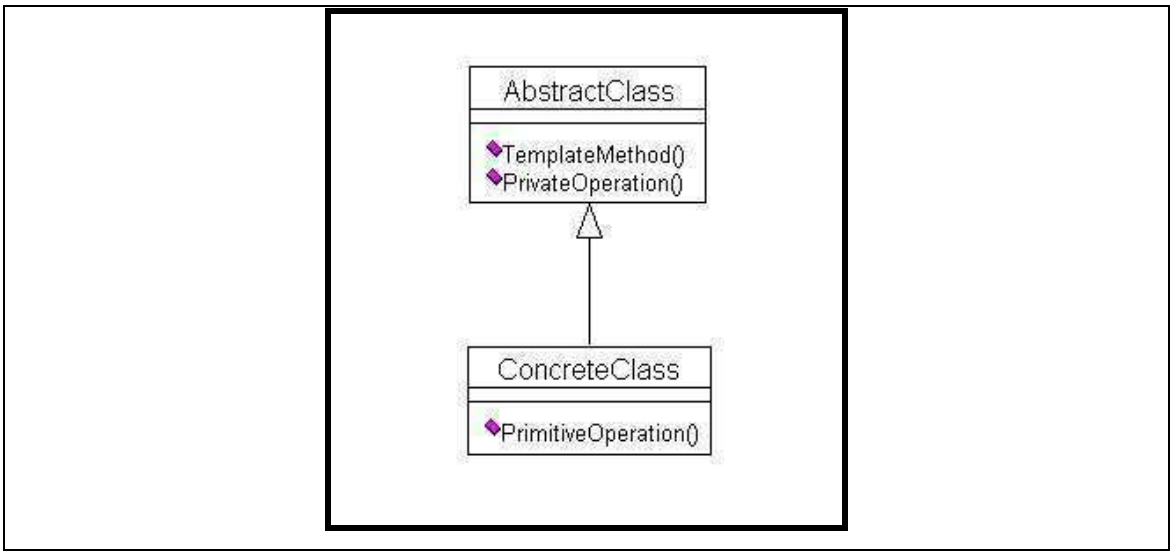
primeira descreve os frameworks do ponto de vista do projeto, enquanto que a segunda, os descreve do ponto de vista mais funcional.

Para que uma nova aplicação seja criada a partir de um framework ele deve ser **estendido**. Entende-se por este termo, no contexto dos frameworks, a capacidade destes sistemas de software de serem modificados e se adaptarem às necessidades da aplicação a ser criada. Com relação à técnica utilizada para esta adaptação, [Roberts, 1997] classificam os frameworks como *whitebox* e *blackbox*.

Nos tipos *whitebox,* a herança e o *binding* dinâmico são as chaves para a criação de novas aplicações. A extensão pode ser feita através das seguintes atividades:

- Implementação das interfaces do framework. Os métodos definidos no projeto são codificados pelo desenvolvedor, de modo particular, para a sua aplicação.
- Criação de sub-classes que herdam de super-classes do framework, ampliando o seu comportamento através da criação de novos métodos.
- Sobrecarga de métodos da super-classe. Determinadas classes ou interfaces dos frameworks deixam "ganchos" em seu conjunto de métodos a fim de que suas sub-classes redefinam o seu comportamento de forma a particularizar seu procedimento. Esta é a idéia do padrão **Template** [Gamma *et al*, 1994], de acordo com o quadro a seguir.

Padrão Template
"Este padrão tem como objetivo definir um algoritmo em uma operação, através da derrefêrencia de alguns passos para as suas sub-classes. Os métodos template permitem que as sub-classes redefinam certos passos de um algoritmo sem mudar a sua estrutura. As classes são denominadas ***Templates*** *e os métodos* ***Hook****".*



A utilização de frameworks *whitebox* requer um conhecimento bastante aprofundado de seu projeto. As aplicações criadas apresentam um forte acoplamento com o framework. Tal fato pode ocasionar problemas nas fases de manutenção e evolução, pois o impacto no código da aplicação criada é grande.

Nos frameworks *blackbox*, a adaptação para a criação de novas aplicações não causa tanto acoplamento destas com o framework quanto nos *whitebox*. Em alguns casos, inclusive, não é disponibilizado ao desenvolvedor o código fonte e a criação da aplicação é feita através de parametrizações e arquivos de definições. A composição de classes ou associação de componentes também pode ser utilizada para a extensão do framework. As classes ou componentes da nova aplicação devem ser criadas, de acordo com as interfaces definidas no projeto, para que se encaixem perfeitamente à estrutura existente.

No geral, a utilização de frameworks *blackbox* é mais fácil do que a utilização dos *whitebox*. O desenvolvedor não precisa entender como ele funciona, apenas o que deve fazer para parametrizá-lo ou quais interfaces adotar na criação de classes ou componentes de sua aplicação. Entretanto, o desenvolvimento de um framework *blackbox* é bem mais complexo. Requer um alto nível de abstração e de conhecimento do domínio ao qual se destina o framework.

Usualmente, não há uma divisão tão rigorosa entre as classificações *blackbox* e *whitebox.* A grande maioria deste tipo de sistema oscila entre uma ou outra classificação de acordo com a funcionalidade a ser implementada. Além do mais, no ciclo de

desenvolvimento proposto por, a classificação *blackbox* é a etapa final a ser atingida pelos frameworks. Um framework *whitebox* evolui para *blackbox* através de várias iterações de seu ciclo de desenvolvimento. Em cada uma delas são feitos novos refinamentos e o projeto evolui. O processo de desenvolvimento de frameworks será abordado mais adiante.

Embora considerados mais difíceis de serem utilizados, os frameworks *whitebox* apresentam um grau mais elevado de adaptabilidade com relação ao *blackboxs*, e em decorrência disso oferece mais poder ao desenvolvedor. Os *whitebox* são mais abertos enquanto que os *blackboxs* atuam em um nicho de aplicações mais restrito.

No geral, os frameworks apresentam as seguintes características [Fayad, 1999]:

- **Modularidade:** Obtida através do encapsulamento, em interfaces bem definidas, de pontos da implementação que são passíveis de mudança. Isto minimiza o impacto gerado quando da manutenção do software e facilita o seu desempenho.
- **Reusabilidade:** A utilização do projeto de framework na criação de novas aplicações e das definições de interfaces para a construção de componentes que se adaptem à sua estrutura ratificam a característica de reusabilidade assegurada para frameworks. O aumento da produtividade dos programadores, da qualidade e da confiabilidade dos programas gerados são ganhos diretamente vinculados ao reuso promovido pelo framework.
- **Adaptabilidade:** O framework permite que o desenvolvedor crie suas aplicações através de adaptações: extensão de interfaces e métodos pré-determinados. Desta forma, está assegurada a criação de infinitas possibilidades de novas aplicações com base no mesmo framewrok.
- **Inversão de controle:** Esta característica fundamental dos frameworks permite que eles definam que método específico da aplicação a ser gerada deve ser invocado em resposta a um evento externo. Neste sentido, os frameworks têm o funcionamento contrário ao de uma aplicação que utiliza uma biblioteca de classes. Esta característica é conhecida como princípio de *Hollywood "Não nos chame, nós que chamaremos você"*. Na Figura 5, o código escrito pelo desenvolvedor invoca métodos ou utiliza recursos das bibliotecas de classe já existentes na aplicação 1. Na aplicação 2, construída a partir de um framework, o código escrito pelo desenvolvedor é invocado pelos elementos do framework.

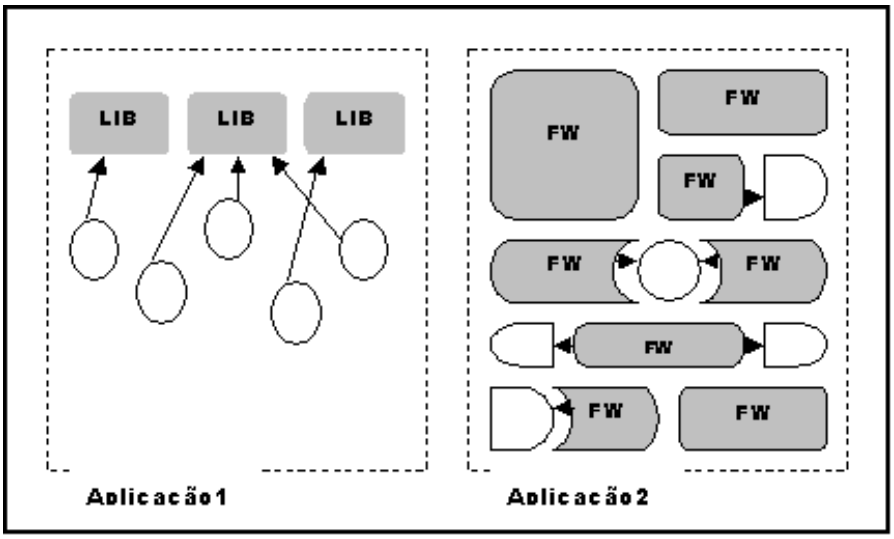


Figura 5 - Inversão de controle

2.2.2 Desafios e Questões

São grandes os desafios no desenvolvimento de frameworks. Por este motivo, muitas iniciativas de construção deste tipo de sistemas não chegam ao seu final. A construção de sistemas complexos demanda um grande esforço de desenvolvimento. Se comparado ao desenvolvimento de frameworks, que devem ser genéricos o suficiente para a derivação de inúmeras aplicações complexas em cima de um mesmo domínio, concluímos que o esforço de desenvolvimento neste caso é ainda maior.

O primeiro grande desafio é definir o escopo e o domínio ao qual o framework irá atender. A correta definição do domínio é fundamental para assegurar a viabilidade de um projeto. Um domínio muito abrangente oferece grandes riscos do projeto não ser concluído ou, devido à reduzida reusabilidade não ser adequado. A orientação a ser seguida na escolha do domínio é concentrar-se em nichos de negócio bem definidos. Algumas dificuldades encontradas na definição do tamanho do domínio estão na Tabela 4.

Grandes domínios	Pequenos domínios
- Dificuldade na aplicabilidade do framework; - Custo de desenvolvimento proibitivo;	- As fronteiras da aplicação gerada ultrapassam as do próprio frameworks exigindo mudanças no

- Duração muito longa do projeto;	projeto do mesmo.

Tabela 4 - Domínios dos frameworks

A fase de análise no processo de desenvolvimento dos frameworks também apresenta algumas dificuldades. Esta fase envolve todas as atividades realizadas na análise de um sistema comum, mas vai mais além. No caso dos frameworks, esta fase tem um objetivo mais amplo. Ela deve ser suficientemente abrangente para que sejam previstas diversas variações de aplicações dentro do domínio do problema, ou seja, o desenvolvimento de frameworks requer análise de domínio.

Segundo [Schaffer, 1994], a análise de domínio tem como objetivo descrever o domínio que será coberto pelo framework. Para capturar os requisitos e identificar os conceitos envolvidos, o desenvolvedor deve lançar mão de aplicações produzidas dentro do domínio. Informações importantes são colhidas através do auxílio de especialistas na área. Uma outra contribuição é obtida através da utilização de padrões, soluções comprovadamente eficientes, existentes para o domínio em estudo. O resultado desta fase é composto pelo modelo de análise do domínio, os conceitos e vocabulário usual, além dos relacionamentos entre os elementos participantes.

O processo de desenvolvimento de frameworks, independentemente da técnica utilizada, é de natureza iterativa [Johnson, 1988]. Um ciclo de iteração completa deve incluir, além dos refinamentos de análise e projeto, a produção de aplicações, o que corresponderia à fase de testes. São necessárias várias iterações até se chegar a um projeto razoavelmente estável. Isto acontece porque o refinamento da análise do domínio só pode ser feito depois de uma aplicação prática do framework. Neste sentido, quanto mais aplicações forem geradas neste processo maior será a consolidação das abstrações e maior será a chance de sucesso – reuso do framework.

As aplicações geradas pelo framework também são extremamente importantes para a determinação e validação dos *hot spots*. Os *hot spots* são as partes do framework abertas à extensão e adaptação [Pree, 1995]. Estes pontos de extensão devem, inicialmente, ser identificados na fase de análise de domínio [Fayad, 1999]. Os componentes, acoplados aos *hot spots* para a criação da nova aplicação, podem ser feitos pelos próprios desenvolvedores de acordo com as interfaces fornecidas pelo projeto do framework.

Frameworks podem ser complicados e de difícil compreensão. Logo, uma documentação de boa qualidade é fundamental para que eles sejam entendidos e utilizados. Johnson [Johnson, 1992] propõe que a documentação apresente pelo menos três fases distintas: descrição do propósito do framework, descrição de como utilizá-lo e descrição de seu projeto.

Tendo em vista que o objetivo norteador da construção de frameworks é a reutilização, a documentação tem um papel de grande importância. Uma documentação clara e suficientemente abrangente capacita o desenvolvedor a utilizá-los e atrai novos usuários, popularizando o software desenvolvido.

Uma boa documentação não se restringe apenas à descrição de componentes, suas classes e interfaces. Isto é importante, mas não é suficiente. A dinâmica do sistema e a sua estrutura são, didaticamente, melhor apresentadas através de exemplos de utilização. Há várias técnicas para a confecção da documentação, dentre elas: o *Cookbook* [Krasner, 1988], *Unified Modeling Language* – UML [Fowler, 1999], Padrões de Projeto [Gamma *et al*, 1994] e Contratos [Meyer, 1992]. Na Tabela 5 são tecidos comentários a respeito dessas técnicas. A maior preocupação, todavia, no aspecto de documentação, é com o elevado custo de sua confecção. Porém, sem a documentação adequada, é quase improvável a aceitação do framework.

Livro de Receitas (Cookbook)
Uma receita descreve como realizar um caso típico de reuso durante o desenvolvimento de uma aplicação utilizando o framework. Ela é apresentada de maneira informal e utilizando vários elementos gráficos e códigos exemplo.
Padrões de Projeto
Padrões de projeto são descrições de problemas recorrentes e suas usuais soluções, para um contexto particular. Padrões podem ser utilizados para a documentação desde a fase de análise até o projeto. Utilizando padrões arquiteturais, podem ser definidos modelos adaptáveis de alto nível para todas as aplicações do domínio em estudo.
Contratos
Os contratos são especificações de obrigações e colaborações. Os contratos de interface definem as interfaces das classes. Os contratos de interação definem o comportamento colaborativo entre os elementos que desempenham atividades em função de um objetivo

comum.
UML
Para a especificação 1.3 da UML, os frameworks são modelados como um *package* estereotipado. Consistindo em sua maioria de padrões, eles são considerados padrões arquiteturais. Os frameworks na UML 1.3 são descritos como colaborações parametrizadas. O aspecto comportamental e colaborativo do framework não é modelado.

Tabela 5 - Técnicas de documentação

Aprender a utilizar um framework não é fácil. Requer um certo esforço de investimento em treinamento, em alguns casos. Em decorrência disto, a curva de aprendizado é um fator a ser pesado antes da adoção de um framework para determinado projeto. Deve ser ponderado se o valor investido será amortizado nas próximas utilizações do framework. Se isto não ocorrer, não vale a pena adotá-lo.

Capítulo 3

Diliframe – Um framework para bibliotecas digitais

Neste capítulo, será apresentado o Diliframe, um framework para a construção de bibliotecas digitais. Será discutida a motivação para este projeto e os benefícios que serão obtidos. Este capítulo traça a linha de evolução do Diliframe desde a sua concepção até a definição de sua arquitetura.

Assim, o capítulo divide-se em duas partes fundamentais, antecedidos por uma seção de apresentação. A primeira parte, discute os aspectos conceituais relativos à concepção do Diliframe, a partir da análise de seus requisitos. A segunda parte, detalha os aspectos arquiteturais relevantes para a construção do Diliframe, finalizando com a sua arquitetura.

3.1Apresentação

As bibliotecas digitais são ferramentas de reconhecida importância para aqueles que dependem ou precisam de informações agrupadas por termos de interesse coletivo. As ferramentas de busca oferecidas na Internet são, muitas vezes, ineficientes e não permitem consultas mais refinadas. Conseqüentemente, a recuperação da informação desejada é comprometida. Além do mais, as ferramentas de busca não asseguram a qualidade e a procedência da informação recuperada.

Bons projetos de bibliotecas digitais mantêm um índice equilibrado de *recall* X *precision,* como salientado na seção 2.2.2, e se responsabilizam pelo conteúdo oferecido. Neste sentido, os usuários se asseguram de que estão diante de uma boa fonte de informação.

Diversas instituições já dispõem, ou planejam dispor, de bibliotecas digitais quer seja para uso interno, quer seja para oferecer este serviço a uma comunidade de usuários mais ampla. No segundo capítulo, pôde-se observar que em todo o mundo surgem novos projetos de bibliotecas digitais. Em sua maioria, são projetos acadêmicos e têm como objetivo contribuir com a área acrescentando novas particularidades e temas de pesquisa ainda não explorados.

O que é proposto nesta dissertação é mais um projeto para a construção de bibliotecas digitais. Todavia, com um propósito mais amplo: o de ser genérico, flexível e adaptável o suficiente a fim de servir como base para a implementação das mais distintas bibliotecas digitais possíveis.

A necessidade da construção de um framework, tecnologia discutida no segundo capítulo, para bibliotecas digitais foi identificada, fundamentalmente, dada à demanda destes sistemas. Utilizando-se frameworks, as aplicações são, teoricamente, construídas mais rapidamente, com um nível de qualidade mais elevado e de forma mais fácil. Além do mais, um sistema de biblioteca digital não é trivial e nem todas as instituições que desejam dispor deste têm de pessoal qualificado para construí-lo. O Diliframe tem como objetivo popularizar a solução e facilitar a disseminação de recursos e serviços através das bibliotecas digitais.

Por tratar-se de um framework, o Diliframe tem, intrinsecamente, a característica de ser evolutivo. Um bom framework pode ser considerado maduro quando passou pelo processo de uso e refinamento diversas vezes, para que, em cada um destes ciclos, novas características e funcionalidades sejam agregadas ao projeto inicial. Dificilmente um projeto de um framework é dado por encerrado. Se isto acontece sinaliza que ele não está mais sendo utilizado, ou que, pelas razões mais diversas, foi descontinuado.

É apresentado nesta dissertação o projeto inicial do Diliframe, uma vez que por ser de característica evolutiva, o projeto de um framework pode mudar bastante. O projeto procura ser bastante genérico e abrangente. O seu desenvolvimento foi baseado em cuidadosa análise de projetos já existentes e de contribuições de especialistas neste domínio. Procurou-se levar em consideração os requisitos mais freqüentes e os pontos positivos de cada projeto, desde que se adequassem ao propósito do Diliframe.

O DiliFrame é composto por um projeto completo de um sistema de biblioteca digital, incluindo diversos artefatos; seu modelo de dados; seu modelo de metadados, e ainda componentes e classes já implementadas – que compõem a *component library* deste framework. Por tratar-se de um framework classificado como *White Box*, o Diliframe necessita de intervenção programática do desenvolvedor para a geração de suas aplicações.

3.2Aspectos Conceituais

Nesta seção são analisados os aspectos conceituais estudados na concepção do projeto do Diliframe. Para tanto, são discutidos, dentro do processo de desenvolvimento adotado, os principais artefatos relativos a esta fase. O processo de desenvolvimento utilizado na construção do Diliframe é caracterizado por duas premissas: ser iterativo e incremental. É um processo de desenvolvimento baseado ao RUP – *Rational Unified Process* [RUP, 2002], mas aplicado de forma semelhante a Larman em seu livro "*Applying UML and Patterns*" [Larman, 1998] e adaptado para a realidade do desenvolvimento deste framework. A linguagem de modelagem utilizada para a produção dos artefatos foi a UML – "*Unified Modelling Language*" [Fowler, 1999].

3.2.1 Análise dos Requisitos

Entende-se por requisitos as reais necessidades do sistema que se está tentando produzir [Larman, 1998]. É necessário profundo estudo e documentação dos requisitos do sistema, a fim de definir o escopo do projeto e permitir a validação do mesmo ao seu término. O grau de sucesso do planejamento, e em conseqüência disto, a qualidade do sistema desenvolvido, pode ser medido contrastando-se os requisitos iniciais com o resultado final alcançado.

Com relação aos requisitos de um sistema, podemos distinguí-los em duas classes: os requisitos funcionais e os requisitos não-funcionais. Ao primeiro grupo, cabe a definição das reais necessidades do sistema suas funcionalidades, objetivos e seus atributos. No segundo grupo objetiva-se investigar as restrições tecnológicas e sociais que atuam no sistema: ambiente de produção, necessidades técnicas, determinações orçamentárias,

público alvo, cronograma, dentre outros. Para a definição dos requisitos funcionais deste projeto, o foco foi detido no framework, e não nas funcionalidades da aplicação criada.

A seguir, estão relacionados os onze requisitos funcionais do Diliframe.

- **RF1 – Suportar a criação de Bibliotecas Digitais.**

O objetivo final deste framework é dar suporte ao desenvolvimento de bibliotecas digitais. Deseja-se oferecer ao desenvolvedor um projeto reutilizável e componentes já implementados que auxiliem nesta tarefa. O projeto do Diliframe é uma arquitetura genérica deste domínio de problema. Depois de especializado e estendido, o desenvolvedor obterá a sua aplicação particular.

- **RF2 – Prever um comportamento *default* para as aplicações derivadas.**

O Diliframe deve oferecer uma implementação padrão para a criação de uma biblioteca digital simples. Isto permitiria que uma aplicação pudesse ser criada, rapidamente, de maneira simples, sem a necessidade massiva de codificação. Neste sistema padrão, os módulos do framework que podem ser particularizados, ou seja, os *hot spots*, já estariam definidos e implementados.

Tomando a interface com o usuário como exemplo, o Diliframe deverá disponibilizar uma interface padrão já implementada. Obviamente, este módulo poderá ser particularizado para as necessidades das aplicações.

- **RF3 – Permitir o armazenamento e recuperação de dados em formatos variados.**

Os dados que podem ser armazenados e recuperados, nas bibliotecas digitais geradas a partir do Diliframe, podem ser multimídia.

- **RF4 – Armazenar metadados sobre as coleções e os recursos de acordo com um padrão determinado.**

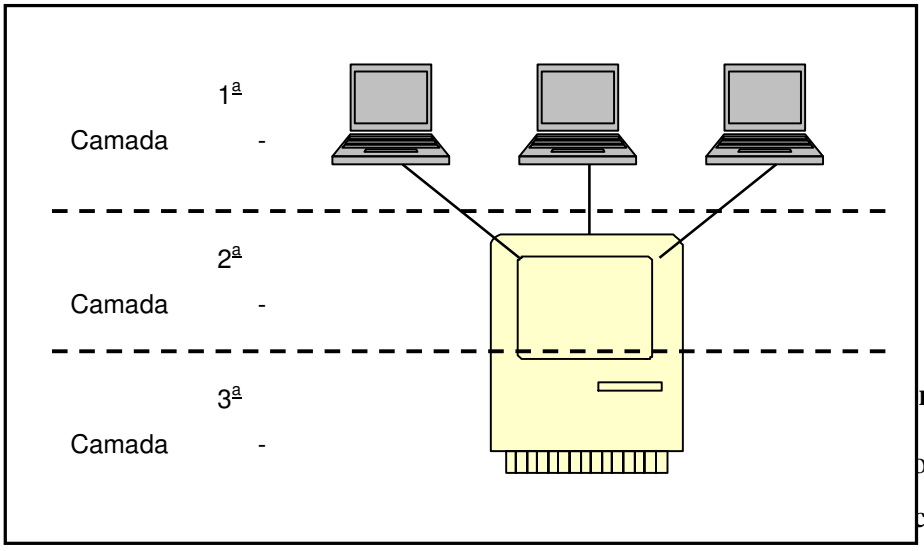
É necessário que o framework defina um padrão de metadados a ser adotado pelas suas aplicações derivadas. Metadados são importantes no escopo de bibliotecas digitais, principalmente, nas que viabilizam o acesso a dados multimídia. Eles são utilizados para a descrição dos recursos digitais, independentemente, de seu formato. Desta

forma, é possível a criação de um mecanismo de busca único e transparente ao usuário quaisquer que sejam os tipos de dados que serão recuperados.

- **RF5 – Possibilitar o acesso a repositórios distribuídos.**

Os dados, ou recursos, das bibliotecas digitais são agrupados em coleções e, logicamente, armazenados em repositórios. Com relação à dispersão dos dados nas camadas do sistema, identificamos as seguintes situações:

- Um repositório presente na mesma máquina do servidor de aplicação. Nesta situação, os elementos da segunda e da terceira camada: a aplicação e as suas regras de negócio e os dados, respectivamente, coexistem na mesma máquina.



r

- ntém o
cessado,

mas com uma localização fixa. Desta forma, a segunda e a terceira camada estão em máquinas separadas. Por tratar-se apenas de um repositório, a localização dos dados é bem determinada.

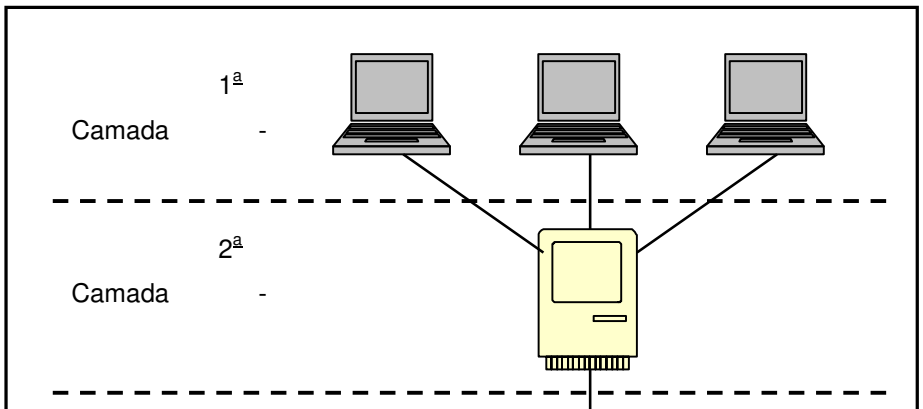


Figura 7 – Repositório de dados e aplicação em servidores distintos

- Diversos repositórios distribuídos na rede e acessados pela aplicação localizada em um servidor de aplicações. Nesta situação, a camada de dados está fisicamente dispersa em vários servidores. Foge do escopo deste framework manipular uma arquitetura de banco de dados distribuída. Esta última situação, apenas ilustra o acesso de uma aplicação a diversos bancos de dados distintos, localizados em máquinas também distintas que armazenam coleções que juntas fazem parte do acervo de uma biblioteca digital.

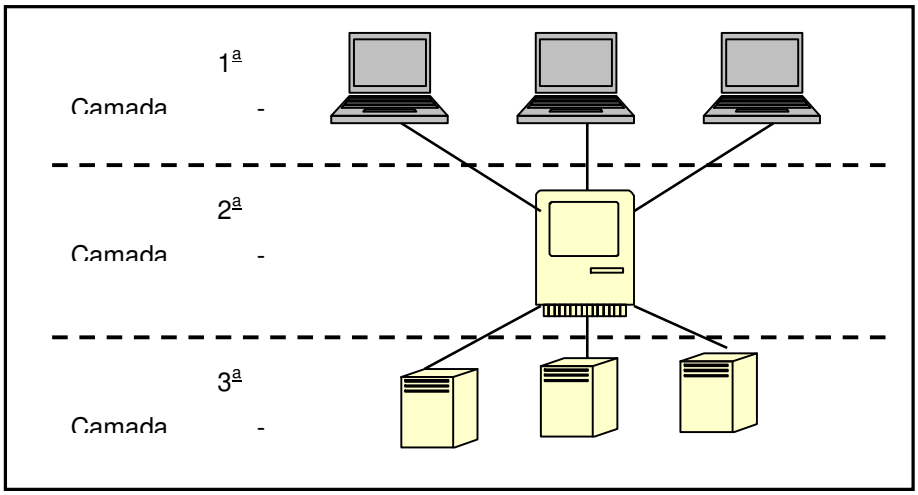


Figura 8 – Vários repositórios de dados acessados por uma aplicação

É requisito do Diliframe permitir a criação de bibliotecas digitais que adotem qualquer uma das três escolhas descritas anteriormente.

- **RF6 – Suportar a adoção de um sistema de segurança.**

Esta questão depende do escopo, da missão e das funcionalidades implementadas no sistema de biblioteca digital. O Diliframe deverá oferecer suporte à implementação de um esquema de segurança, oferecendo uma implementação que assegure, pelo menos, um nível de segurança básico. Entretanto, a possibilidade de composição de módulos que assegurem um nível de segurança mais refinado deve ser permitida.

- **RF7 – Suportar a adoção de um esquema de cobrança.**

Assim como a questão da segurança, o esquema de cobrança das bibliotecas digitais é uma decisão de projeto bastante particular de cada sistema. Novamente, tal decisão depende do escopo, da missão e das funcionalidades implementadas no sistema. O Diliframe deve permitir a composição de módulos implementados pelo desenvolvedor, que assegurem o processo de cobrança pelos serviços prestados pela biblioteca digital.

- **RF8 – Permitir a agregação de novos serviços.**

As bibliotecas digitais podem oferecer uma vasta gama de serviços que vão além do processo de busca e recuperação das informações. As aplicações geradas com o Diliframe devem permitir a composição de módulos implementados pelo desenvolvedor capazes de oferecer serviços que agreguem valor ao sistema.

- **RF9 – Dispor de ferramentas para a administração do catálogo de usuários.**

O Diliframe deve oferecer às suas aplicações derivadas um módulo que viabilize a manipulação dos dados dos usuários: alteração, adição, exclusão e consulta. Esta ferramenta é importante para facilitar a administração do catálogo de usuários.

Há sistemas tão simples que não demandam a necessidade de registro de usuários. Portanto, o Diliframe deve contemplar a criação deste tipo de sistema também. Desta forma, o módulo de administração de usuários não é compulsório na aplicação gerada, mas, tão somente, uma escolha do desenvolvedor.

- **RF10 – Dispor de ferramentas para a administração dos recursos e coleções.**

Várias coleções podem compor o acervo de uma biblioteca digital. As coleções agrupam recursos segundo uma dada característica. A administração das coleções inclui operações como: cadastro de novas coleções, exclusão de coleções, alteração dos dados da coleção, etc. O Diliframe deve oferecer às suas aplicações derivadas ferramentas para a administração das coleções.

Assim como o requisito funcional anterior o módulo de administração de usuários, este também não é um módulo compulsório. A sua utilização será determinada pela necessidade da aplicação.

- **RF11 – Fornecer uma sintaxe de consulta comum para as coleções armazenadas em repositórios heterogêneos.**

O Diliframe permite a criação de aplicações cujas coleções estão distribuídas em várias máquinas. Os dados podem ser armazenados em bancos de dados de diferentes fornecedores: Oracle [Oracle, 2002], IBM [IBM, 2002], Informix [Informix, 2002], dentre outros.

Convencionou-se, para o Diliframe, que suas coleções devem ser construídas apenas de dados armazenados em bancos de dados relacionais. Entretanto, não está instituído um fornecedor único. Neste cenário, uma biblioteca digital, gerada com o Diliframe, poderia ser capaz de permitir consultas em várias coleções, localizadas em servidores distintos, e armazenados em bancos de dados de diferentes fornecedores.

O RF11 orienta que, para que esta situação seja transparente para o usuário, defina-se uma forma de consulta única, em cima dos atributos de metadados, independente das coleções que irão ser atingidas.

A Tabela 6 sintetiza os requisitos funcionais do Diliframe:

Requisito	Função
RF1	Suportar a criação de bibliotecas digitais.
RF2	Prever um comportamento *default* para as aplicações derivadas.
RF3	Permitir o armazenamento e a recuperação de recursos em formatos variados.
RF4	Armazenar metadados sobre as coleções e os recursos de acordo com um padrão determinado.
RF5	Possibilitar o acesso a repositórios distribuídos.
RF6	Suportar a adoção de um sistema de segurança.
RF7	Suportar a adoção de um esquema de cobrança.
RF8	Permitir a agregação de novos serviços.
RF9	Dispor de ferramentas para a administração de catálogos de usuários.
RF10	Dispor de ferramentas para a administração dos dados e coleções.
RF11	Fornecer uma sintaxe de consulta comum para as coleções armazenadas em repositórios heterogêneos

Tabela 6 – Requisitos Funcionais

A seguir, serão listados os requisitos não-funcionais do Diliframe.

- **RNF1 – Possibilitar, de forma simplificada, o desenvolvimento de sistemas de bibliotecas digitais.**

Este requisito não funcional qualifica o RF1, ou seja, ambos referem-se à capacidade de geração de bibliotecas digitais, que é a principal funcionalidade deste framework. O RNF1 determina que não basta dar suporte à criação de bibliotecas digitais, é necessário que isto seja feito de modo a facilitar o trabalho do desenvolvedor.

Este é o principal requisito não-funcional do Diliframe. Ele existe para garantir que o framework seja realmente útil, para que ele seja reutilizado. Para tanto, o maior atrativo é a simplificação do desenvolvimento.

- **RNF2 – Possibilitar a fácil adição de novos elementos ao sistema.**

A extensão das funcionalidades previstas no Diliframe para a criação de bibliotecas digitais deve ser facilitada. Para a criação de aplicações mais elaboradas e que sejam adequadas às necessidades particulares de cada desenvolvedor, o acoplamento de novos elementos ao sistema é necessário. Tais elementos podem ser: novos serviços, especialização de serviços já existentes, novas coleções, dentre outros.

A necessidade de permitir a agregação de outros módulos foi prevista nos requisitos funcionais RF7, RF8 e RF9. Este requisito não-funcional visa a qualificar a forma como isso deve ser feito, ou seja, de forma facilitada. Isto é importante, por que se não houver um incentivo para a agregação de valor na construção de bibliotecas digitais mais complexas, utilizando o Diliframe, os desenvolvedores podem optar por não utilizá-los.

- **RNF3 – Deve ser acessível a desenvolvedores com experiência na criação de aplicações distribuídas.**

O público-alvo a quem se destina o Diliframe são os desenvolvedores. No entanto, para que sejam quantificados os conceitos de "facilidade" e "complexidade" no desenvolvimento das aplicações, é necessário estabelecer o nível de experiência de quem vai utilizá-lo.

Desta forma, pode-se dizer que "o Diliframe facilita o desenvolvimento de sistemas de bibliotecas digitais" para desenvolvedores que já têm uma certa experiência na criação de aplicações deste tipo.

- **RNF4 – Oferecer uma interface de acesso bem definida, a fim de que os desenvolvedores tenham flexibilidade para a criação de suas próprias interfaces.**

Através do estabelecimento das diretrizes sobre como o usuário interagirá com o sistema, diversas apresentações ou interfaces podem ser utilizadas para viabilizar esta interação. O Diliframe deve prover flexibilidade aos desenvolvedores para a criação de suas próprias interfaces. Assim, utilizando as mesmas diretrizes de acesso, aplicações podem ser geradas tanto com interface Web, quanto com interface de janelas, por exemplo.

- **RNF5 – Deve oferecer uma rica documentação aos módulos desenvolvidos.**

Uma das mais importantes premissas para o sucesso de um framework está relacionada à facilidade de utilização. Dentro deste espoco encontra-se a necessidade de uma boa documentação do sistema. Diversas formas para a documentação deste tipo de sistema vêm sendo utilizadas. Algumas apresentam mais êxito que outras, dentre elas, estão as que se propõem a utilização de exemplos.

A Tabela 7 sintetiza os requisitos não funcionais deste framework para construção de bibliotecas digitais:

Requisito	Função
RNF1	Possibilitar, de forma simplificada, o desenvolvimento de sistemas de bibliotecas digitais.
RNF2	Possibilitar a fácil adição de novos elementos ao sistema.
RNF3	Deve ser acessível a desenvolvedores com experiência na criação de aplicações distribuídas.
RNF4	Oferecer uma interface de acesso bem definida, a fim de que os desenvolvedores tenham flexibilidade para a criação de suas interfaces.
RNF5	Deve oferecer uma rica documentação aos módulos desenvolvidos.

Tabela 7 – Requisitos não-funcionais

3.2.2 Análise de Casos de Uso

A seção anterior avaliou os requisitos levantados para o projeto do Diliframe. Tal estudo resultou numa análise ampla das necessidades do framework. Nesta seção, a granularidade do estudo é menor. Procurou-se listar as necessidades funcionais de uma aplicação gerada com o Diliframe. Para tanto, foi considerada uma biblioteca digital que contemplasse grande parte dos requisitos listados na seção 3.2.1.

Para explorar as funcionalidades desta aplicação, na perspectiva do usuário final, foi utilizada a técnica dos Casos de Uso. Os casos de uso, popularizados por Jacobson [Jacobson, 1992], são descrições de uma interação completa entre um usuário e o sistema em estudo. A UML [Rumbaugh, 1999] reconhece e adota esta técnica na definição e estudo do comportamento dos sistemas:

"A construção de casos de uso é usada para definir o comportamento de um sistema ou entidade semântica, sem expor a estrutura interna desta entidade. Cada caso de uso especifica uma seqüência de ações, incluindo suas variações, que a entidade pode realizar interagindo com os atores".

As ações ou funcionalidades, ilustradas no caso de uso, são desempenhadas por atores. Os atores são entidades externas ao sistema que o estimula através de eventos de entrada ou na recepção de algum processamento, i.e. um evento de saída. Os diagramas de casos de uso são a representação gráfica de todos os casos de uso para determinado sistema. Embora tenham representação visual de um ser humano, nem sempre o papel de um ator é atribuído a usuários ou a operadores humanos. Dependendo do contexto e das fronteiras do caso de uso, os atores podem ser sub-sistemas, módulos, componentes externos, dentre outros.

Para a descrição textual dos casos de uso, não há um consenso, padrão ou modelo estabelecido. Desde a primeira versão da especificação da UML até a sua versão mais recente [OMG-UML, 2002], a orientação para a estruturação de casos de uso é, em sua essência, a mesma:

"Um caso de uso pode ser descrito como texto simples; usando operações métodos e atributos; como um grafo de atividades; como uma máquina de estados ou qualquer outra técnica de descrição de comportamento, como a definição de pré-condições e pós-condições".

A ausência de uma definição formal de como estruturar textualmente os casos de uso abriu espaço para o surgimento de várias propostas com este objetivo [Larman, 1998], [Schneider, 1998], [Cockburn, 1997], [Rumbaugh, 1994].

Neste trabalho, optou-se por adaptar a proposta de [Cockburn, 1997], gerando um novo modelo para a descrição textual de casos de uso. O objetivo deste modelo, em benefício àqueles que escrevem os casos de uso, é tornar mais fácil a sua descrição e ainda viabilizar o registro de idéias e sugestões encontradas durante esta fase. Dentro dessas premissas, buscou-se a eliminação da complexidade, sem abrir mão da completude. O modelo adotado para a descrição dos casos de uso do Diliframe é apresentado no Anexo B.

Análise de Casos de Uso de uma Biblioteca Digital[2]

Nesta subseção, são apresentados os principais casos de uso listados para uma biblioteca digital. Esta biblioteca é uma instância do Diliframe, ou seja, uma aplicação gerada com base neste framework. Buscou-se contemplar desde os casos de uso mais básicos até os mais avançados. Todavia, é impossível exaurir todas as possibilidades de interação, já que o projeto deve permitir a adaptação de novos módulos que originam novos casos de uso.

Na Tabela 8, os atores participantes dos casos de uso são listados juntamente com suas descrições.

Ator	Descrição
Usuário	É o usuário final da biblioteca digital. É ele quem usufrui os serviços por ela oferecidos.
Administrador de usuários	Responsabiliza-se pelo cadastro e manutenção dos usuários da biblioteca digital e de seus dados.
Administrador de coleções	Responsabiliza-se pelo cadastro e manutenção das coleções.
Provedor de dados	Responsabiliza-se por disponibilizar os recursos a serem recuperados pela biblioteca digital
Provedor de metadados	Responsabiliza-se por disponibilizar os metadados que descrevem os recursos da biblioteca digital

Tabela 8 – Atores que interagem com a biblioteca digital

A Tabela 9 relaciona os principais casos de uso considerados para uma biblioteca digital gerada com o Diliframe. Em seguida, é apresentada a descrição completa de alguns deles.

Casos de Uso		
Listar_Coleções	Recuperar_Item	Recuperar_Info_Coleções
Realizar_Login	Pesquisar_Item(Simples)	Pesquisar_Item (Avançada)
Excluir_Usuário	Inserir_Usuário	Alterar_Dados_Usuário

[2] Biblioteca digital gerada a partir do Diliframe.

Listar_usuários	Inserir_Coleção	Alterar_Dados_Coleção
Excluir_Coleção	Inserir_Repositório	Excluir_Repositório
Alterar_Dados_Repostiório	Listar_Repositórios	Listar_Coleções_Repositórios
Criar_Metadado	Incluir_Metadados	Alterar_Metadado
Excluir_Metadado		

Tabela 9 – Casos de Uso considerados

1. Descrição de Caso de Uso - Efetuar Login na Biblioteca Digital

2. Finalidade

Entrar na biblioteca digital para ter acesso a seus serviços.

3. Atores

Usuário

4. Pré-condições

Usuário cadastrado na biblioteca digital.

5. Fluxo Principal

P1. O caso de uso se inicia quando o Usuário entra no sistema;

P2. O sistema apresenta uma tela de login;

P3. O Usuário informa seu nome e senha;

P4. O sistema verifica se o nome do Usuário está cadastrado no sistema; E1

P5. O sistema verifica se a senha informada confere com a cadastrada; E2

P6. O sistema abre uma sessão para o Usuário permitindo o seu acesso à biblioteca digital.

6. Fluxos Alternativos

Nenhum

7. Exceções

E1. O nome do Usuário não consta no cadastro do sistema

1. Segue caso de uso "Cadastrar usuário"

E2. A senha informada pelo Usuário não confere com a cadastrada

1. O sistema informa ao Usuário que a senha informada é incorreta.
2. Segue P2.

8. Pós-condições

Aberta uma sessão para o usuário, identificando que ele está conectado ao sistema.

9. Testes sugeridos

Nenhum

1. Descrição de Caso de Uso – Cadastrar usuário

2. Finalidade

Registrar dados no catálogo de usuários da biblioteca digital para se obter acesso a seus serviços.

3. Atores

Usuário

4. Pré-condições

Usuário não cadastrado no sistema.

5. Fluxo Principal

P1. O caso de uso se inicia quando o Usuário tem acesso ao formulário de cadastramento;

P2. O Usuário informa os dados requeridos no formulário e o submete ao sistema; E1, E2

P3. O sistema informa que o cadastro foi realizado com sucesso;

6. Fluxos Alternativos

Nenhum

7. Exceções

E1. Campos obrigatórios não preenchidos ou inconsistentes

1. O sistema devolve a tela de cadastro do Usuário indicando que campo deve ser preenchido novamente;

2. Segue P2.

E2. Login informado já existe

1. O sistema informa que já há Usuário cadastrado com o login informado;

2. O sistema apresenta a tela de cadastro com os dados do Usuário;

3. Segue P2.

8. Pós-condições

Usuário cadastrado no sistema.

9. Testes sugeridos

Submeter dados de um Usuário já existente.

1. Descrição de Caso de Uso – Listar coleções

2. Finalidade

Obter a lista de coleções disponíveis para pesquisa na biblioteca digital.

3. Atores

Usuário (dispara o evento)

4. Pré-condições

Usuário conectado ao sistema.

Existência de pelo menos uma coleção cadastrada na biblioteca digital.

5. Fluxo Principal

P1. O caso de uso se inicia quando o Usuário dispara a opção de consulta;

P2. O sistema recupera a lista de coleções cadastradas; E1

P3. O sistema verifica na lista de coleções aquelas que o Usuário pode ter acesso; E1

P4. O sistema apresenta a lista de coleções ao Usuário

6. Fluxos Alternativos

Nenhum

7. Exceções

E1. Não há coleções disponíveis

1. O sistema informa que não há coleções disponíveis para consulta;
2. O caso de uso é finalizado.

8. Pós-condições

Nenhuma

9. Testes sugeridos

Executar esta funcionalidade para usuários sem acesso a coleções.

1. Descrição de Caso de Uso – Cadastrar coleções

2. Finalidade

Registrar uma nova coleção na biblioteca digital.

3. Atores

Administrador de coleções

4. Pré-condições

Usuário com perfil de Administrador de coleções cadastrado e conectado ao sistema.

5. Fluxo Principal

P1. O caso de uso se inicia quando o Administrador de coleções dispara a opção de cadastramento de coleções;

P2. O sistema verifica se o Usuário tem perfil de Administrador de coleções e apresenta formulário de cadastramento; E1

P3. O Administrador de coleções submete informações sobre a coleção e seus dados;

P4. O sistema verifica a consistência das informações submetidas; E2

P5. O sistema apresenta mensagem de cadastramento realizado com sucesso;

6. Fluxos Alternativos

Nenhum

7. Exceções

E1. O Usuário não é Administrador de coleções

1. O sistema informa ao Usuário que ele não tem privilégios para realizar o cadastro de coleções;

2. O caso de uso é finalizado.

E2. Informações inconsistentes sobre a coleção

1. O sistema informa ao Administrador de coleções que determinado dado sobre a coleção é inconsistente;

2. O sistema apresenta novo formulário de cadastro;

3. Segue P3.

8. Pós-condições

Uma coleção é cadastrada na biblioteca digital.

9. Testes sugeridos

Cadastrar uma coleção já existente.

1. Descrição de Caso de Uso – Selecionar coleções

2. Finalidade

Escolher as coleções em cima das quais será realizada a consulta.

3. Atores

Usuário

4. Pré-condições

Execução do caso de uso “Listar coleções”.

5. Fluxo Principal

P1. O caso de uso se inicia quando o Usuário tenta selecionar uma coleção aonde será realizada a consulta;

P2. O sistema realiza a seleção da coleção escolhida;

P3. Segue para o caso de uso “Pesquisar item (Simples)” PA1, PA2

6. Fluxos Alternativos

PA1.

1. O Usuário tenta selecionar outra coleção, desde que exista, para que nela também seja realizada a consulta;

2. Segue P2.

PA2.

1. Segue para o caso de uso “Pesquisar item (avançado)”.

7. Exceções

Nenhuma

8. Pós-condições

Uma ou mais coleções selecionadas para que nelas sejam realizadas consultas.

9. Testes sugeridos

Nenhum

1. Descrição de Caso de Uso – Pesquisar item (Simples)

2. Finalidade

Realizar pesquisa na biblioteca digital. A consulta pode ser feita em cima de uma ou mais coleções.

3. Atores

Usuário

4. Pré-condições

Execução do caso de uso "Selecionar coleções".

5. Fluxo Principal

P1. O caso de uso se inicia quando o Usuário dispara a opção de consulta simples;

P2. O sistema apresenta um formulário no qual a consulta simples deve ser realizada;

P3. O Usuário submete o termo a ser consultado;

P4. O sistema prepara e encaminha a consulta para cada uma das coleções selecionadas; E1

P5. O sistema consolida os resultados e devolve ao Usuário uma breve descrição de cada recurso encontrado.

6. Fluxos Alternativos

Nenhum

7. Exceções

E1. Uma coleção para a qual está sendo enviada a consulta está inacessível

1. O sistema verifica se há pelo menos uma coleção das selecionadas disponível e encaminha a consulta; E2

2. Segue P5.

E2. Não há coleções acessíveis para a consulta em curso

1. O sistema informa ao Usuário que as coleções por ele selecionadas não estão acessíveis;

2. O caso de uso é finalizado.

8. Pós-condições

Conjunto resultado da pesquisa.

9. Testes sugeridos

Nenhum.

1. Descrição de Caso de Uso – Pesquisar item (Avançada)

2. Finalidade

Realizar pesquisa na biblioteca digital de forma um pouco mais elaborada que a descrita no caso de uso "Pesquisar item (simples)". Esta pesquisa é feita em cima de campos, ou combinação deles, através do uso de operadores lógicos.

3. Atores

Usuário, Administrador de coleções.

4. Pré-condições

Execução do caso de uso "Selecionar coleções".

5. Fluxo Principal

P1. O caso de uso inicia quando o Usuário dispara a opção de consulta avançada;

P2. O sistema consulta o Administrador sobre os campos em cima dos quais podem ser realizadas as consultas dadas as coleções selecionadas;

P3. O Administrador de coleções informa ao sistema os campos de consulta para as coleções;

P4. O sistema apresenta o formulário de consulta avançada;

P5. O Usuário relaciona os campos convenientes e submete a consulta;

P6. O sistema prepara e encaminha a consulta para cada uma das coleções selecionadas; E1, E2

P7. O sistema consolida os resultados e devolve ao Usuário uma breve descrição de cada recurso encontrado, se for o caso.

6. Fluxos Alternativos

Nenhum

7. Exceções

E1. Erro na utilização do formulário de consulta

1. O sistema informa ao Usuário que sua consulta foi montada incorretamente;
2. Segue P4.

E2. Uma coleção para a qual está sendo enviada a consulta está inacessível

1. O sistema verifica se há pelo menos uma coleção das selecionadas disponível e encaminha a consulta; E2
2. Segue P7.

E3. Não há coleções acessíveis para a consulta em curso

1. O sistema informa ao Usuário que as coleções por ele selecionadas não estão mais acessíveis;

2. O caso de uso é finalizado.

8. Pós-condições

Conjunto resultado da pesquisa se houver algum.

9. Testes sugeridos

Cortar o acesso a coleções durante uma consulta

1. Descrição de Caso de Uso – Recuperar item

2. Finalidade

Obter acesso ao item recuperado após consulta na biblioteca digital.

3. Atores

Usuário, Administrador de coleções.

4. Pré-condições

Execução do caso de uso “Pesquisar item (Simples)” ou “Pesquisar item (Avançada)”

5. Fluxo Principal

P1. O caso de uso inicia quando o Usuário clica no local indicado para a recuperação de determinado item resultante de uma pesquisa;

P2. O sistema obtém do Administrador de coleções o caminho para a recuperação do item desejado; E1

P3. O sistema busca o item desejado e o retorna ao Usuário;

6. Fluxos Alternativos

Nenhum

7. Exceções

E1. Não constam informações sobre o acesso ao repositório de dados

1. O sistema informa ao Usuário que houve falha na recuperação do item selecionado;
2. O caso de uso é finalizado.

8. Pós-condições

O Usuário tem acesso ao item consultado, salvo a ocorrência de alguma exceção.

9. Testes sugeridos

Nenhum

1. Descrição de Caso de Uso – Recuperar informações sobre coleções

2. Finalidade

Obter informações sobre as coleções disponibilizadas pela biblioteca digital para consulta, a fim de facilitar a seleção das coleções onde serão realizadas as pesquisas.

3. Atores

Usuário, Administrador de coleções.

4. Pré-condições

Execução do caso de uso "Listar coleções"

5. Fluxo Principal

P1. O caso de uso inicia quando o Usuário clica no local indicado para a recuperação de informações sobre as coleções;

P2. O sistema consulta o Administrador de coleções para a recuperação das informações sobre a coleção;

P3. O Administrador de coleções retorna as informações requeridas; E1

P4. O sistema apresenta ao Usuário as informações sobre a coleção requerida;

6. Fluxos Alternativos

Nenhum

7. Exceções

E1. Não constam informações sobre a coleção

1. O sistema informa ao Usuário que foi impossível a recuperação de dados sobre a coleção;

2. O caso de uso é finalizado.

8. Pós-condições

Nenhuma.

9. Testes sugeridos

Recuperar informações de uma coleção inexistente.

1. Descrição de Caso de Uso – Inserir dados

2. Finalidade

Alimentar o repositório de dados associado a uma determinada coleção.

3. Atores

Provedor de dados, Administrador de coleções.

4. Pré-condições

Haver pelo menos uma coleção cadastrada na biblioteca digital sem repositório a ela associado.

5. Fluxo Principal

P1. O caso de uso inicia quando o Provedor de dados aciona a opção de carregar dados na coleção;

P2. O Administrador de coleções lista as coleções cadastradas;

P3. O Provedor de dados seleciona a coleção desejada; E1

P4. O Administrador de coleções apresenta um formulário para a inserção de dados sobre o repositório;

P5. O Provedor de dados submete os dados sobre o repositório;

P6. O Administrador de coleções verifica a consistência dos dados e informa que o repositório foi cadastrado com sucesso; E2

6. Fluxos Alternativos

Nenhum

7. Exceções

E1. A coleção selecionada já está associada a algum repositório

1. O Administrador de coleções informa ao Provedor de dados que a coleção selecionada já tem um repositório a ela associada;

2. O caso de uso é finalizado.

E2. Informações inconsistentes sobre o repositório

1. O Administrador de coleções informa ao Provedor de dados que a coleção foi impossível realizar a operação de inserção de dados;

2. O caso de uso é finalizado.

8. Pós-condições

Alteração do status da coleção alimentada para “disponível para a consulta”.

9. Testes sugeridos

Cadastrar um repositório inexistente.

Cadastrar um repositório fora do ar.

Cadastrar um repositório que esteja em uma máquina distinta da do servidor de aplicação.

3.3Aspectos Arquiteturais

Nesta seção, são abordados os aspectos arquiteturais do projeto do Diliframe. Neste sentido, são discutidos e apresentados os aspectos tecnológicos envolvidos na definição desta arquitetura e o impacto causado por tais escolhas.

A subseção 3.3.1 apresenta o modelo de metadados adotado nas bibliotecas geradas a partir do Diliframe. São apresentadas características da implementação, as adaptações e as restrições que serão adotadas na aplicação do Dublin Core ao Diliframe.

A subseção 3.3.2 apresenta o projeto arquitetural do Diliframe, tendo em vista sua arquitetura de alto nível e o modelo de metadados adotado. Nesta oportunidade, serão explicitadas a estratificação do projeto em camadas e as funcionalidades encerradas em cada uma delas. Serão listados os componentes que constituem o projeto e suas responsabilidades. Em cima disto, será definida a espinha dorsal do Diliframe, ou seja, a estrutura fundamental oferecida pelo framework para a construção de bibliotecas digitais. E, finalmente, será determinado o que pode ser estendido pelo desenvolvedor e em que pontos ele pode intervir adicionando componentes, a fim de incrementar novas funcionalidades.

3.3.1 Modelo de Metadados

Um bom padrão de metadados deve primar pela completude e pela simplicidade [Baptista, 2000]. A simplicidade visa a maximizar a usabilidade. A completude procura maximizar a representação dos recursos através dos atributos de metadados. Neste sentido, o padrão de metadados buscado para o Diliframe deveria ser simples o suficiente a fim de

facilitar o processo de geração de metadados e cadastramento das coleções. Por outro lado, o conjunto gerado pelos atributos de metadados, ao qual denominamos registro de metadados, deve ser rico o suficiente para tornar as pesquisas factíveis e eficientes.

O padrão de metadados escolhido para ser aplicado ao Diliframe foi o Dublin Core [DC, 2002] – DC. Além de este padrão cumprir as duas premissas anteriormente citadas, ele vem sendo aceito em projetos em todo o mundo. O DC vem se tornando o padrão *de facto* para a descrição de recursos digitais na Web. Uma outra vantagem que pode ser vislumbrada é o lançamento de bases para a implementação de interoperabilidade com outras bibliotecas digitais em uma versão futura do framework. À medida que o Diliframe entra no grupo das bibliotecas digitais que utilizam o DC para a descrição de seus recursos, ele, potencialmente, se credencia para trabalhar em conjunto com estas, segundo o nível de interoperabilidade acordado. O histórico, as particularidades e os benefícios obtidos pela escolha do padrão Dublin Core [DC, 2002] estão descritos no Anexo C.

No contexto do Diliframe, os elementos do Dublin Core serão utilizados para descrever os recursos e as coleções das bibliotecas digitais. Cada recurso deve corresponder a um e somente um registro de metadados[3]. Igualmente, cada coleção deve corresponder a um e somente um registro de metadados (Figura 9).

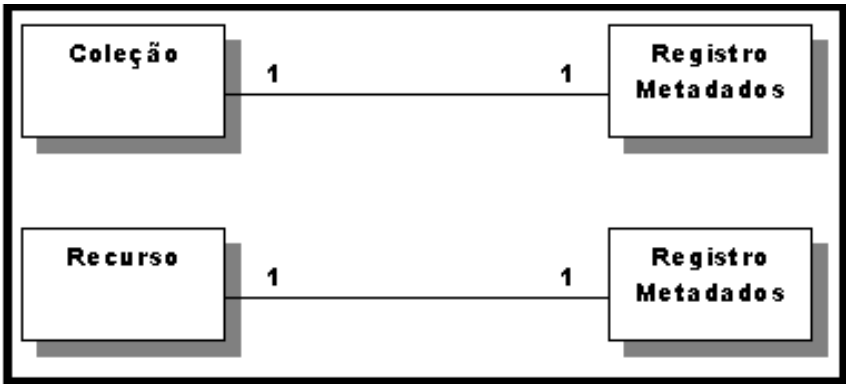


Figura 9 - Relacionamentos: registro de metadados X coleções e recursos

A consulta dos recursos, nas bibliotecas digitais geradas com o Diliframe, é realizada em cima dos atributos de metadados. Sendo assim, para que um determinado

[3] Como visto anteriormente, define-se o termo registro de metadados como sendo o conjunto de elementos do padrão de metadados, neste caso o Dublin Core, usado para descrever o recurso.

recurso seja disponibilizado na biblioteca digital, ele deve, obrigatoriamente, estar relacionado a uma coleção e ter o seu registro de metadados correspondente. Este procedimento é garantido pelo processo de cadastramento de recursos na biblioteca digital.

No Diliframe, os recursos estão separados dos metadados. Os metadados de todas as coleções e de seus recursos estão centralizados em um banco de dados distinto, enquanto que as coleções podem estar distribuídas entre diversos bancos de dados. A Figura 10 mostra este arranjo. Com este procedimento, procura-se simplificar o processo de consulta: todas elas serão direcionadas a um só banco de dados, independente da coleção consultada. Vale ressaltar, ainda, que a sobrecarga gerada pela distribuição da consulta aos diferentes servidores, a coleta das respostas e a consolidação dos resultados serão eliminadas. Por outro lado, a recuperação do recurso no servidor de dados não será feita de forma tão direta. Será necessário desenvolver mecanismos para a identificação das coleções e seus respectivos recursos. O registro de metadados manterá o relacionamento com o elemento que descreve através do elemento DC *identifier*.

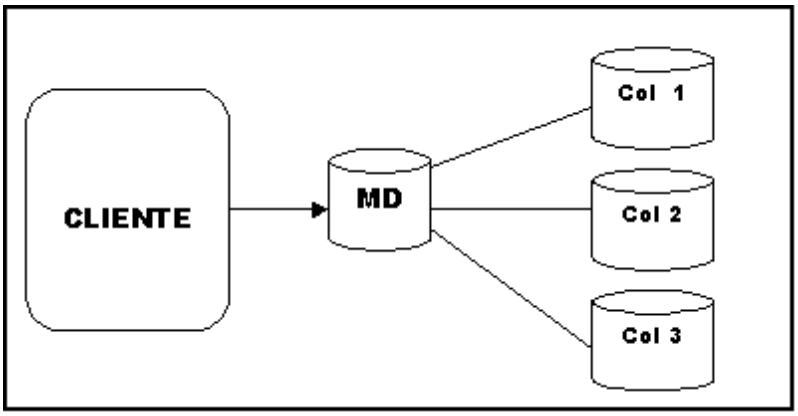


Figura 10 – Centralização dos registros de metadados

Assim como os recursos, as coleções também são relacionadas a registros de metadados. Este procedimento possibilita a obtenção de informações descritivas sobre a coleção. Além disso, permite manter uma maior visibilidade sobre as coleções que podem estar distribuídas na rede. O registro de metadados da coleção não lista os recursos que a ela estão relacionados. Esta identificação é feita no registro de metadados do recurso através do elemento DC *relation*.

Na implementação referência do Diliframe, algumas convenções sobre a forma de utilização do Dublin Core foram estabelecidas. Como, por exemplo, não ser admitido o uso de elementos qualificadores do Dublin Core para o registro de metadados. No entanto, alguns esquemas de codificação foram convencionados como vocabulário controlado para determinados elementos. Outras convenções dizem respeito à obrigatoriedade dos elementos e a possibilidade de repetição de valores para determinado elemento.

A Tabela 10 condensa as escolhas de projeto tomadas para a utilização de elementos do Dublin Core no Diliframe. A primeira coluna indica o elemento DC. A segunda coluna informa se este elemento é obrigatório ou opcional no registro de metadados. A terceira coluna indica se pode haver ou não repetição do elemento no registro, ou seja, para o elemento *creator*, que representa o autor intelectual do recurso, são admitidos vários valores contemplando os co-autores. A quarta coluna informa se o valor do elemento está restrito a um vocabulário controlado e que vocabulário é este. Abaixo de cada elemento são tecidas algumas considerações sobre a forma ou o significado do valor atribuído ao elemento.

	Obrigatoriedade	**Repetição**	**Vocabulário Controlado**
Title	Obrigatório	Sim	Não
	Considerando que a consulta pelo título do recurso é de extrema importância, determinou-se a obrigatoriedade deste elemento no registro de metadados. A repetição de valores para este elemento permite que sejam agregados outros títulos, como a tradução de um título, por exemplo. Quando o padrão estiver sendo usado para a descrição de uma coleção, o valor deste elemento não deve ser repetido.		
Creator	Obrigatório	Sim	Não
	Considerando que a consulta pelo autor do recurso é de extrema importância, determinou-se a obrigatoriedade deste elemento no registro de metadados. A repetição de valores para este elemento permite que sejam agregados co-autores, por exemplo, na descrição do elemento.		

Subject	Obrigatório	Sim	Sim
	Considerando que a consulta pelo assunto do recurso é de extrema importância, determinou-se a obrigatoriedade deste elemento no registro de metadados. A repetição de valores para este elemento permite que sejam atribuídos diversos assuntos à descrição do elemento, tornando a caracterização do recurso, segundo este aspecto, mais completa.		
Description	Obrigatório	Não	Não
	Considerando que a consulta pelas palavras chaves presentes na descrição do recurso é de extrema importância, determinou-se a obrigatoriedade deste elemento no registro de metadados. Este elemento não deve ser repetido uma vez que uma descrição completa é suficiente para a caracterização do recurso.		
Publisher	Opcional	Não	Não
	Considerando que este elemento não se aplica à descrição de recursos de bibliotecas digitais mais simples, ele foi caracterizado como opcional.		
Contributor	Opcional	Não	Não
	Considerando que este elemento pode ou não se aplicar à descrição de recursos, ele foi caracterizado como opcional.		
Date	Obrigatória	Não	Não
	O valor deste elemento deve corresponder a data de publicação do recurso na biblioteca digital. O formato determinado para o seu valor é: “AAAA-MM-DD”.		
Type	Opcional	Não	Não
	Considerando que este elemento pode dificultar a descrição do recurso por não-especialistas, e pouco contribuiria com o processo de consulta, ele foi caracterizado como opcional.		
Format	Obrigatório	Sim	Sim (Formato [MIME, 1996])

	Considerando que este elemento é de extrema importância para a caracterização do recurso e que pode contribuir em outros processos da biblioteca como, por exemplo, a recuperação do conteúdo, ele foi considerado obrigatório. A repetição deste elemento permite a completa caracterização do recurso no caso de recursos compostos. A utilização de um vocabulário controlado permite a padronização dos formatos e mídias disponíveis.		
Identifier	Obrigatório	Não	Não
	Este elemento é utilizado para manter a ligação do recurso com o seu registro de metadados. O formato utilizado para expressar o valor deste elemento deve garantir a identificação do recurso através de sua coleção e localização. O valor deste elemento é utilizado para desencadear o processo de recuperação do recurso. Quando este elemento estiver sendo usado para a descrição de uma coleção, seu valor deve ser mantido vazio.		
Source	Opcional	Não	Não
	Considerando que este elemento pode ou não se aplicar à descrição de recursos, ele foi caracterizado como opcional.		
Language	Opcional	Não	Não
	Considerando que a atribuição de valor a este elemento é uma opção de projeto particular de cada implementação de biblioteca, optou-se por caracterizá-lo como opcional.		
Relation	Obrigatório	Não	Não
	Este elemento é utilizado para indicar a coleção ao qual o recurso está relacionado. Na implementação do Diliframe um recurso pode pertencer a apenas uma coleção. Por isto não é possível a repetição de valores para este elemento. O valor deste elemento deve corresponder ao valor do elemento *Title* do registro de metadados da coleção referenciada. Quando este elemento estiver sendo usado para a descrição de uma coleção, o seu valor deve ser mantido vazio.		
Coverage	Opcional	Não	Não

	Considerando que este elemento pode ou não se aplicar à descrição de recursos, ele foi caracterizado como opcional.		
Rights	Opcional	Não	Não
	Considerando que este elemento pode ou não se aplicar à descrição de recursos, ele foi caracterizado como opcional.		

Tabela 10 – Convenções de utilização do Dublin Core no Diliframe

3.3.2 Modelo arquitetural

Nesta seção será apresentado o modelo arquitetural do Diliframe. Este artefato trata-se de um conjunto de decisões de projeto, escolhas de estratégias e diagramas, referentes à arquitetura do framework. O modelo arquitetural é resultado de um cuidadoso estudo sobre como transformar em software os requisitos definidos no segundo capítulo desta dissertação. Com o modelo arquitetural inicia-se a fase de projeto.

A primeira grande decisão de projeto tomada para o Diliframe foi a adoção da plataforma J2EE para o seu desenvolvimento. O Anexo D apresenta a arquitetura J2EE – Java 2 *Enterpise Edition* [Sun Microsystems, 2002], os seus aspectos técnicos e detalhes de implementação. Esta abordagem é necessária, pois as características arquiteturais de alto nível do Diliframe, tendo o seu desenvolvimento baseado em J2EE, foram herdadas desta plataforma.

Outras decisões de projeto dizem respeito a: subdivisão do sistema em camadas e a atribuição de responsabilidades em cada uma delas; estruturação do sistema em módulos funcionais e a determinação de suas atribuições; aplicação da arquitetura *Model-View-Controller* (MVC) e ao detalhamento dos módulos que atuam em cada uma de suas camadas.

No geral, as estratégias adotadas para garantir um nível mais alto de reuso do framework foram:

- Maximizar a modularização da aplicação;
- Isolar, sempre que possível, pontos do código que não são estáveis em arquivos de configuração (XML, por exemplo);

- Elevar o nível das classes de negócio através da construção de interfaces ou classes abstratas;

A fim de cumprir tais preceitos, o Diliframe faz amplo uso de padrões de projeto J2EE [Crupi, 2001].

De acordo com a arquitetura de alto nível do Diliframe, as camadas nas quais estará dividida uma aplicação de biblioteca digital gerada com o Diliframe são as seguintes:

- Camada Cliente – representada pelo browser;
- Camada Web – implementada no servidor;
- Camada EJB – implementada no servidor;
- Camada de dados – representada pelo banco de dados que conterão os dados da aplicação.

Na camada cliente, está localizada a interface com o usuário do sistema de biblioteca digital. Por ser uma aplicação Web, está interface será apresentada dentro de um browser. Ela é responsável por efetivar a comunicação entre o usuário e o sistema: recolhendo as informações que o usuário deseja enviar e apresentando os dados enviados pelo sistema.

A camada Web responsabiliza-se pela geração do conteúdo dinâmico das páginas e pelo encaminhamento das requisições dos usuários. A camada Web não é responsável por processar a requisição ou o acesso aos dados armazenados. Ela encaminha o pedido para o componente de negócio que, efetivamente, irá realizá-lo.

No Diliframe, a lógica de negócio e, principalmente, o acesso aos dados serão implementados na camada EJB. Esta estratégia é uma recomendação de boas práticas no desenvolvimento de aplicações corporativas J2EE feitas pela *Sun*. A arquitetura EJB oferece um paradigma de programação que promove o encapsulamento e o uso de componentes, o que resulta em um software mais fácil de gerir, à medida que, a aplicação torna-se mais complexa [Singh, 2002]. Além do mais, os serviços oferecidos aos componentes pelo container EJB facilitam sobremaneira a implementação de aplicações corporativas escaláveis.

A camada de dados do Diliframe armazenará informações sobre usuários, recursos e coleções. De acordo com o que foi definido nos requisitos funcionais deste framework, as coleções podem estar distribuídas e armazenadas em bancos de dados de

diferentes fornecedores. Esta configuração é um fator complicador a ser gerenciado pelos componentes EJBs que fazem acesso aos dados.

Para facilitar o desenvolvimento das aplicações e garantir a qualidade de seu projeto optou-se por utilizar a arquitetura definida pelo padrão *Model-View-Controller* aplicada a sistemas Web. Esta arquitetura divide a aplicação em três camadas lógicas: a *View,* responsável pelas questões de apresentação das aplicações; a *Model,* que se responsabiliza pelas regras de negócio e acesso aos dados; e a *Controller* que recebe e interpreta as requisições dos usuários, encaminha-as para os objetos de negócio designados para o cumprimento da tarefa e seleciona a próxima página da *view* a ser apresentada.

A utilização da arquitetura MVC em aplicações J2EE é uma das recomendações de boas práticas de desenvolvimento existentes na aplicação referência definida pela especificação J2EE [Sun Microsystems, 2002]. O projeto da Dilib, visando à qualidade do resultado final, procura seguir recomendações e reaproveitar artifícios encontrados na aplicação referência que acompanha a especificação. Este comportamento é observado, especialmente, nas camadas *View* e *Controller* da arquitetura do Diliframe. Na *Model,* onde a lógica de negócio é definida, há uma maior particularização da aplicação. Nesta camada, o projeto se apóia fortemente nos padrões de projeto J2EE, também recomendados pela *Sun* [Crupi, 2001].

A proposta deste framework é definir uma arquitetura completa para as aplicações, ou seja, uma arquitetura abrangente que englobe todas as camadas da arquitetura MVC para a biblioteca digital e, além disso, forneça recursos que auxiliem na montagem da biblioteca. Com base em tais premissas, o Diliframe foi dividido em três sub-aplicações que fazem parte de frentes distintas: no *front-end* encontra-se a "Dilib" e, no *back-end,* encontram-se a "Repository Maker" e a "Metadata Maker".

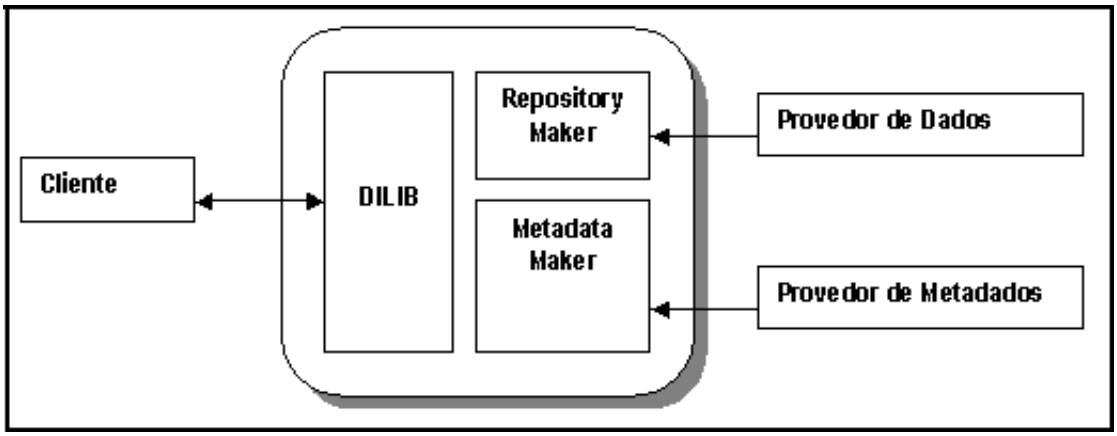


Figura 11 - Sub-Aplicações do Diliframe

A Dilib é a interface Web da biblioteca digital gerada com o Diliframe. A ela têm acesso os usuários comuns, os administradores de usuários e os administradores de dados. Desta forma, sob uma interface comum são oferecidas funcionalidades distintas a tipos de usuários diferentes. A Figura 12 representa o diagrama de alguns casos de uso, discutidos na sub-seção 3.2.2, descrevendo a interação entre os atores citados e a Dilib.

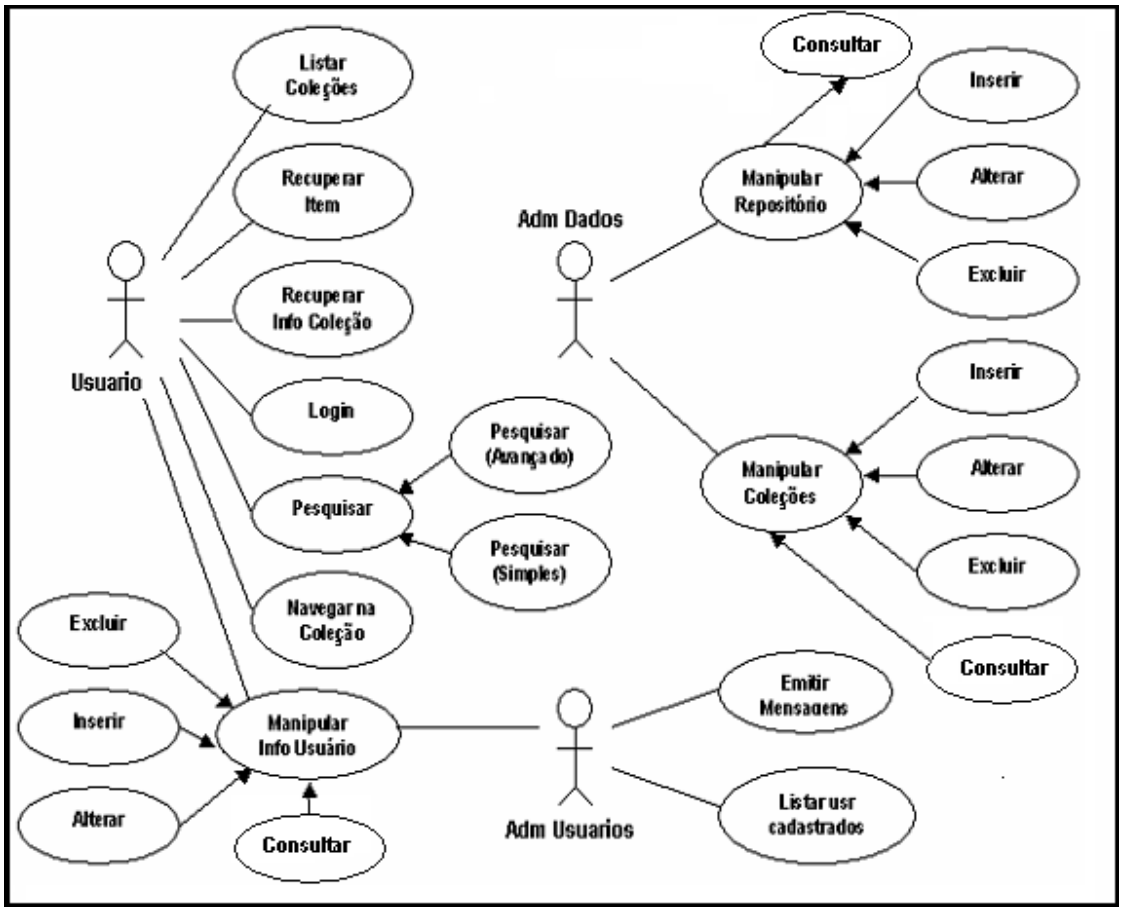


Figura 12 - Diagrama de casos de uso da Dilib

Cabem aqui algumas considerações sobre as ações desempenhadas pelos administradores de usuários e administradores de dados. Dentro do conjunto de requisitos funcionais, relacionados para o Diliframe, consta a necessidade de oferecer ferramentas para a administração do catálogo de usuários e para a administração das coleções. É através dos módulos da Dilib que estes requisitos serão satisfeitos.

A segunda sub-aplicação do Diliframe, denominada *Metadata Maker,* é utilizada para auxiliar a montagem do repositório centralizado de metadados. Ela permite que o provedor de metadados crie o registro de metadados, a partir da aplicação, e que, também a partir dela, alimente o repositório de metadados. A *Metadata Maker* permite, ainda, que a carga do repositório de metadados seja feita a partir de registros já existentes formatados em arquivos XML de acordo com o Dublin Core. Esta funcionalidade facilita a adição de novas coleções que já contam com seus registros de metadados prontos.

A sub-aplicação *Metadata Maker* é responsável por garantir a consistência dos elos de ligação do registro de metadados ao recurso que ele descreve e deste recurso à coleção à qual está associado. Estes relacionamentos são mantidos através dos elementos Dublin Core *identifier* e *relation*, respectivamente. A Figura 13 representa o diagrama de casos de uso que descreve a interação entre o ator provedor de metadados e a sub-aplicação *Metadata Maker*.

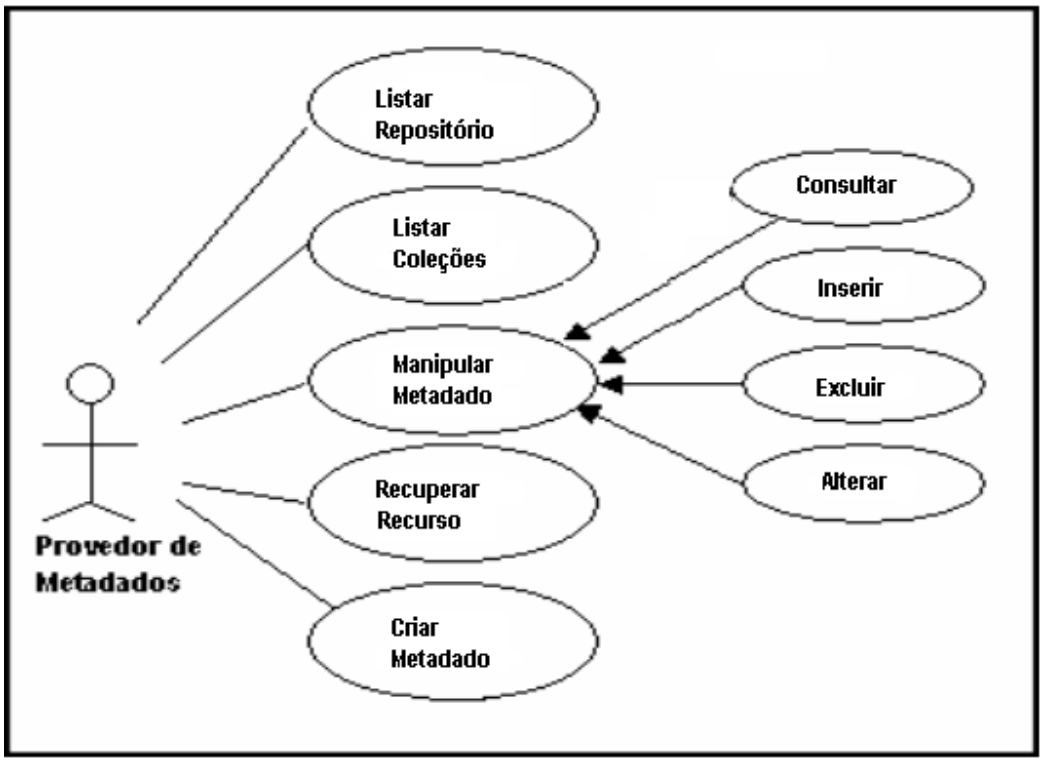


Figura 13 - Diagrama de casos de uso da *MetadataMaker*

A terceira sub-aplicação do Diliframe é o *Repository Maker*. O seu objetivo é facilitar o povoamento do repositório fazendo a carga do banco de dados de recursos. O *Repository Maker* é útil apenas nos casos em que o banco ainda não está carregado. Sua funcionalidade é a mesma que a desempenhada por um *script* de carga de dados. Entretanto, na maioria das vezes os dados das bibliotecas digitais são dados multimídia e não dados em formato de tabelas, logo, a utilização de um instrumento para fazer a carga de dados facilita muito esta tarefa.

Junto à *Repository Maker* é disponibilizado o *script* de criação das tabelas do modelo de dados do Diliframe. Não é da alçada do provedor de dados o cadastramento do repositório na biblioteca. Esta atividade é de responsabilidade do administrador de dados e é realizada na sub-aplicação Dilib. A Figura 14 representa o diagrama de casos de uso que descreve a interação entre o ator "provedor de dados" e a sub-aplicação *Repository Maker.*

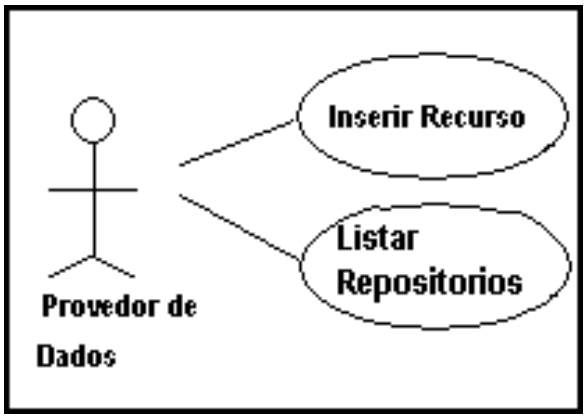


Figura 14 - Diagrama de casos de uso da *RepositoryMaker*

3.3.2.1 A arquitetura da Dilib

A sub-aplicação Dilib é o sistema Web que implementa a biblioteca digital. A Dilib oferece serviços aos atores do sistema: usuários, administradores de usuários e administradores de dados. Como parte do processo de definição da arquitetura é necessário realizar a sub-divisão da sub-aplicação em objetos e componentes associando-os às camadas correspondentes da arquitetura MVC. Este processo é a decomposição em objetos [Singh, 2002].

Algumas premissas devem ser observadas na decomposição dos módulos funcionais em objetos. Estas premissas visam a assegurar os requisitos fundamentais para o desenvolvimento do framework: flexibilidade, adaptabilidade, modularização e reusabilidade. Desta forma, foram delineadas seguintes necessidades:

- Estabelecer fronteiras bem definidas entre o código que raramente muda, daquele que muda normalmente;
- Modularizar funcionalidades de forma a permitir fácil substituição do módulo ou extensão visando particularização;

- Facilitar a atribuição de responsabilidades na equipe de desenvolvimento, através da modularização de funcionalidades, de acordo com a tecnologia de desenvolvimento. Por exemplo, o isolamento de código nas páginas Web – que é de responsabilidade dos programadores da camada Web – da programação visual – responsabilidade dos Web designers.

Na arquitetura da Dilib estão previstos módulos funcionais que modelam a aplicação de acordo com a arquitetura do MVC. A Figura 15 representa graficamente a subdivisão da Dilib em seus módulos funcionais.

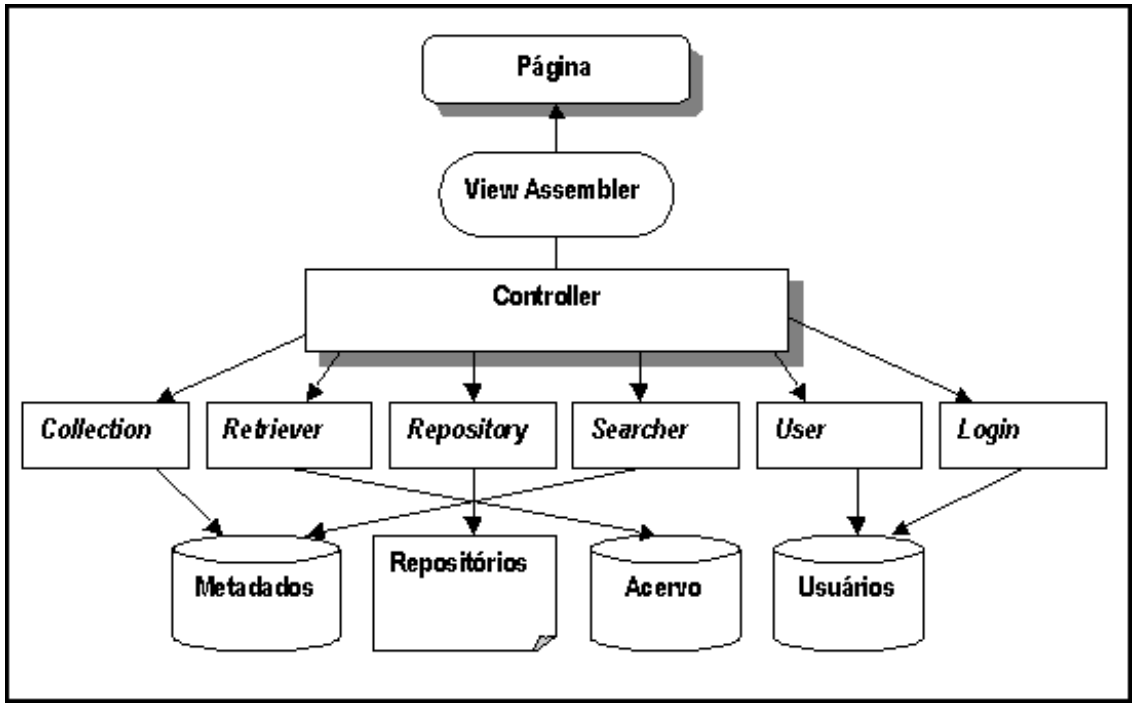


Figura 15 - Módulos Funcionais da Dilib

– *View Assembler*: A modularização das páginas da View, além de obedecer aos requisitos definidos para o desenvolvimento do Diliframe, facilita a gerência e criação da camada de apresentação. O padrão de projeto responsável por detalhar esta idéia é o *Composite View*. Ele parte do princípio que todas as páginas da aplicação tem uma estrutura fundamental, por exemplo: cabeçalho, conteúdo e rodapé. Assim, as páginas são resultado de uma composição [Crupi, 2001].

"Uma Composite View é construída usando outras views[4] reutilizáveis. Caso ocorram mudanças em alguma dessas views reutilizáveis, elas são automaticamente refletidas às views compostas que a reutilizam."

As classes deste módulo são responsáveis pela implementação deste conceito. Com relação à arquitetura MVC o *View Assembler* localiza-se na camada *View*.

- *Controller*: O módulo *Controller* faz a ponte entre a camada de apresentação - *View* e a camada da aplicação – *Model.* É recomendado que esta ponte seja feita a partir de um único ponto de acesso. Esta estratégia visa a facilitar o controle dos serviços oferecidos e o tratamento a eles dispensados. O padrão de projeto *Front Controller* consolida esta idéia [Crupi, 2001].

"O padrão de projeto Front Controller define um único componente que é responsável por processar as requisições da aplicação. Um controlador desta natureza centraliza determinadas funções como: seleção da próxima página a ser exibida, segurança e montagem da página. E, além disso, deve aplicá-las de forma consistente a todas as páginas. Conseqüentemente, quando o comportamento de alguma dessas funções muda, apenas uma pequena porção de código da aplicação precisa ser alterado: o controlador e suas classes auxiliares".

Uma outra funcionalidade relevante associada ao módulo *Controller* é um filtro de requisições. Antes de chegar ao controlador efetivamente, as requisições são filtradas, ou seja, analisadas a fim de verificar se há alguma restrição para o seu encaminhamento. Na Dilib o filtro pode ser utilizado para implementar as restrições de segurança ou os mecanismos de cobrança. As idéias apresentadas no padrão de projeto *Intercepting Filter* são utilizadas na implementação desta funcionalidade [Crupi, 2001].

"O padrão de projeto Intercepting Filter protege as aplicações com um filtro que intercepta a recepção da requisição e a transmissão da resposta. O filtro de interceptação pode ser usado para pré-processar ou redirecionar as requisições; ou, ainda, para pós-processar ou substituir o conteúdo das respostas para a aplicação. Filtros diferentes

[4] O termo "View", utilizado nesta seção, refere-se às páginas web do site. Sabe-se que a camada View da arquitetura MVC não se limita, apenas, à apresentação no formato web. Entretanto, como esse é o caso deste projeto e para facilitar o entendimento do texto, optou-se por assim descrevê-lo.

podem ser enfileirados, formando uma cadeia e com isso adicionando mais serviços à aplicação, sem alteração de seu código".

Com relação à arquitetura MVC, este módulo localiza-se na camada *Controller*. A Figura 16 apresenta a seqüência de interação de uma requisição de serviço passando por uma cadeia de filtros de interceptação.

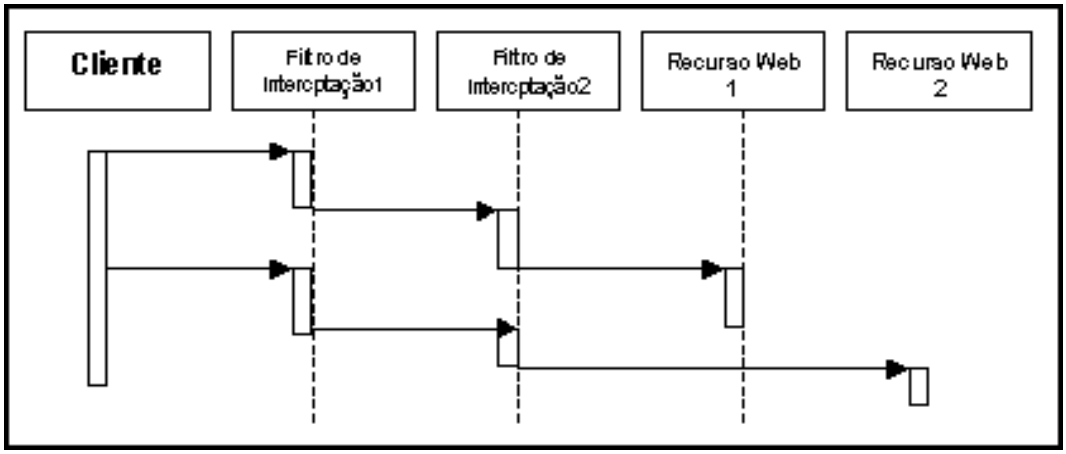


Figura 16 - Filtro de Interceptação

- *Login*: Este módulo funcional é responsável por realizar a autorização de acesso de usuários à biblioteca digital. Esta autorização é feita através do par nome do usuário e senha, informado por um usuário previamente cadastrado. As informações passadas pelo usuário são utilizadas para verificar a existência do mesmo no cadastro de usuários da biblioteca.

O módulo de *Login* não é um módulo complexo nem apresenta requisições especiais do ponto de vista arquitetural. Conseqüentemente, não houve necessidade da utilização de padrões para orientar o seu projeto. Com relação à arquitetura MVC o *Login* localiza-se na camada *Model*.

- *User*: O módulo funcional *User* foi criado para representar o usuário no escopo da biblioteca digital. O usuário é uma entidade persistente, cujos atributos e características devem ser armazenados em bancos de dados. A entidade usuário não se limita apenas aos atributos nome e senha. Os dados cadastrais, tais como: email, endereço, telefone, instituição, dentre outros; também são importantes. E, além disso, o perfil do usuário também deve ser armazenado. No perfil, podem constar dados sobre cobrança,

permissões de acesso, preferências utilizadas nos serviços personalizados – por exemplo, *bookmarks*.

A escolha dos atributos do usuário, além de nome e senha, é flexível, ou seja, pode ser redefinida pelo desenvolvedor da aplicação. Estas definições são bastante particulares para cada aplicação, pois estão diretamente relacionadas ao escopo e à missão da biblioteca que está sendo criada. A Figura 17 representa graficamente o relacionamento entre o usuário e as informações a associadas ele. Com relação à arquitetura MVC, o *User* localiza-se na camada *Model*.

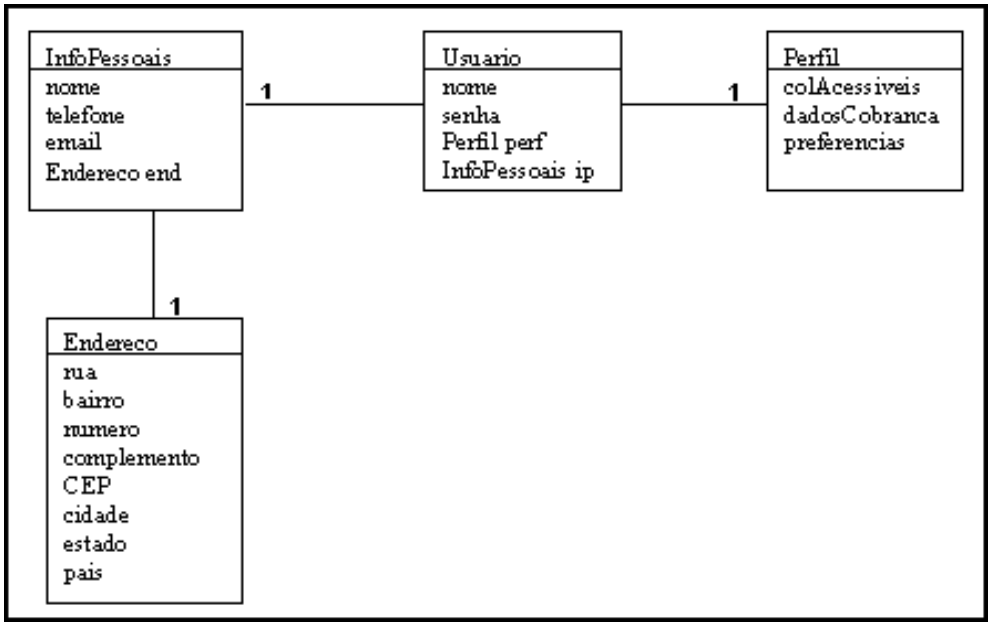


Figura 17 - Relacionamento entre os dados do *User*

– *Searcher*: Este módulo funcional é responsável por implementar as funcionalidades de pesquisa na biblioteca digital que podem ser de dois tipos: simples ou avançadas. As primeiras consideram sentenças de pesquisa com um item de consulta em apenas um elemento. As segundas oferecem maior flexibilidade, permitindo a combinação de vários itens de consulta através de operadores lógicos em um ou mais elementos.

As consultas realizadas na Dilib são feitas sobre uma base de dados que centraliza os registros de metadados dos recursos e coleções. O conjunto resultado desta operação não apresenta os recursos, mas sim, uma lista com a descrição resumida daqueles que atendem às restrições de consulta da sentença de pesquisa.

O resultado da consulta é apresentado ao usuário através de uma página de resposta apropriada. O *Searcher* deve dispor um conjunto de respostas consolidado, com os atributos que devem ser apresentados, para o módulo responsável por montar a página de resposta para o usuário. Dentre os elementos do conjunto resposta, deve constar obrigatoriamente o *id*. Este elemento representa o elo de ligação entre o registro de metadados que o descreve e a sua localização física. Com relação à arquitetura MVC, o *Searcher* localiza-se na camada *Model*.

- *Retriever*: Este módulo funcional é responsável pela recuperação de recursos. Os recursos estão distribuídos entre repositórios que podem estar localizados em diferentes servidores de bancos de dados. Um sistema de identificação e mapeamento dos repositórios é definido pelo módulo *Retriever.* Esta identificação viabiliza o acesso ao repositório específico e define as sentenças de pesquisa que irão recuperar os recursos com base no seu identificador.

O *Retriever* tem acesso ao recurso através de classes especiais de conexão. Estas classes têm o objetivo de isolar o código que implementa o acesso aos dados. Isto permite que a substituição da fonte de dados por outra aconteça de forma localizada e evita que o código da aplicação seja alterado. O padrão de projeto *Data Access Object* descreve esta idéia [Crupi, 2001].

"O padrão de projeto Data Access Object ou, DAO, procura dissociar o código cliente, que acessa os dados, do mecanismo de acesso efetivamente. Além disso, faz a adaptação de mecanismos específicos de acesso aos dados, para uma interface cliente genérica".

A utilização do DAO é especialmente útil para este framework, pois viabiliza o acesso a bancos de dados de fornecedores diferentes, através de uma interface comum. A estratégia adotada para tanto foi a de utilizar arquivos de configuração XML, para definir o banco de dados que será acessado, sua localização e até os comandos SQL utilizados, por exemplo, para recuperar o recurso. Com relação à arquitetura MVC, o *Retriever* localiza-se na camada *Model*.

- *Collection*: Este módulo funcional é utilizado pelo administrador de dados para manipular as coleções. Através dele novas coleções podem ser criadas, excluídas ou terem sua descrição alterada. A criação de uma coleção consiste no cadastramento de

um registro de metadados referente a uma coleção no repositório centralizado de metadados. Para a adição de recursos a esta coleção, a sub-aplicação *Metadata Maker* deve ser utilizada. Apenas o recurso faz referência à coleção, o contrário não é verdadeiro. Por conseguinte, a exclusão de uma coleção não implica na exclusão de seus recursos. Eles, apenas, tornar-se-ão inacessíveis. O processo de alteração das coleções é feito em cima das informações cadastradas no repositório central de metadados.

Uma coleção deve estar associada à apenas um repositório, a fim de garantir que seus recursos estejam todos agrupados. As coleções são entidades persistentes, sendo assim, implementadas como *entity beans.* Com relação à arquitetura MVC, o *Collection* localiza-se na camada *Model*.

- *Repository*: Este módulo funcional é utilizado pelo administrador de dados para cadastrar novos repositórios à biblioteca digital. Este cadastramento consiste na inclusão do repositório no esquema de identificação utilizado pelo módulo *Searcher* e nos arquivos de configuração DAO do módulo *Retriever.* Esta identificação viabiliza o acesso ao repositório e define as sentenças de pesquisa que irão recuperar os recursos com base no seu identificador.

Este módulo atualiza arquivos de configuração XML e não registros no banco de dados. Desta forma, optou-se por não utilizar *entity beans* com persistência gerenciada pelo *container* para a implementação da entidade *Repository*. Com relação à arquitetura MVC, o *Repository* localiza-se na camada *Model.*

3.3.2.2 A arquitetura da MetadataMaker

A *MetadataMaker* é uma aplicação cliente utilizada para a realizar o cadastro dos registros de metadados dos recursos disponibilizados pela biblioteca digital. De acordo com o que foi apresentado neste capítulo com relação a metadados, um registro de metadados comporta informações que descrevem um recurso com base em um conjunto de elementos de características. Convencionou-se utilizar um sub-conjunto de elementos do Dublin Core como padrão para a descrição dos recursos do Diliframe.

No Diliframe, os recursos e os metadados estão armazenados separadamente. Os metadados encontram-se em um repositório centralizado, enquanto que os recursos estão dispersos. A centralização dos metadados facilita o processo de consulta em múltiplas

coleções, relacionadas a repositórios distintos. Todavia, este esquema de centralização implica em um delicado controle de consistência envolvendo o recurso e seus metadados. Esta é a causa das maiores dificuldades na implementação deste esquema.

Além do relacionamento físico com o recurso, o registro de metadados relaciona-se, em nível superior de abstração, com a coleção. Uma coleção, como visto anteriormente, é um agrupamento de recursos de acordo com determinada classificação. Todo registro de metadados deve relacionar-se a uma e somente coleção. Um recurso não deve fazer parte de mais de uma coleção. A consistência deste relacionamento também deve ser assegurada pela *MetadataMaker*.

A arquitetura da sub-aplicação *MetadataMaker*, assim como a da Dilib, segue a arquitetura proposta pelo padrão MVC. Diferentemente da Dilib, a camada *View* da *MetadataMaker* não é implementada por páginas Web. Optou-se por implementá-la como uma aplicação J2EE cliente, com acesso remoto aos componentes da *Model* através dos componentes da *Controller*. O motivo que levou a separação desta sub-aplicação da Dilib foi definir mais claramente o papel do "provedor de metadados", isolando na Dilib as funcionalidades administrativas relacionadas apenas aos repositórios, coleções e usuários.

A divisão em camadas e a modularização das funcionalidades, que são características desta arquitetura, permite ao desenvolvedor a substituição da *View*. Sendo assim, pode ser criada uma *View* Web para a *MetadataMaker* reutilizando os componentes de negócio da *Model*. Mais ainda, caso julgue necessário, o desenvolvedor pode acoplar a *MetadataMaker* à Dilib.

A divisão em módulos funcionais que se encaixam na arquitetura Model-View-Controller para a sub-aplicação *MetadataMaker* é representada graficamente na Figura 18.

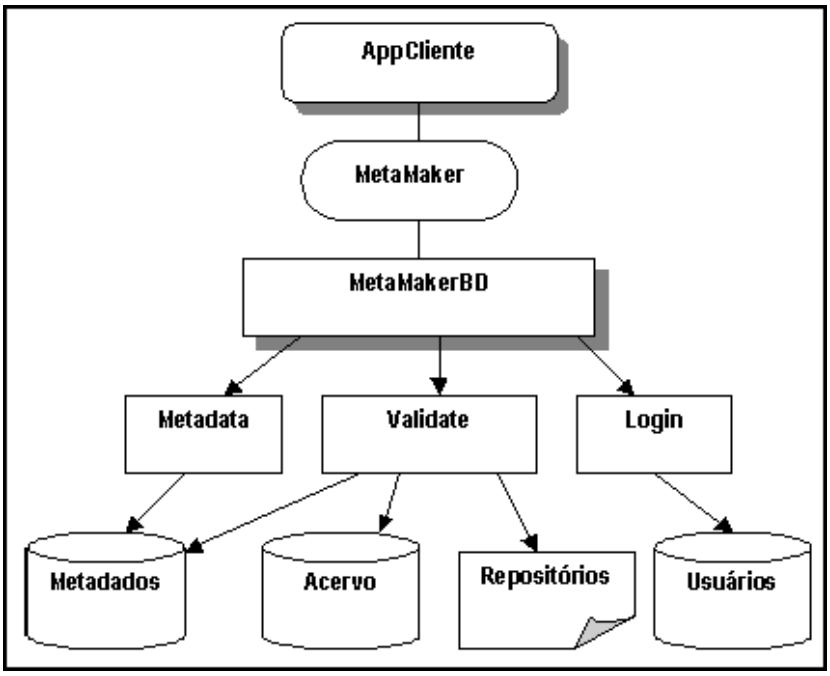


Figura 18 - Módulos Funcionais da *MetadataMaker*

- *MetaMaker:* Este módulo funcional faz a ponte entre as classes de interface gráfica apresentadas ao usuário e os componentes de negócio desta aplicação. O *MetaMaker* segue a idéia básica do padrão de projeto *Front Controller* [Crupi, 2001]: *"Estabelecer um componente único de negócio responsável por processar as solicitações do usuário, encaminhá-las aos componentes de negócio e definir como será feita a sua apresentação"*.

  Na arquitetura MVC, este módulo está enquadrado na camada *Controller*.
- *MetaMakerBD:* Este módulo funcional é responsável por encaminhar as solicitações do cliente aos componentes de negócio que, efetivamente, executam uma função. O *MetaMakerBD* faz acesso a objetos distribuídos, isto implica em ações como: *Lookup*[5] e tratamento de exceções remotas. O encapsulamento destas complexidades torna o código da aplicação mais limpo, além de facilitar a sua manutenção e reutilização. O padrão de projeto *Business Delegate,* no qual se baseia o *MetaMakerBD,* elabora esta idéia [Crupi, 2001]:

[5] Localização de objetos remotos com base em um serviço de nomes. No caso da plataforma J2EE é utilizado JNDI.

"O Business Delegate desacopla os componentes de negócio do código que os utiliza. Este padrão de projeto sugere a criação de uma classe que faça a gerência da complexidade advinda da utilização de componentes distribuídos. Esta classe deve ser responsável pelas operações de lookup e manipulação de exceções remotas".

De acordo com as camadas da arquitetura MVC, este módulo encaixa-se na fronteira entre a *Controller* e a *Model*.

- *Metadata:* Este módulo funcional contém a representação do registro de metadados. O módulo funcional *Metadata* viabiliza a manipulação de registros de metadados que caracterizam recursos, permitindo as ações de criação de novos registros, alteração e exclusão dos já existentes. A ação de criação de novos recursos deve assegurar a consistência nas informações. Esta tarefa é assegurada pelo módulo funcional *Validate* antes de, efetivamente, ser criado um novo registro de metadados. O módulo funcional *Metadata* situa-se na camada *Model* da arquitetura MVC.
- *Validate:* Este módulo funcional tem como principal função assegurar a consistência das informações dos registros de metadados que estão sendo criados. Tais informações dizem respeito, mais especificamente, ao elemento *Identifier* e ao elemento *Relation*. O *Identifier,* que é a identificação do recurso, guarda informações chave que possibilitam a sua recuperação. É a lei de formação do valor deste elemento que permite que ele seja devolvido ao usuário pelo módulo *Retriever* da Dilib (sub-seção 3.3.2.1). A string que forma o *Identifier* deve ser composta por:

Id = / <repositório>/<coleção>/<PKrecurso>
Onde: Pkrecurso é a chave primária do recurso na tabela do banco de dados

Tabela 11 – Identificador do recurso

O módulo funcional *Validate* deve assegurar a validade de cada parte desta string de forma a obter um identificador válido para o recurso. As questões a serem consideradas no processo de validação são:

1. O repositório informado foi cadastrado na Dilib?
2. A coleção informada foi cadastrada na Dilib?
3. A coleção informada é referente ao repositório informado?
4. É possível ter acesso ao recurso a partir das informações repositório, coleção e Pkrecurso?

O elemento *Relation* refere-se à coleção à qual o recurso está associado. As questões a serem observadas no processo de validação deste elemento são:

1. A coleção informada foi cadastrada na Dilib?
2. A coleção informada é referente ao repositório informado?
3. A coleção informada é idêntica à segunda parte da string que constitui o valor do elemento *id* deste registro?

O registro de metadados será considerado válido se responder de forma afirmativa a todas estas questões e, ainda, aos preceitos de obrigatoriedade de determinados elementos de metadados (determinados na seção 3.3.1). A obtenção de respostas para estas questões é feita através de consultas às entidades persistentes adequadas. As informações sobre o repositório são obtidas a partir do módulo funcional *Repository*. Ele informa todos os repositórios que foram cadastrados na Dilib através do acesso à lista de repositórios. As informações sobre as coleções podem ser obtidas através do EJB *Metadata*. As coleções cadastradas na Dilib constam no repositório centralizado de metadados. As diferenças entre um registro de recurso e um registro de coleção estão nos elementos: *identifier* – na descrição de uma coleção, o valor deste elemento não possui a terceira parte da string – e no *relation* – este elemento não apresenta valor. Com base nesta última característica, é possível, facilmente, identificar um registro de coleção e, conseqüentemente, validar a coleção informada. Para validar se a coleção refere-se ao repositório informado, basta verificar o valor do elemento *identifier*. Para assegurar-se que o recurso realmente é acessível através das informações passadas, o módulo *Validate* tenta fazer a recuperação do recurso através dos componentes constituintes do módulo *Retriever*. Se a recuperação for bem sucedida o *identifier* foi considerado válido.

- *Login:* Este módulo funcional é o mesmo existente na sub-aplicação Dilib. Todavia, a autorização do acesso às funcionalidades do módulo *MetaMaker* é obrigatória. Apenas os usuários do tipo "provedor de metadados" podem ter acesso à *MetadataMaker*. Este usuário especial deve ser cadastrado na Dilib pelo administrador de usuários.

3.3.2.3 A arquitetura da RepositoryMaker

A *RepositoryMaker* é uma aplicação cliente utilizada para povoar os bancos de dados dos repositórios que servem à Dilib. O objetivo desta sub-aplicação é facilitar a inserção de recursos no repositório, elevando o nível de abstração no que se refere ao acesso aos dados. O "provedor de dados", a quem se destina esta sub-aplicação, não teria mais necessidade de escrever código SQL para a inserção no banco de dados. Um importante benefício atingido com a utilização da *RepositoryMaker,* é observado quando do cadastramento de dados multimídia – imagem, som, vídeo, etc. A inserção destes dados através de uma aplicação específica é bem mais fácil que através da utilização de scripts.

Os recursos armazenados nas tabelas dos repositórios devem ter um identificador único que viabilize a sua recuperação. A garantia de consistência deste valor deve ser assegurada pela *RepositoryMaker.* Esta sub-aplicação também disponibiliza a função de criação das tabelas do repositório.

A arquitetura da sub-aplicação *RepositoryMaker*, de forma análoga à da *MetadataMaker* e à da Dilib, segue a arquitetura proposta pelo padrão MVC. Assim como a *MetadataMaker*, esta sub-aplicação também implementa a camada *View* como uma aplicação J2EE cliente, com acesso remoto aos componentes da *Model* através dos componentes da *Controller*. Esta estratégia objetiva definir mais claramente o papel do "provedor de dados", isolando na Dilib as funcionalidades administrativas relacionadas apenas aos repositórios, coleções e usuários.

A divisão em módulos funcionais que se encaixam na arquitetura MVC para a sub-aplicação *RepositoryMaker* são representadas graficamente na Figura 19.

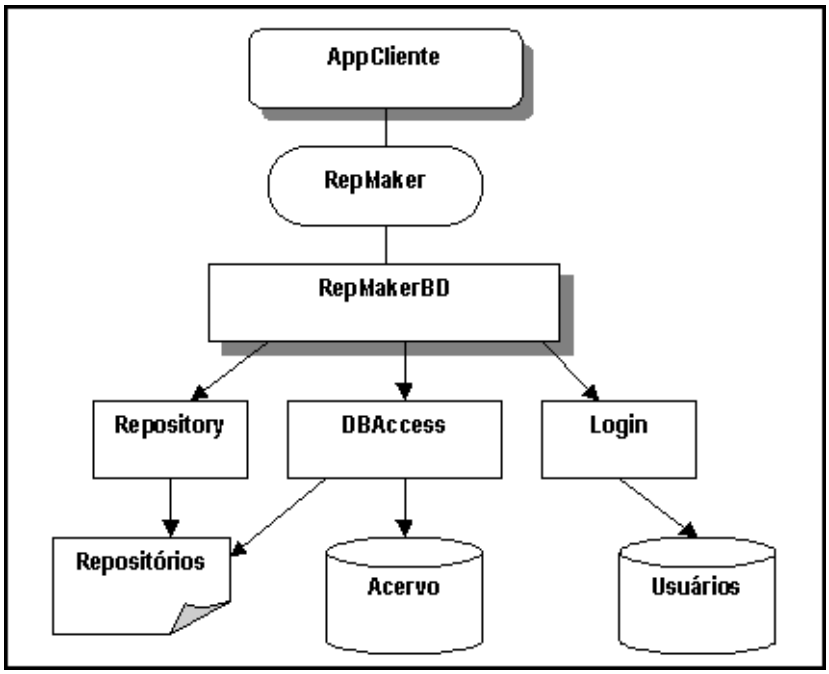


Figura 19 - Módulos funcionais da *RepositoryMaker*

- *RepMaker:* Este módulo funcional tem o mesmo papel que o seu correspondente *MetaMaker* no contexto daquela sub-aplicação. Ele é o elo de ligação entre as classes de interface gráfica apresentadas ao usuário e os componentes de negócio. A idéia defendida no padrão de projeto *Front Controller* [Crupi, 2001] é seguida por este módulo funcional. Na arquitetura MVC este módulo está enquadrado na camada *Controller*.
- *RepMakerBD:* Assim como o seu correspondente *MetaMakerBD,* este módulo funcional faz o encaminhamento das solicitações do cliente aos componentes de negócio que executam as ações. O *RepMakerBD* lida com objetos distribuídos, o que implica na realização de operações como: *Lookup* e tratamento de exceções remotas. O padrão de projeto *Business Delegate,* no qual se baseia este módulo, sugere o encapsulamento das complexidades, ao acessar objetos remotos em uma única classe [Crupi, 2001]. De acordo com as camadas da arquitetura MVC, este módulo encaixa-se na fronteira entre a *Controller* e a *Model*.
- *Login:* Este módulo funcional é o mesmo existente na sub-aplicação Dilib. O acesso às funcionalidades do módulo *RepMaker* será autorizado apenas a usuários do tipo

"provedor de dados". O procedimento de *Login* é obrigatório. Os usuários devem ser cadastrados, previamente, pelo administrador de usuários na sub-aplicação Dilib.

- *Repository:* Este módulo também é o mesmo utilizado pela sub-aplicação Dilib. Entretanto, na *RepositoryMaker,* ele é utilizado apenas para listar os repositórios cadastrados na Dilib. Como premissa, os recursos só podem ser inseridos em repositórios já cadastrados.
- *DBAccess:* Através deste módulo funcional os recursos são, efetivamente, cadastrados nos bancos de dados. Fazem parte deste módulo, classes que implementam conexão com o banco de dados. Estas classes, assim como as definidas para o módulo *Retriever* da sub-aplicação Dilib, seguem a idéia do padrão de projeto DAO – Data Access Object [Crupi, 2001].

  Para uma operação de inserção de dados, o primeiro passo é selecionar o repositório. Nos arquivos de configuração XML do repositório, o *DBAccess*, tomando como ponto de partida o banco de dados em questão, obtém as informações necessárias para a geração das sentenças de consulta SQL. Uma vez obtidos os comandos SQL correspondentes, o método de inserção é acionado e o registro inserido no banco. Desta forma, a alimentação de bancos de dados independentemente de seu fornecedor pode ser facilmente implementada.

Capítulo 4

Aspectos do Projeto do Diliframe

O presente capítulo objetiva detalhar o projeto deste framework para bibliotecas digitais. Este capítulo apresenta uma visão mais aprofundada do modelo arquitetural do Diliframe visto no capítulo anterior, incluindo decisões de projeto mais próximas das questões tecnológicas e de implementação. Será apresentada, também, a estratégia de implementação do framework que se reflete nas iterações do processo de desenvolvimento a serem cumpridas.

Com este propósito o capítulo divide-se em duas partes. A primeira, referente ao projeto do Diliframe, explora cada uma das sub-aplicações que o constituem: a *Dilib*, a *MetadataMaker* e a *RepositoryMaker*. As questões relevantes ao projeto de seus módulos funcionais serão demonstradas. Características de implementação destes módulos, consideradas importantes para a compreensão do contexto, também serão levantadas. A segunda parte do capítulo destina-se às questões de implementação do Diliframe. Serão traçadas as iterações necessárias para a sua implementação.

4.1 Projeto do Diliframe

No capítulo anterior, foi apresentado o projeto arquitetural do Diliframe. Observou-se a subdivisão do framework em três sub-aplicações distintas constituídas por módulos funcionais:

- Dilib – Representa a biblioteca digital com interface Web. Permite ao usuário a consulta e recuperação de itens. Além disso, oferece aos administradores da biblioteca ferramentas para a execução de seu trabalho.

- MetadataMaker – Aplicação de auxílio ao provedor de metadados para a alimentação do repositório centralizado de metadados.
- RepositoryMaker – Aplicação de auxílio ao provedor de dados para a alimentação dos repositórios de dados.

O projeto do Diliframe, discutido neste capítulo, apresenta uma solução lógica, baseada nos componentes discutidos e no paradigma de orientação a objetos, para cada uma das sub-aplicações. Na fase de projeto, dois importantes artefatos são produzidos: diagramas de interação e diagramas de classe.

Será dada maior ênfase aos diagramas de interação. Segundo Larman, [Larman, 1998] "(...), os diagramas de interação são os mais importantes, do ponto de vista do desenvolvimento de um bom projeto, e requerem um alto grau de esforço criativo. A criação de diagramas de interação necessita da aplicação de princípios de atribuição de responsabilidades e do uso de padrões de projeto".

Além da discussão das decisões de projeto de alto nível definidas para cada sub-aplicação, um estudo mais detalhado de cada módulo funcional é apresentado. Com isso, pretende-se consolidar o projeto do Diliframe e dar liberdade ao desenvolvedor que deseje utilizar o framework, assumindo a responsabilidade de sua implementação. Nesta situação, o Diliframe seria caracterizado como um framework conceitual.

Este capítulo apresentará as sub-aplicações do Diliframe; as decisões estratégicas que afetam cada uma delas; seus módulos funcionais e os elementos que os constituem, acompanhados do modelo de interação que representa o seu processo; e, ainda, alguns diagramas de classe ou exemplos de implementação essenciais para o entendimento do contexto.

4.1.1 A Dilib

A sub-aplicação Dilib oferece a porta de entrada para os serviços da biblioteca digital. A necessidade de se oferecer uma solução mais completa, que contemplasse as três camadas da arquitetura MVC, surgiu de refinamentos do planejamento do Diliframe. O mapeamento das funcionalidades das sub-aplicações em componentes localizados em cada

camada resultou nos módulos funcionais: *View Assembler, Controller, Collection, Repository, User, Login, Searcher* e *Retriever*.

Nesta seção, serão apresentadas três decisões estratégicas que interferem diretamente na implementação da *Dilib*.

- **Repositório centralizado de Metadados**

Conceitualmente, o repositório centralizado de metadados é uma estrutura de armazenamento destinada a receber os metadados referentes aos itens da biblioteca digital. São considerados itens da Dilib os recursos e as coleções.

A principal diferença entre os itens da biblioteca digital é a sua representação física. Os recursos têm representação física. Eles são os textos, imagens ou vídeos, por exemplo, recuperados através da biblioteca. Já as coleções, representam um agrupamento de recursos, ou seja, um relacionamento de objetivo organizacional que não tem representação física. Todavia, ambos os itens são descritos em termos de metadados, de acordo com o mesmo padrão, o Dublin Core. Algumas ressalvas devem ser guardadas com relação aos elementos descritivos *identifier* e *relation*. Na descrição de coleções, o valor do elemento *relation*, que informa a que coleção o item pertence, deve ser nulo e o valor do elemento *idemtifier* não deverá apresentar a informação referente ao identificador do recurso.

Assim, como o repositório centralizado de metadados armazena os registros de metadados de todos os recursos, independente da sua localização física, e de todas as coleções, o processo de pesquisa será facilitado por ser dirigido a apenas uma estrutura de armazenamento.

Além deste, outros benefícios são trazidos com a implementação de um repositório centralizado: melhor desempenho nas consultas, separação das atribuições do provedor de dados do provedor de metadados e maior controle sobre os metadados disponibilizados. Por outro lado, uma maior dificuldade pode ser observada na associação do registro de metadados com o recurso e a recuperação do mesmo. O esquema de identificação, busca e recuperação do recurso deve ser bem definido, a fim de que esta dificuldade não venha a sobrepujar os pontos positivos inviabilizando a estratégia.

- **Arquitetura Local**

A Dilib é o *front-end* do Diliframe. As questões relativas ao bom desempenho e ao adequado funcionamento da aplicação necessitam de uma maior atenção. A tecnologia de *Enterprise JavaBeans* (EJBs), em sua especificação mais recente [Sun Microsystems, 2002] permite a criação de EJBs locais, ou seja, que residam na mesma JVM – *Java Virtual Machine*, onde executam os clientes que a acessam. A opção pela adoção deste esquema visa reduzir o alto custo das chamadas remotas e ainda continuar obtendo serviços oferecidos pelo container EJB.

Dois importantes pontos foram ponderados [Sun Microsystems, 2002c]:

- Aplicações corporativas, cujos clientes têm acesso aos EJBs através de interfaces remotas, resultam em alta escalabilidade e disponibilidade.
- Aplicações corporativas, cujos clientes têm acesso aos EJBs através de interfaces locais, fazem o acesso com um alto desempenho.

O projeto da Dilib adota a opção de arquitetura com acesso a interfaces locais. Dadas às características da aplicação e os objetivos a serem alcançados com o desenvolvimento da mesma, observou-se que os benefícios da arquitetura local são preponderantes.

- **Framework para aplicações Web**

O negócio para o qual o Diliframe está endereçado é a área de bibliotecas digitais, sendo este o foco maior do projeto e o alvo das possíveis contribuições aqui discutidas. Entretanto, uma aplicação não se constitui apenas de componentes de negócio, mas, também, de uma estrutura de acesso e controle capaz de oferecer uma interface adequada para os usuários.

Diante disto, optou-se por utilizar um framework de aplicação para a camada Web a fim de facilitar o desenvolvimento, manutenção e extensão das aplicações criadas para acessar os componentes de negócio da Dilib. Numa estrutura de camadas, um framework para aplicações Web situa-se acima da plataforma J2EE, oferecendo funcionalidades comuns às aplicações, tais como: o direcionamento das requisições, chamada aos métodos dos componentes da camada *model*, seleção e montagem das *views* [Singh, 2002]. A Figura 20 ilustra este cenário.

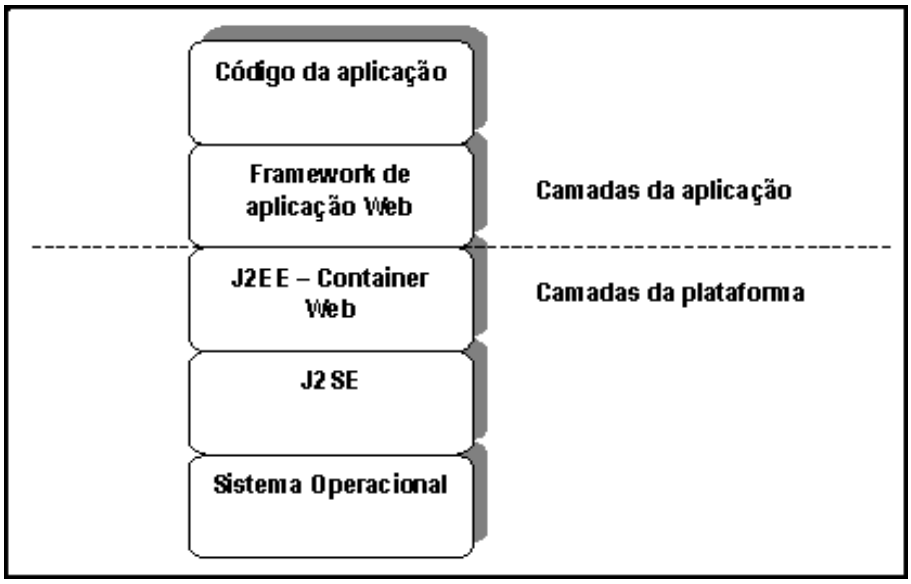


Figura 20 - Aplicação J2EE mais o framework para aplicações Web

Atualmente, já estão disponíveis para os desenvolvedores, alguns frameworks para aplicações Web. Há os que são proprietários e acoplados ao servidor de aplicação. Outros são livres e tendem a se tornar um padrão "*de facto*" no desenvolvimento deste tipo de aplicação. Dentre eles, são citados o JavaServerFaces [Sun Microsystems, 2002d], o Apache Struts [Struts, 2002] e o Web Application Framework – WAF [Sun Microsystems, 2002b] [Singh, 2002].

No geral, as vantagens de se utilizar um framework de aplicações Web estão sumariadas na Tabela 12:

Utilização de framework de aplicação Web
Separação da apresentação e da lógica do negócio em componentes.
Separação das atribuições dos desenvolvedores.
Viabilização de um ponto central de controle.
Maior facilidade no processo de execução de testes e manutenção.
Maior economia, pois pode ser adquirido ou reutilizado ao invés de ser desenvolvido.

Favorece o uso e o desenvolvimento de componentes padronizados.
Favorece a criação de aplicações escaláveis.
Tem suporte de uma comunidade de usuários.

Tabela 12 - Vantagens da utilização de um framework de aplicações Web

No projeto da infraestrutura da Dilib optou-se por utilizar o WAF. Os fatores que pesaram para tal escolha foram, além dos serviços por ele disponibilizados: a sua facilidade de utilização e entendimento, a documentação disponibilizada e o exemplo de utilização completa na aplicação referência J2EE Blueprints [Sun Microsystems, 2002e]. O WAF oferece serviços de filtro e encaminhamento de requisições, geração de *views* baseada em modelos, um conjunto de *tags* personalizadas[6] JSP, controle de fluxo de telas das aplicações, dentre outros [Sun Microsystems, 2002c].

A sub-aplicação Dilib deverá ser construída utilizando os serviços e a tecnologia oferecidos pela plataforma J2EE; compartilhando os serviços e recursos disponibilizados através da adoção do WAF (no que se diz respeito à apresentação e controle da aplicação) e utilizando os componentes de negócio do Diliframe desenvolvidos para esta sub-aplicação. A Figura 21 apresenta a composição arquitetural da Dilib sob este prisma.

[6] As "*custom tags*" são pedaços de código formatados como linguagem de marcação que podem se distribuir ao longo da página JSP. Antes de ser apresentada ao usuário, o servidor substitui as *tags* pelo código fonte da funcionalidade que implementa e processa o conteúdo dinâmico da página.

Figura 21 - Composição arquitetural da Dilib

Os módulos funcionais da Dilib *View Assembler* e *Controller* representam a direta utilização do WAF na Dilib. Nestes módulos, a lógica de funcionamento é definida pelo WAF, bem como as interfaces e classes a serem estendidas na implementação da aplicação.

Uma vez que a arquitetura do Diliframe está dividida em camadas, e a camada *model*, que implementa a lógica de negócio, está resguardada, é possível que o desenvolvedor substitua o WAF por outro que lhe pareça mais adequado. Ou, ainda, que não faça uso de framework para aplicações Web. Entretanto, sugere-se um estudo mais detalhado nas características do WAF e o que ele pode oferecer antes da tomada de decisão. A proposta do WAF é bastante adaptável e de fácil manutenção. A sua associação com o Diliframe resulta em um framework mais completo (não apenas com a camada *model*) e consistente.

A seguir, o projeto de cada um dos módulos será discutido, bem como, alguns detalhes de sua implementação.

4.1.1.1 View Assembler

O módulo funcional *View Assembler* reflete o serviço de padronização de páginas do framework WAF. O seu objetivo é gerar as páginas da aplicação seguindo um mesmo padrão visual. Este padrão visual é obtido através do padrão de projeto *Composite*

View. A página formada será uma associação de *views* dispostas segundo as regras de um documento mestre. A Figura 22 mostra a disposição dos elementos nas páginas da Dilib.

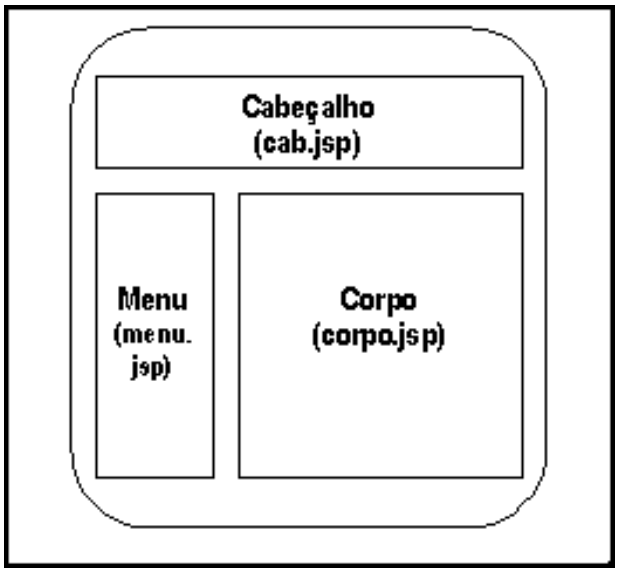


Figura 22 - Página formada a partir da composição de *Views*

O módulo funcional *View Assembler* é assim composto: o servlet ViewAssembler, um arquivo JSP (template.jsp) e um arquivo XML (screendefinitions.xml). Além disso, são utilizadas *tags* personalizadas, notadamente, *tags* de inserção para a composição das *views*. Estas *tags* são oferecidas pelo WAF e constituem-se num importante material de auxílio para o desenvolvimento de páginas JSP.

- O servlet ViewAssembler é responsável por receber a requisição da página, buscá-la e encaminhar a página resposta. Foi convencionado, para o WAF, que as requisições às páginas viriam com a extensão ".screen".
- O template.jsp é o arquivo modelo em cima do qual as *views* serão montadas. Ele define a estrutura de tabelas nas quais se encaixarão as páginas JSP que compõem a *view*. Para apontar que página deve ser inserida no local apropriado, o template.jsp usa a *tag* personalizada *insert*. Esta *tag* é disponibilizada pelo WAF. Sua definição encontra-se no quadro a seguir.

```
<!-- Template insert tag -->
  <tag>
    <name>insert</name>
    <tagclass>com.sun.j2ee.blueprints.waf.view.template.tags.InsertTag</tagclass>
    <bodycontent>JSP</bodycontent>
```

```
    <info>An insertion tag</info>

    <attribute>
      <name>id</name>
      <required>false</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>parameter</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>
```

- O arquivo de definições screendefinitions.xml apresenta as descrições de todas as telas da aplicação. Para cada tela são informados uma série de parâmetros e a página JSP correspondente. Tais parâmetros são especificados nas *tag* de inserção, dentro do template.jsp. A seguir, observa-se um excerto do arquivo screendefinitions.xml:

```
<screen-definitions>
  <default-template>/template.jsp</default-template>
  <screen name="main">
    <parameter key="title" value="Dilib 1" direct="true"/>
    <parameter key="header" value="/ header.jsp" />
    <parameter key="menu" value="/menu.jsp" />
    <parameter key="body" value="/main.jsp" />
   </screen>
   <screen name="login">
    <parameter key="title" value="Log In" direct="true"/>
    <parameter key="header" value="/ header.jsp" />
    <parameter key="menu" value="/menu.jsp" />
    <parameter key="body" value="/login.jsp" />
   </screen>
</screen-definitions>
```

O processo de recuperação e composição das *views* é simples e segue os passos descritos a seguir:

1. A tela de “login.screen” é solicitada ao servlet ViewAssembler;

2. O servlet busca no arquivo “screendefinitions.xml”a descrição da tela e o nome do arquivo modelo que servirá para compor a tela de acordo com o template.jsp;
3. O servlet faz a substituição dos parâmetros indicados das *tags* personalizadas de inserção para os arquivos JSPs correspondentes, através das informações obtidas do “screendefinitions.xml”.
4. O arquivo template.jsp montado para a tela requerida é devolvido;
5. Quando for processado pelo servidor de aplicações Web, o seu conteúdo será traduzido e a tela composta pelas sub-views será apresentada.

O diagrama de seqüência na Figura 23 ilustra este processo utilizando os elementos constituintes do módulo.

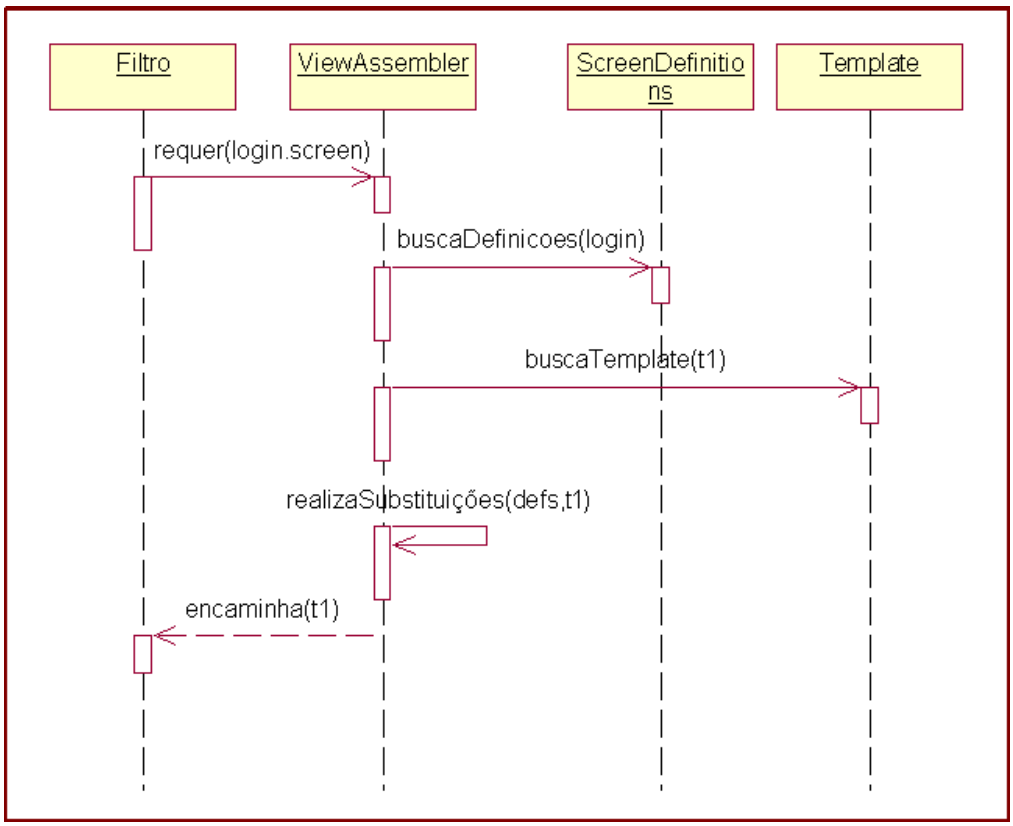


Figura 23 - Seqüência de ações no funcionamento do ViewAssembler

4.1.1.2 Controller

O módulo funcional *Controller* reflete o papel do módulo homônimo no framework WAF [Sun Microsystems, 2002c]. O papel desempenhado por este módulo é chave no funcionamento da aplicação. Destarte, esta sub-seção apresenta o seu modo de trabalho, bem como o seu projeto. No entanto, não se esgotarão aqui os detalhes de implementação do *Controller,* até porque, uma vez que faz parte de um framework, ele foi reutilizado e não produzido. Podem ser obtidas mais informações sobre o WAF e seu *Controller,* em [Singh, 2002] e [Sun Microsystems, 2002c].

A camada Web de uma aplicação J2EE atende requisições do protocolo HTTP. Os seus componentes, os servlets, páginas JSP, arquivos XML e HTML, são organizados em um processo para a recepção e resposta das requisições dos clientes. Este ciclo é constituído de quatro elementos fundamentais, que podem ser estendidos e adaptados, na seguinte ordem [Singh, 2002]:

1) Receber as requisições dos clientes;
2) Encaminhá-las para a *model* executá-las;
3) Selecionar a *view* a ser apresentada;
4) Gerar a *view*.

A Figura 24 apresenta este ciclo simplificado do processo que se desenrola na camada Web.

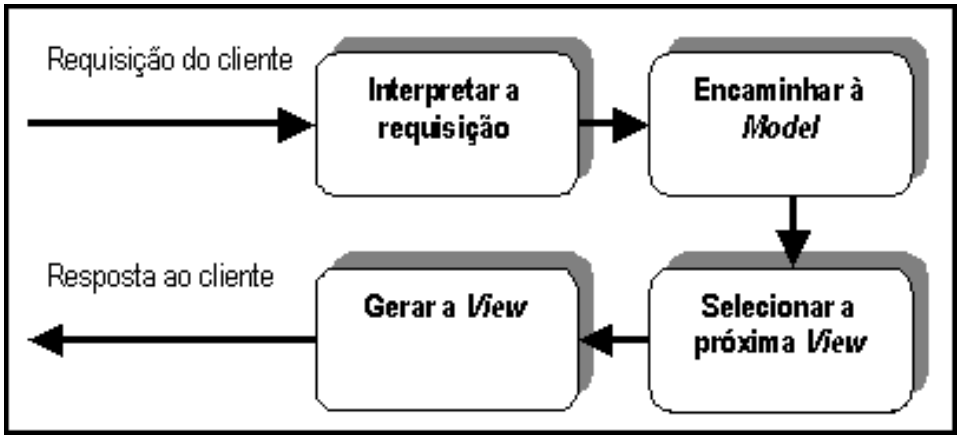


Figura 24 - Ciclo de atividade na camada Web

O WAF segue o processo descrito na figura 25. No entanto, o ciclo de vida de seus serviços pode ultrapassar as barreiras da camada Web atingindo a camada EJB. Quando uma requisição chega ao servlet de controle, é realizado o mapeamento da

operação solicitada para uma ação HTML. O resultado da operação pode implicar no acionamento de uma ação na camada EJB. Isto é indicado através do retorno de um EJBEvent. Em seqüência, um evento EJB é mapeado para uma ação EJB. O WAF executa a ação EJB passando o evento EJB como parâmetro. Finalmente, uma nova *view* é escolhida, com base em um arquivo de configuração XML, gerada e apresentada. A Figura 25 mostra o diagrama de estados do processo.

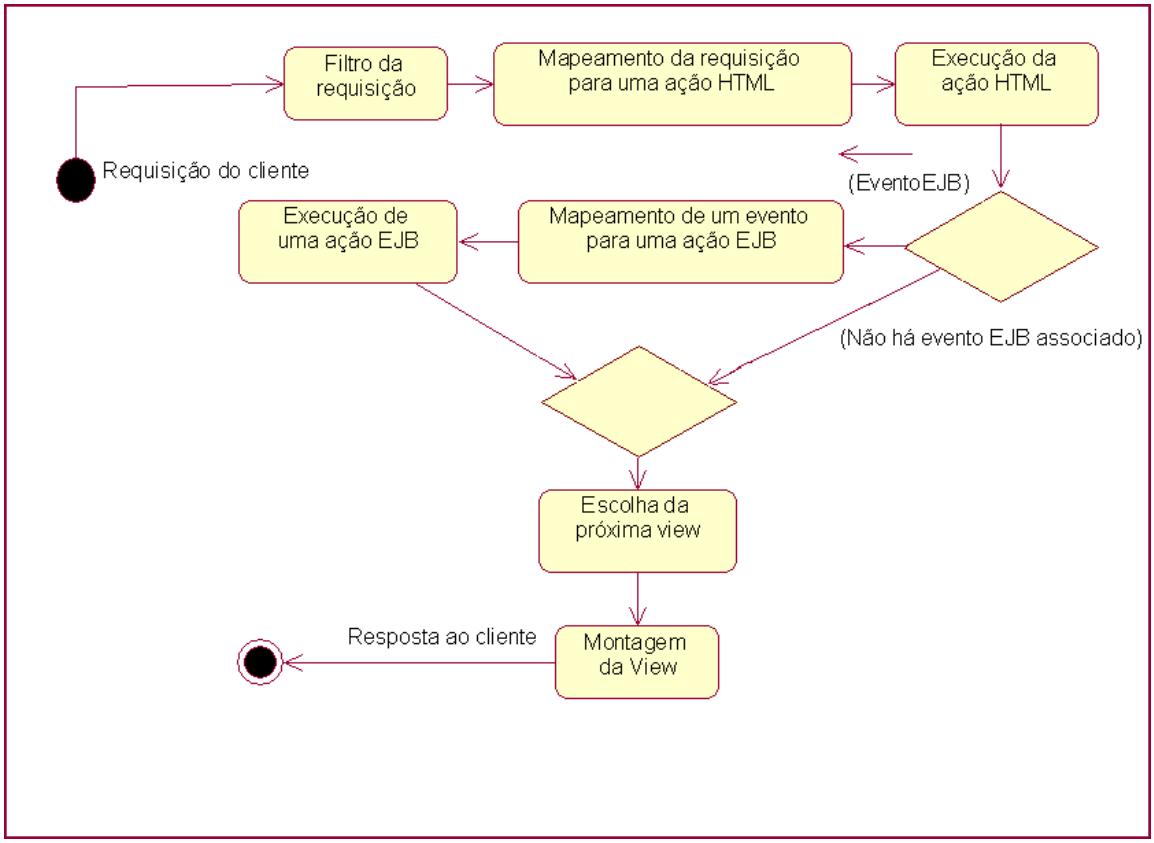


Figura 25 - Diagrama de estados do WAF

Vários elementos são utilizados pelo WAF para a implantação completa deste complexo processo, dentre eles: servlets, arquivos XML, interfaces e classes auxiliares. O desenvolvedor que utiliza o WAF pode estendê-lo, e assim personalizar as suas aplicações, através da implementação de determinadas interfaces, construção de sub-classes, além da adição de elementos nos arquivos de configuração. A seguir, os seis principais elementos que formam o WAF são analisados e seu papel destacado no projeto da Dilib.

- **ServletFilter**. Os filtros de requisição são utilizados para adicionar pré ou pós-processamento às requisições dos usuários. Eles são implementados como servlets. Para

que estes filtros sejam utilizados é preciso que eles sejam registrados no servidor de aplicação no momento da implantação da aplicação. É o *container* Web do servidor de aplicação que faz a gerência e o redirecionamento das requisições para os filtros servlets correspondentes.

Para a Dilib, prevê-se a criação de um filtro de Login. Este filtro é necessário quando as bibliotecas digitais implementadas têm requisitos de segurança que necessitem de autenticação e para permitir o acesso às ferramentas administrativas. Outros possíveis filtros de requisição poderiam ser utilizados para implementar o sistema "pay-per-view", ou seja, o pagamento de taxa mediante a entrega de um recurso. Neste caso a operação de recuperação de recurso seria interceptada e redirecionada para o módulo de cobrança. Após a finalização deste processo, a operação seria retomada.

Os filtros são desenvolvidos como servlets que implementam a interface `javax.servlet.Filter`.

- **FrontController**. Este é o elemento que efetivamente recebe as requisições feitas à aplicação, com exceção daquelas que são direcionadas para os filtros. Ele também controla o fluxo de apresentação das telas e dispara a montagem das mesmas através do módulo *View Assembler*. As requisições recebidas são encaminhadas para o *Request Processor*.

  Na Dilib, a classe que implementa o Front Controller será estendida, ou seja, uma classe filha da pertencente ao WAF será implementada e os métodos adequados serão sobrescritos.

- **RequestProcess**. O processador de requisições é responsável por mapear as requisições do usuário em ações HTML e executá-las. O mapeamento das requisições para ações é feito com base nas informações contidas no arquivo de configuração XML mappings.xml. Neste arquivo encontram-se informações relativas a que classe de ação HTML implementa determinada requisição e que tela deve ser exibida após o processamento da mesma. O código abaixo é um excerto do arquivo mappings.xml.

```
<mappings>
    <url-mapping url="login.do" screen="login.screen" >
```

```
        <web-action-
class>diliframe.dilib.controller.web.actions.LoginHTMLAction</web-action-class>
   </url-mapping>
    <url-mapping url="search.do" screen="search.screen" >
        <web-action-
class>diliframe.dilib.controller.web.actions.SearchfHTMLAction</web-action-class>
   </url-mapping>
</mappings>
```

As classes de ações HTML devem implementar a interface HTMLAction. Esta interface é fornecida pelo framework WAF [Sun Microsystems, 2002c]. Cada operação que o cliente pode requisitar da aplicação dará origem a uma HTMLAction. Elas são o elo de ligação entre a requisição do cliente e a realização da operação de negócio correspondente. Caso esta operação implique em alguma ação na camada EJB, a HTMLAction irá retornar um evento ao *RequestProcessor*. A Figura 26 apresenta um diagrama de classes de uma ação HTML.

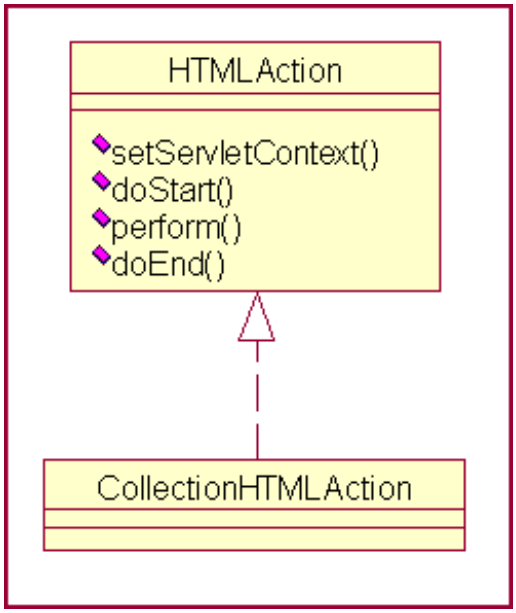


Figura 26 - Diagrama de classe HTMLAction

Na Dilib, devem ser implementadas determinadas ações HTML para as requisições dos clientes. O mapeamento para estas ações deve ser incluído no arquivo mappings.xml.

As ações a serem implementadas e as URLs às quais elas se referem estão listadas na Tabela 13.

Ação HTML	URL
UserHTMLAction	"/user.do"
CollectionHTMLAction	"/collection.do"
RepositoryHTMLAction	"/repository.do"
SearchHTMLAction	"/search.do"
RetrieveHTMLAction	"/retrieve.do"

Tabela 13 - Ações HTML e respectivas URLs

- **WebController**. Ao executar uma ação HTML, o *RequestProcessor* pode receber como retorno um evento, como frisado anteriormente. Os eventos retornados são encaminhados ao *WebController*. As classes que representam eventos devem implementar a interface WAF *Event* [Sun Microsystems, 2002c]. Ela encapsula a requisição HTTP, bem como os seus parâmetros, para que seja possível a execução da ação EJB. O evento é encaminhado ao *EJBController.*
  Na Dilib, deverá ser implementada uma classe com a funcionalidade do *WebController* que implementa a interface WAF *WebController* [Sun Microsystems, 2002c]. A Figura 27 apresenta um diagrama da classe *WebController*.

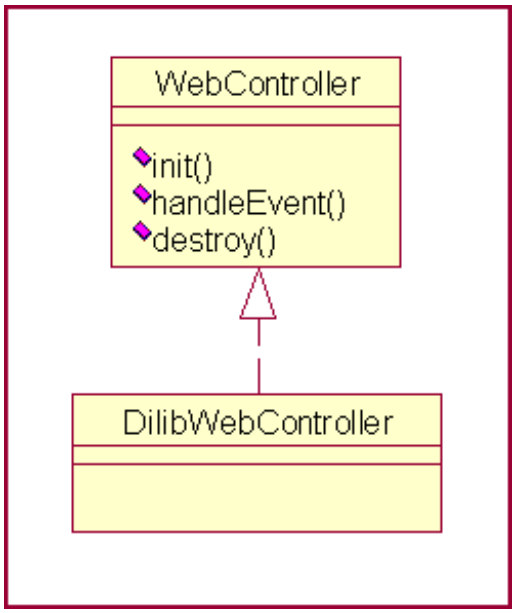


Figura 27 - Diagrama de classe WebController

Além disso, a Dilib deve implementar determinados eventos para viabilizar a realização de suas ações EJB. Estes eventos estão listados na Tabela 14.

Eventos
UserEvent
CollectionEvent
RepositoryEvent
SearchEvent
RetrieveEvent

Tabela 14 - Eventos da Dilib

- **EJBController**. O EJBController é um elemento WAF, mas pertence à camada EJB. Ele deve ser implementado como um "SessionBean". Sua função é receber o evento e, com base na análise de seus dados encapsulados, criar e executar uma ação EJB. As ações EJB são classes que devem implementar a interface WAF *EJBAction* [Sun Microsystems, 2002c]. As ações EJB localizam, criam a instância e utilizam os EJBs disponíveis para executar a operação de negócio. A ação EJB retorna dados para a camada Web através de uma classe do tipo `EventResponse`, que estende a classe

WAF *EventResponse* [Sun Microsystems, 2002c]. A Dilib implementará um EJBController segundo definido pelo WAF. A Tabela 15 mostra as ações EJB que devem ser implementadas e alguns eventos resposta.

Ações EJB	Eventos resposta
UserEJBAction	
CollectionEJBAction	
RepositoryEJBAction	
SearchEJBAction	SearchEventResponse
RetrieveEJBAction	RetrieveEventResponse

Tabela 15 - Ações EJB X Eventos resposta

- **ScreenFlowManager**. O fluxo de apresentação das telas da aplicação é definido pela seqüência de ações, desempenhadas pelo usuário, que é dinâmica. Entretanto, a tela que deve ser apresentada, após a requisição de determinada operação, é definida estaticamente. O ScreenFlowMap é o arquivo de configuração mappings.xml, mencionado anteriormente, que associa as requisições do usuário às ações HTML que serão executadas e a próxima página que deve ser exibida.

O processo completo executado pelo *Controller,* com auxílio do *View Assembler,* na requisição e tratamento de uma requisição do usuário está graficamente representado no diagrama de interação de objetos apresentado na Figura 28. Neste diagrama, o elemento *controller* engloba as funções do *FrontController* e *RequestProcessor*, para fins de simplificação do modelo [Singh, 2002]. Ele tem início com a recepção da requisição do cliente, em seguida é feito o mapeamento da requisição contida na URL para a ação HTML correspondente, encontrada no arquivo mappings.xml. O *RequestProcessor,* então, cria e executa a ação HTML instanciada através da invocação de seu método *Perform*.

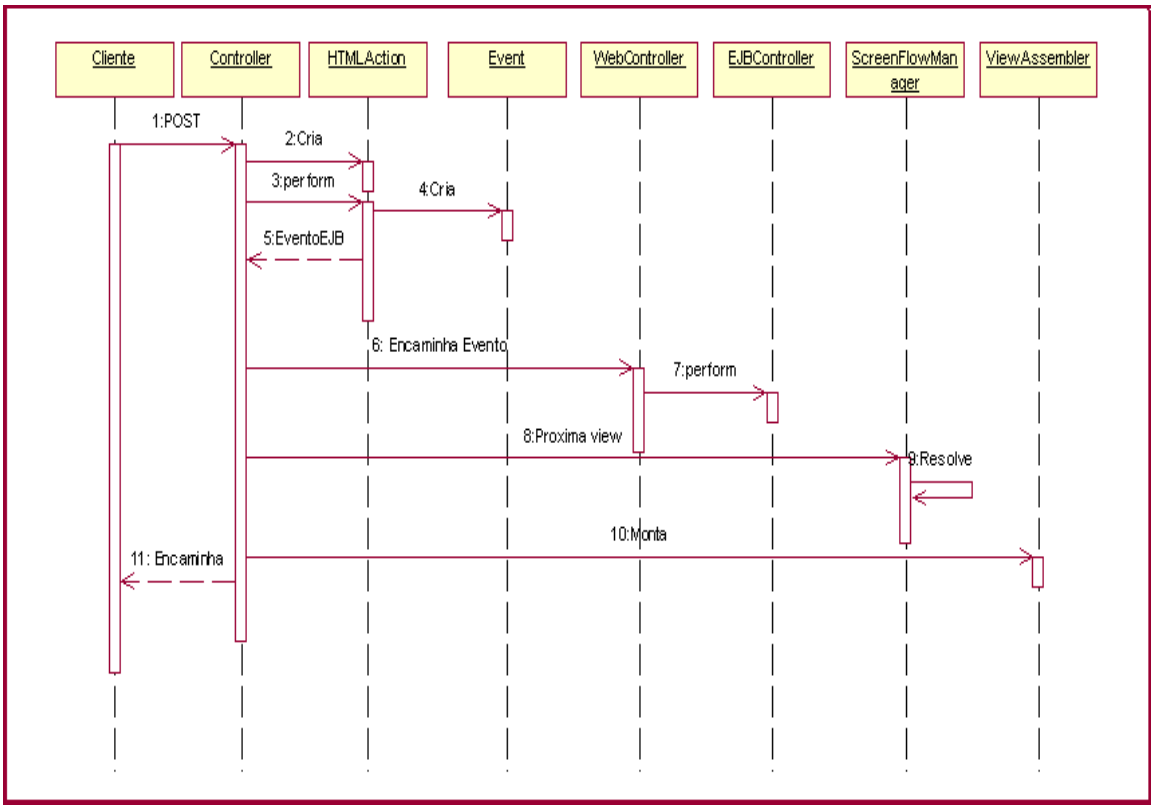


Figura 28 - Diagrama de interação WAF

A Figura 29 representa, graficamente, os elementos do WAF, o relacionamento entre seus módulos e os elementos externos a ele que permitem que o framework seja estendido.

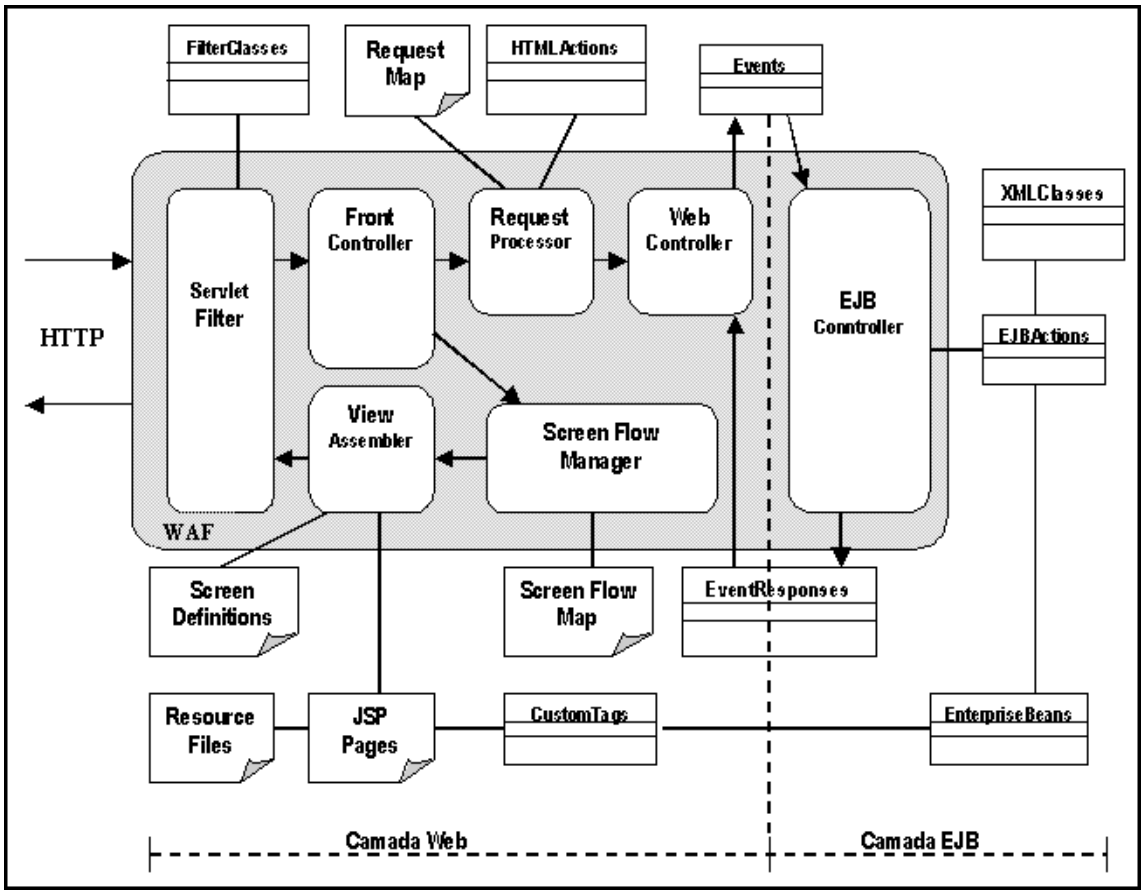


Figura 29 - Representação gráfica do WAF

4.1.1.3 Collection

A partir deste módulo funcional, tem início a apresentação dos elementos de negócio da Dilib. Como visto anteriormente, o módulo funcional *Collection* é utilizado pelo administrador de dados para a consulta, criação, exclusão e atualização de coleções à biblioteca digital. Este módulo também é utilizado para listar as coleções cadastradas. As informações sobre a coleção, descrita por atributos de metadados, são armazenadas no repositório centralizado de metadados.

Optou-se por representar uma coleção por um *Entity Bean* com persistência gerida pelo *container*. A utilização de CMP – *Container Managed Persistence*, facilita sobremaneira a persistência das entidades. A coleção não tem particularidades, como o repositório, que a impeçam de utilizar o modelo CMP. Um dos atributos do EJB *Collection* é o EJB *Metadata*. Ele é utilizado para representar o registro de metadados. A decisão de

criação do EJB *Collection*, a despeito da utilização direta do EJB *Metadata* para representar uma coleção, reflete o interesse de encapsular as regras de negócio desta entidade em seu próprio *bean*.

A sub-aplicação Dilib oferece uma interface para permitir a manipulação das coleções. Uma página contendo os dados da coleção é apresentada, após o preenchimento, dos dados pelo usuário. Algumas preocupações devem ser observadas no momento da criação e alteração dos dados.

Uma coleção deve sempre estar associada a um repositório, que pode ter mais de uma coleção. Dentro do mesmo repositório não deve haver coleções com o mesmo nome. O valor do elemento *identifier*, atributo Dublin Core, deve conter apenas os valores relativos a "<repositório>/<coleção>". O valor do elemento *relation*, atributo Dublin Core, deve conter o nome da coleção. Diante dessas premissas, são traçadas as estratégias de implementação deste módulo funcional.

Deve ser verificada a existência do repositório informado na operação de cadastramento e atualização da coleção. Esta verificação é feita através da interação com o módulo funcional *Repository* da Dilib. Na mesma oportunidade, a existência de uma coleção associada ao repositório, com nome igual àquela que está sendo cadastrada também desqualificaria o cadastro ou alteração. A exclusão de uma coleção implica na remoção de seu nome da lista de coleções do repositório e na exclusão de seu registro de metadados do repositório centralizado. Os registros de metadados dos recursos associados à coleção também são excluídos. Todavia, os recursos armazenados nos repositórios de dados referentes a estes registros não são excluídos. Para a Dilib, o acesso a estes elementos é inviabilizado, assim eles se tornam indisponíveis. A Figura 30 mostra o diagrama de classes simplificado contemplando o *entity bean Collection*.

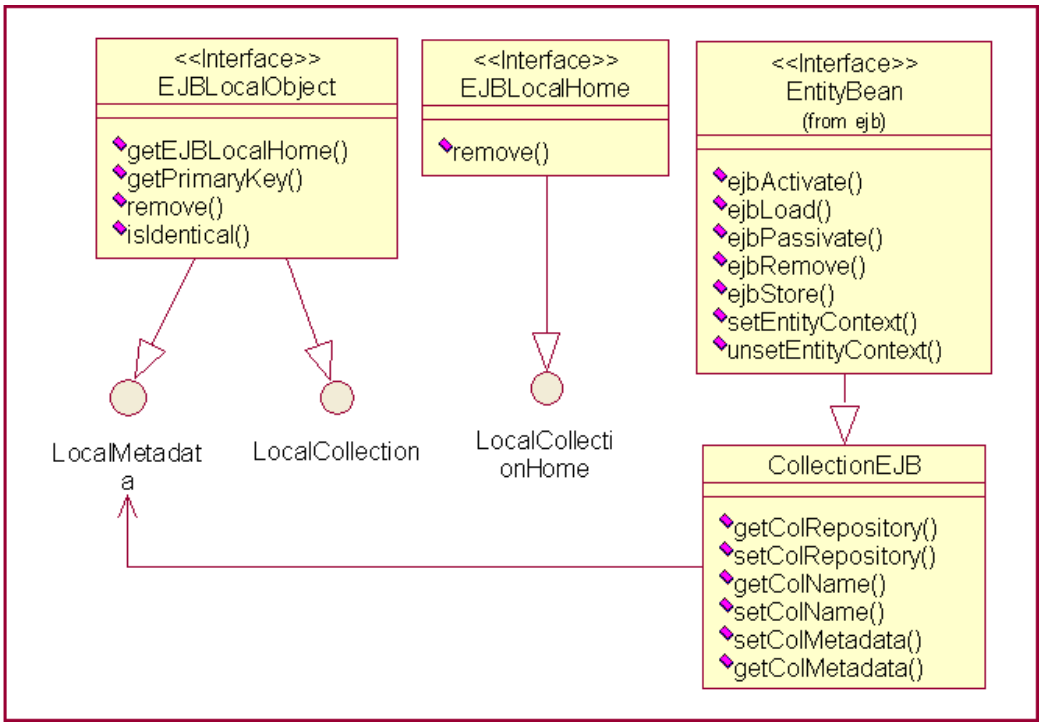


Figura 30 - Diagrama de Classes do entity bean Collection

Assim como as outras funcionalidades da biblioteca, a manipulação da coleção segue o processo de requisição/resposta do WAF explorado na sub-seção 4.1.1.2. O diagrama de interação da, Figura 31, apresenta um esboço da operação de criação de uma coleção.

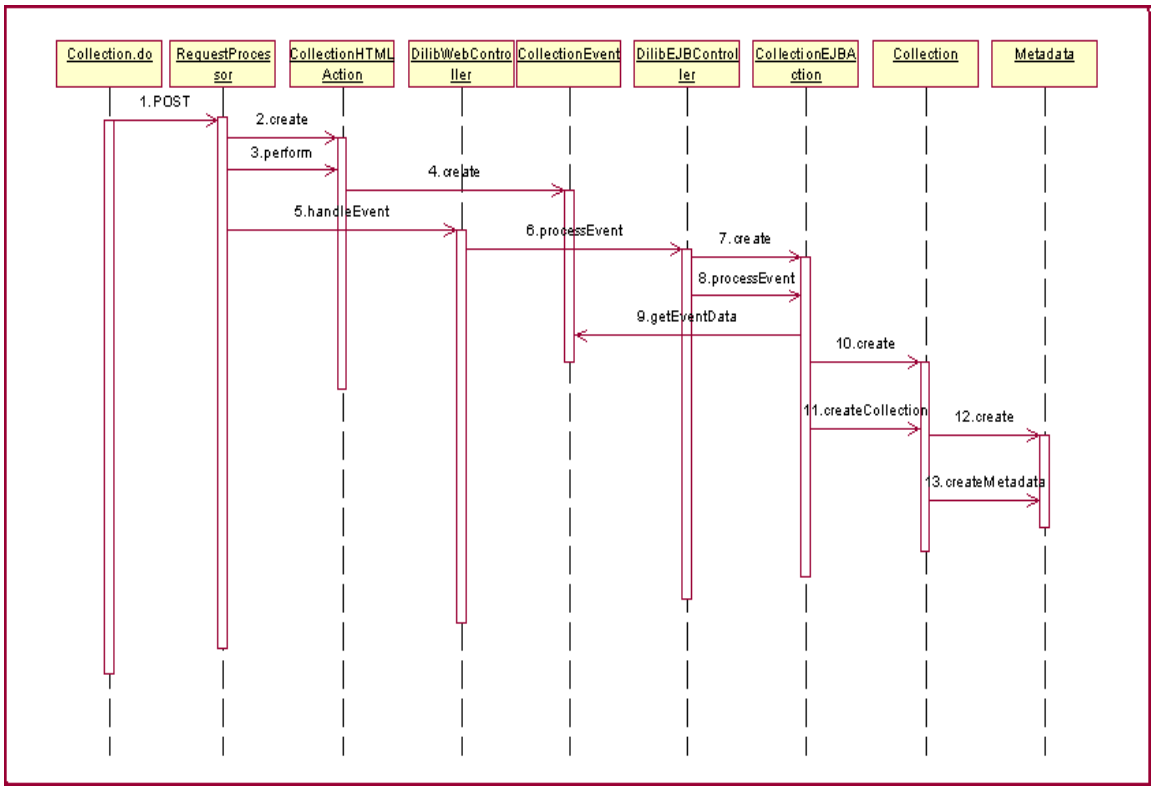


Figura 31 - Diagrama de interação da operação "Create Collection"

O banco de dados, utilizado como repositório centralizado de metadados, deve dispor de uma tabela denominada “metadata”. As colunas desta tabela correspondem aos elementos do padrão Dublin Core. Alguns elementos que admitem mais de um valor por elemento são mapeados como outras tabelas.

4.1.1.4 Repository

O módulo Repository, que faz a gerência dos repositórios disponibilizados para a biblioteca digital, é utilizado tanto pela sub-aplicação Dilib quanto pela *RepositoryMaker*. De forma geral, este módulo contempla o cadastramento de repositórios e indica as configurações necessárias para a utilização dos mesmos no Diliframe.

Os repositórios de dados do Diliframe podem estar em bancos de dados distintos. Para viabilizar a recuperação destes dados, é necessária a criação de um esquema que permita a identificação e o acesso aos mesmos. Além do mais, é necessário oferecer o acesso a estes bancos de forma transparente, a fim de facilitar a implementação da lógica de negócio.

Diante destes requisitos, adotou-se a estratégia de persistência, para os recursos, gerenciada pelo *bean* – e não pelo *container*. O acesso aos recursos será feito utilizando a proposta do padrão de projeto "*Data Access Object*" [Crupi, 2001]. A implementação do DAO, proposta, prevê a criação de uma classe de acesso genérica para qualquer banco de dados relacional. O código SQL, específico para cada banco, para a realização das funcionalidades, está armazenado em um arquivo de configuração XML. Tal classe genérica buscaria neste arquivo o código necessário e assim executaria a operação sql. A Figura 32 apresenta um diagrama de classes em alto nível com o esquema DAO genérico. A utilização do DAO no processo de recuperação de recursos será detalhada posteriormente no módulo *Retriever* (sub-seção 4.1.1.8).

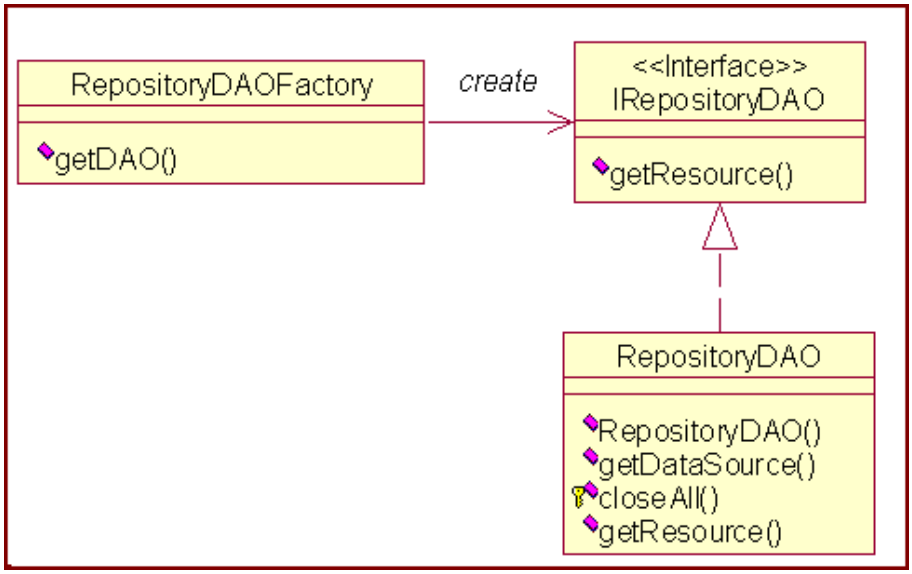


Figura 32 - Diagrama de classes DAO genérico do *Repository*

O arquivo repositoryDAOSQL.xml contém as informações de configuração necessárias para a conexão e consulta aos bancos de dados dos repositórios. O arquivo está dividido em seções, uma para cada banco. Além dos dados necessários para a conexão JDBC, é listado o código SQL necessário para cada método da *Model*. O excerto deste arquivo é listado abaixo.

```
<DAOConfiguration>
          <DAORepository database="oracle" connection_string="jdbc:oracle"
uName="guest" uPass="guest">Repositorio1</DAORepository>
          <DAOCollections>
             <CollectionName>Colecao1</CollectionName>
```

```
            <CollectionName>Colecao2</CollectionName>
         </DAOCollections>
         <DAOStatements>
            <SQLStatement method="GET_RESOURCE">
                  <SQLFragment parameterNb="1">
                        select rec
                        from resource r
                        where a.recid = ?
                  </SQLFragment>
            </SQLStatement>
         </DAOStatements>
</DAOConfiguration>
```

Na Dilib, a interface de acesso ao módulo *Repository* tem como objetivo permitir que os administradores de dados modifiquem as informações sobre os repositórios no arquivo de configuração repositoryDAOSQL.xml. É permitida a criação, exclusão, atualização e consulta de repositórios. É mais uma ferramenta para a edição do arquivo XML existente. Nada impede que os usuários modifiquem o repositoryDAOSQL.xml manualmente, ou através de outra ferramenta.

O acesso ao arquivo XML é feito através de um *stateless session bean* – *Repository*. Este *bean* fará as alterações no XML da forma adequada. Uma classe auxiliar "repositoryData", serializável, também será utilizada dentro deste processo. As atividades deste módulo serão desencadeadas, por uma ação HTML. O diagrama de interação, na Figura 33 apresenta o esboço de uma operação de criação de um repositório.

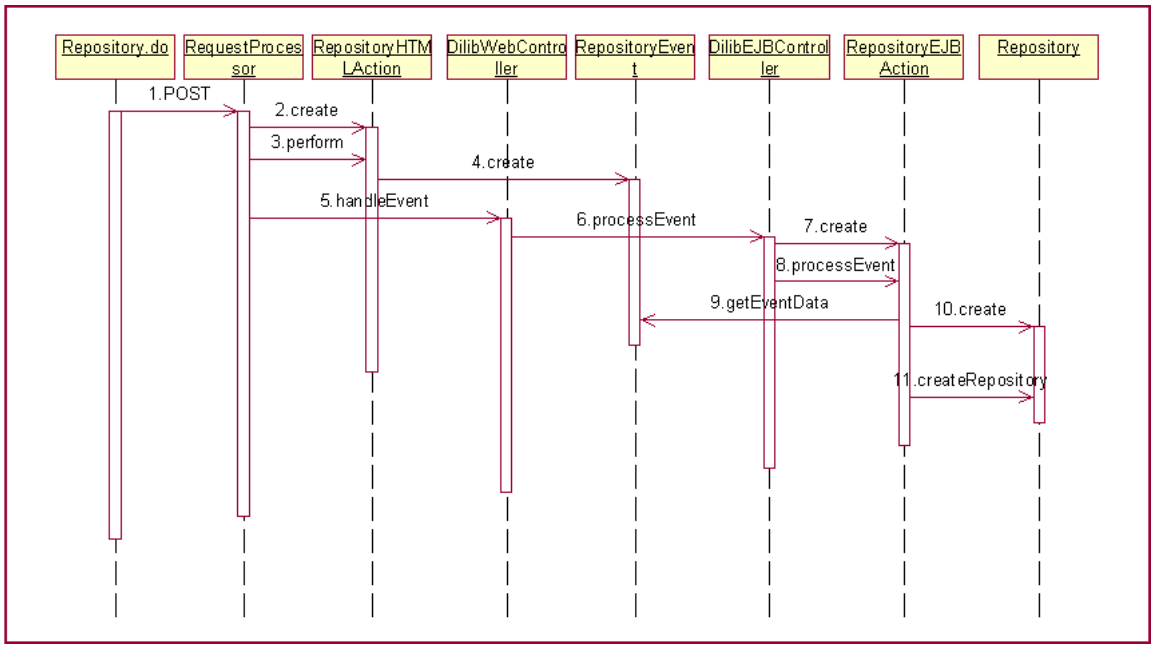


Figura 33 - Diagrama de interação da operação "Create Repository "

Algumas precauções tomadas na criação e alteração dos repositórios visam garantir consistência do sistema. O nome do repositório, indicado pelo valor da *tag* <DAOREPOSITORY> deve ser único. É obrigatório que o desenvolvedor adicione um método de teste de acessibilidade do repositório. Este teste deve envolver as tabelas acessadas, e não deve onerar de maneira marcante o custo da operação. A exclusão de um repositório deve, além de excluir seus dados do arquivo, em cascata, destruir todas as coleções a ele associadas, através de uma interação com o módulo *collection*.

Os bancos de dados utilizados como repositórios da Dilib devem dispor de uma tabela denominada “resource” com os atributos *recid, rec* e *collection*. Por decisão do administrador do banco de dados, uma *view* com esta configuração pode ser disponibilizada, quando as tabelas que contêm os recursos não estiverem estruturadas da maneira requerida. A seguir, é apresentado o script para a criação da tabela no banco de dados Oracle [Oracle, 2002].

```
create table resource (recid varchar(10) not null,
              collection varchar(80) not null,
              rec varchar(255) null,
              constraint pk_resource primary key (recid))
```

4.1.1.5 User

O módulo *User* é responsável por disponibilizar uma representação do ator usuário da biblioteca digital e por viabilizar a sua manipulação. O usuário é uma entidade complexa que agrega, além de seus próprios dados, outras informações pessoais e o seu perfil.

Para atender aos requisitos desta entidade, o *User* foi modelado como um *entity bean*, com persistência gerida pelo *Container* EJB. A necessidade de criação dos EJBs, representando o perfil e as informações pessoais do usuário, respectivamente, *UserProfile* e *UserDetails,* e de sua dissociação do EJB *User,* é resultado da política de maximização da flexibilidade empreendida por este framework. O perfil e as informações pessoais, apesar de dizerem respeito ao usuário, tem natureza de informação diferente. Além disso, a dissociação facilita a substituição do *bean* por outro que se adapte melhor às necessidades do usuário. A Figura 34 mostra o diagrama de componentes que ilustra essa situação.

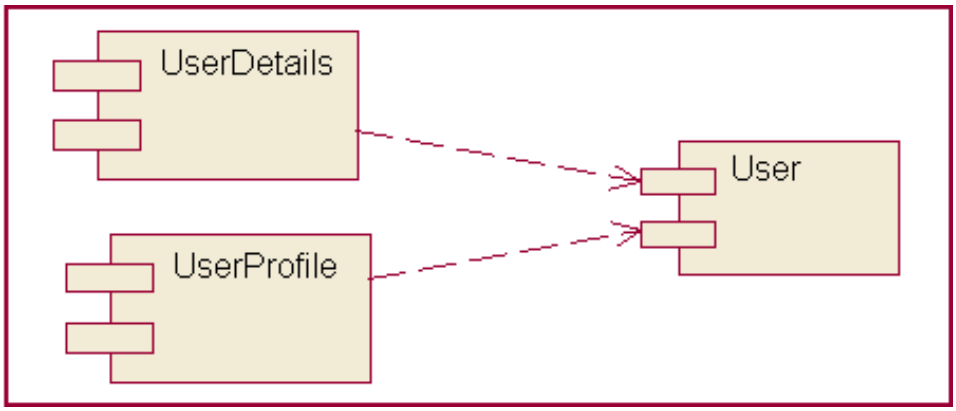


Figura 34 - Diagrama de componentes do *entity bean User*

A Dilib permite que o administrador de usuários crie, exclua e altere os usuários, além de consultar aos seus dados. Estas operações não seriam feitas diretamente, mas sim através de um *session bean* denominado *UserEJBAction*. Através dele, o administrador de usuários também tem acesso aos atributos *UserProfile* e *UserDetails.* Algumas funcionalidades oferecidas pelo *UserEJBAction* estão listadas na Tabela 16.

UserEJBAction
createUser
deleteUser
setUserDetails/getUserDetails
setUserProfile/ getUserProfile
editUserDetail
editUserProfile
getUserByName
getAllUsers

Tabela 16 - Funcionalidades da classe *UserEJBAction*

O *entity bean UserProfile* é um dos atributos do EJB *User*. O *UserProfile* permite que particularidades do usuário no seu relacionamento com a aplicação sejam registradas. A informação fundamental existente no *UserProfile* é o tipo do usuário. A partir desta, serão aplicadas restrições nas *views.* Como a Dilib oferece funcionalidades aos administradores de usuários, administradores de dados e aos usuários comuns, é necessária a restrição de acesso, de acordo com o tipo do usuário. É prevista a implementação de um esquema de cobrança pela informação no Diliframe. Com este fim, foram reservados no *entity bean UserProfile* os atributos colAcessiveis e dadosCobranca e os métodos getColAcessiveis, setColAcessiveis e getDadosCobranca, setDadosCobranca. O perfil do usuário também pode ser utilizado para viabilizar a personalização da aplicação. O atributo "preferências" pode, por exemplo, ser utilizado para guardar o resultado da consulta mais realizada pelo usuário. A substituição deste *entity bean* também é uma alternativa viável para quem deseja implementar cobrança ou personalização de serviços de forma mais refinada. A Figura 35 representa o diagrama de classes simplificado deste EJB.

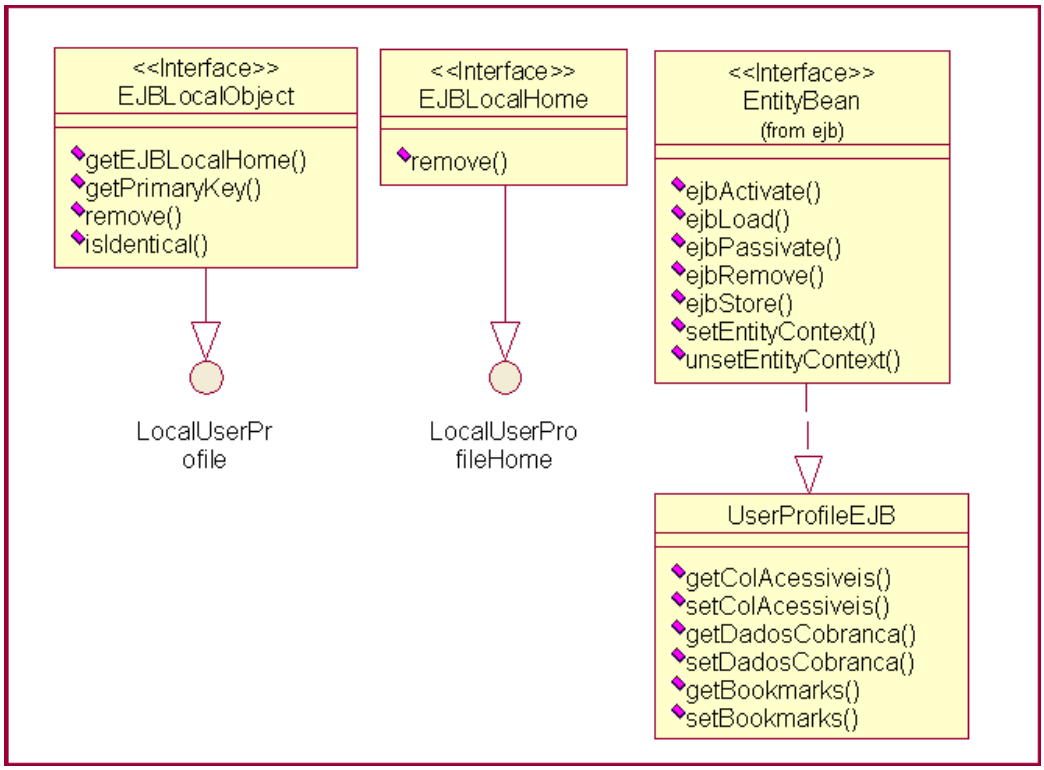


Figura 35 - Diagrama de classes do *entity bean UserProfile*

O *entity bean UserDetails* é utilizado para armazenar a ficha cadastral dos usuários. Nas bibliotecas tradicionais as fichas são de extrema importância. Nas bibliotecas digitais que envolvem cobrança, os dados pessoais do usuário incluindo seu endereço, e os dados de cobrança, também são fundamentais. Entretanto, para bibliotecas digitais que não envolvem sistema de cobrança ou que não necessitam de informações detalhadas do usuário, estes atributos, talvez, não tenham utilidade.

A opção por separar os usuários de suas informações pessoais também se deve ao fato acima mencionado. O *entity bean UserDetails* apresenta uma série de atributos tradicionalmente presentes nos cadastros. Dentre eles, o atributo *Address*, que foi definido como um outro entity bean a fim de que se pudesse reutilizá-lo. Os atributos oferecidos por *UserDetails* talvez não sejam adequados à aplicação que está sendo implementada com o Diliframe. Assim como sugerido para o *UserProfile,* a alternativa é substituir o *UserDetails* caso isto aconteça. A Figura 36 apresenta um diagrama de classes simplificado que ilustra os EJBs *UserDetails.*

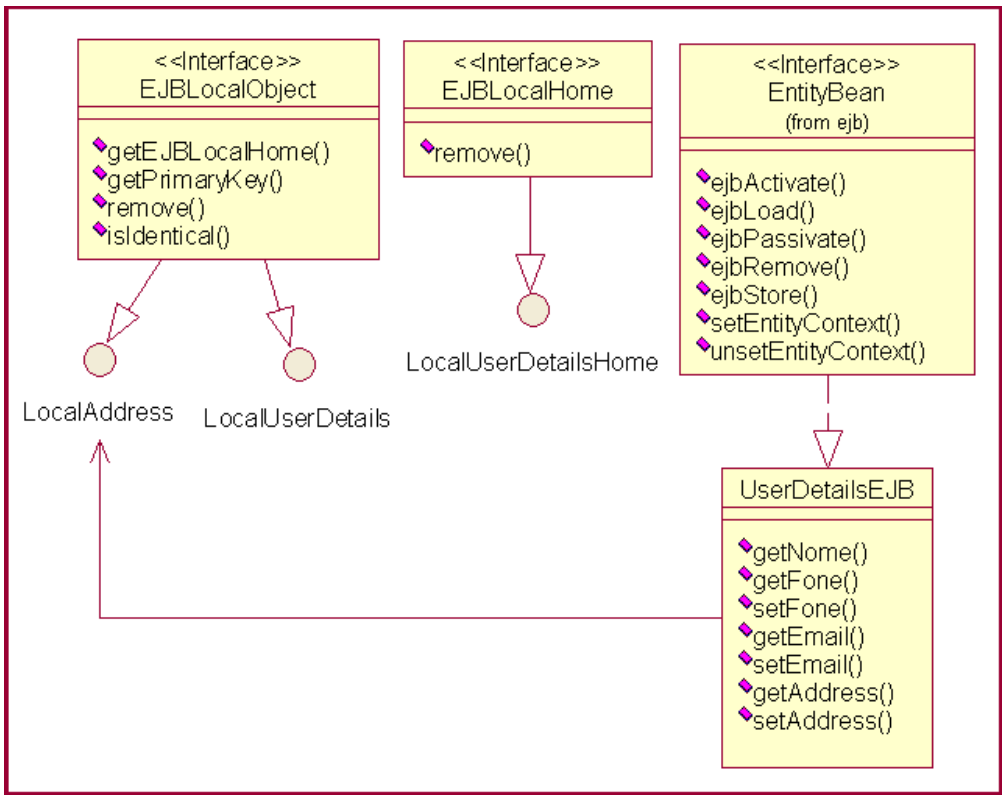


Figura 36 - Diagrama de classes do *entity bean UserDetails*

O *entity bean User* dispõe dos atributos necessários para o login: nome e senha; além do perfil e das informações pessoais. Além de ser utilizado no módulo *Login*, o *User* é o alvo das operações realizadas pelo *UserManager*. O relacionamento com os atributos *UserProfile* e *UserDetail* é de um-para-um. Esta definição é feita no *container*, no momento de implementação da aplicação. A criação de um *entity bean User* dispara a criação de seus dois *entity beans* relacionados. A seguir, um excerto da interface classe *UserEJB* retrata este relacionamento.

```
/*
 *  Implementacao da bean class do ejb "User".
 */
package diliframe.dilib.user.ejb;

import diliframe.dilib.userprofile.ejb.LocalUserProfileHome;
import diliframe.dilib.userprofile.ejb.LocalUserProfile;
import diliframe.dilib.userdetails.ejb.LocalDetailHome;
import diliframe.dilib.userdetails.ejb.LocalDetail;
```

```
import java.util.*;
import javax.ejb.*;
import javax.naming.*;

public abstract class UserEJB implements EntityBean {

    private EntityContext context = null;

    // Metodos de acesso aos campos CMP
    public abstract String getId();
    public abstract void setId(String _id);
    public abstract String getName();
          public abstract void setName(String _name);
          public abstract String getPassword();
    public abstract void setPassword(String _password);

    // Metodos de acesso aos beans relacionados
    public abstract LocalUserProfile getUserProfile();
    public abstract void setUserProfile(LocalUserProfile _lup);

          public abstract LocalUserDetails getUserDetails();
    public abstract void setUserDetails(LocalUserDetails _lud);
 ...
 ...
}
```

Por padrão, na criação de um usuário não há necessidade de informar os dados de *UserProfile* e *UserDetail.* Estes atributos devem ser valorizados posteriormente nas operações de alteração. A exclusão de um usuário ocasiona, por cascata, a exclusão do *UserProfile* e *UserDetail* a ele associados. Esta configuração é feita no servidor de aplicação. A Figura 37 mostra o diagrama de interação que ilustra um esboço do processo de criação do usuário.

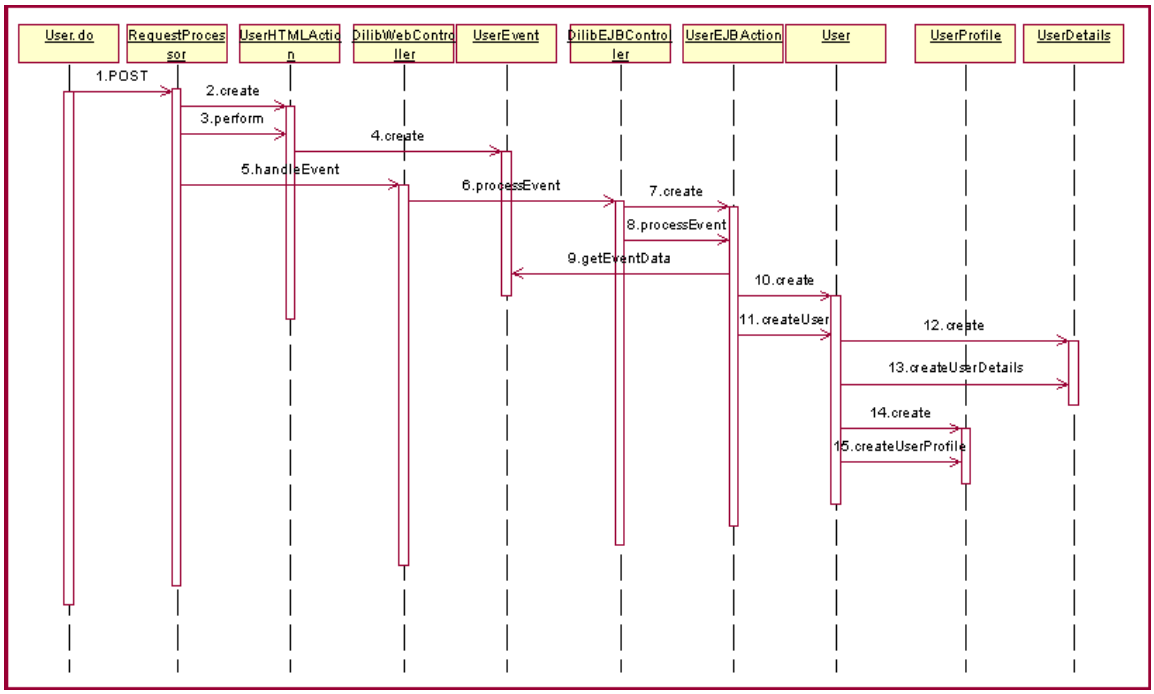


Figura 37 - Diagrama de interação da operação "Create User"

O banco de dados utilizado para a persistência deve dispor das tabelas “User”, “UserDetails”, “Address” e “UserProfile”.

4.1.1.6 Login

O *Login* na Dilib tem como objetivo implementar um sistema de segurança através da restrição de acesso às suas funcionalidades. O acesso é permitido apenas a usuários cadastrados previamente e autenticados mediante “login” e “senha”. Discutiu-se que nem toda aplicação construída utilizando o Diliframe teria necessidade de implementar restrição de acesso às suas funcionalidades. Entretanto, a interface disponibilizada na Dilib para a administração de dados e de usuários deve, obrigatoriamente, ser protegida mediante autorização de acesso.

O projeto e implementação do módulo *Login* procurou suprir as necessidades de flexibilização de sua função de acordo com as características de sua aplicação. Para tanto, optou-se por seguir o padrão “Intercepting Filter” e adotar uma abordagem mista na restrição de acesso, ou seja, apenas determinadas requisições requereriam a autenticação. E, facilmente, alterar esta abordagem para a restrição a qualquer requisição, quando for

necessário. O WAF oferece um filtro deste tipo. O módulo *Login* e seu mecanismo serão implementados utilizando os elementos do WAF.

Todas as requisições feitas à Dilib passarão pelo filtro. Será aplicada a restrição de acesso apenas às operações realizadas pelos administradores de usuários e administradores de coleção. Tais operações são relativas à manipulação de coleções, repositórios e usuários. As páginas de acesso a estas funcionalidades são consideradas protegidas. De acordo com este esquema, não haveria a operação de "*Login"* para usuários simples, apenas para administradores. O diagrama de estados da Figura 38 ilustra o processo [Sun Microsystems, 2002c].

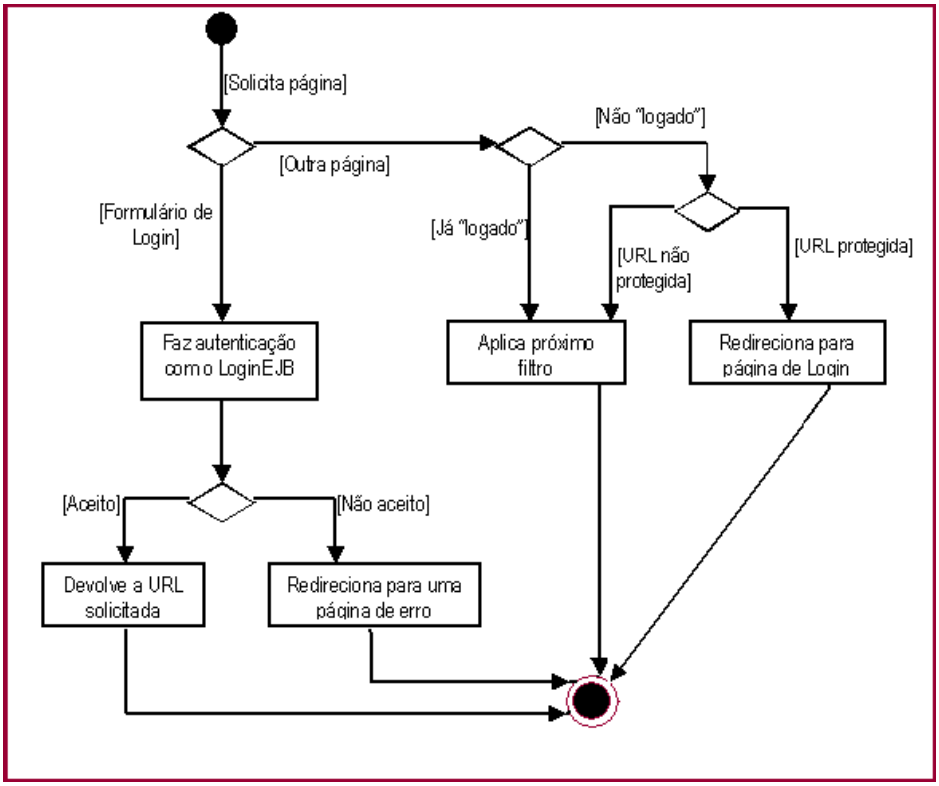


Figura 38 - Diagrama de estados do módulo *Login*

Basicamente, o módulo *Login* seria composto por três elementos: *LoginFilter*, *LoginConfig* e *LoginEJB*.

- *LoginFilter:*Implementa o filtro servlet. Recebe as requisições, verifica se é um recurso protegido e se o usuário que está tentando acessá-lo tem alguma permissão.
- *LoginConfig:* O arquivo de configuração XML loginConfig.xml contém as informações sobre as requisições que as restrições de acesso devem obedecer. O loginConfig.xml é

lido pelo *LoginFilter* que carrega uma lista com as restrições. Um excerto deste arquivo é apresentado a seguir.

```
<login-config>
 <!-- pagina protegida-->
 <security-constraint>
   <web-resource-collection>
    <web-resource-name>Tela de Cadastramento de Colecao</web-resource-name>
    <url-pattern>collection.screen</url-pattern>
   </web-resource-collection>
 </security-constraint>

 <!-- pagina protegida-->
 <security-constraint>
   <web-resource-collection>
    <web-resource-name>Collection Action</web-resource-name>
    <url-pattern>collection.do</url-pattern>
   </web-resource-collection>
 </security-constraint>
...
...
</login-config>
```

- *LoginEJB:* O *LoginEJB* é implementado como um *session bean* responsável por criar a sessão do usuário na Dilib. Ele utiliza o EJB *User*, como citado na sub-seção anterior, para o seu método de autenticação. A Figura 39 apresenta o diagrama de classes simplificado dos elementos participantes deste processo.

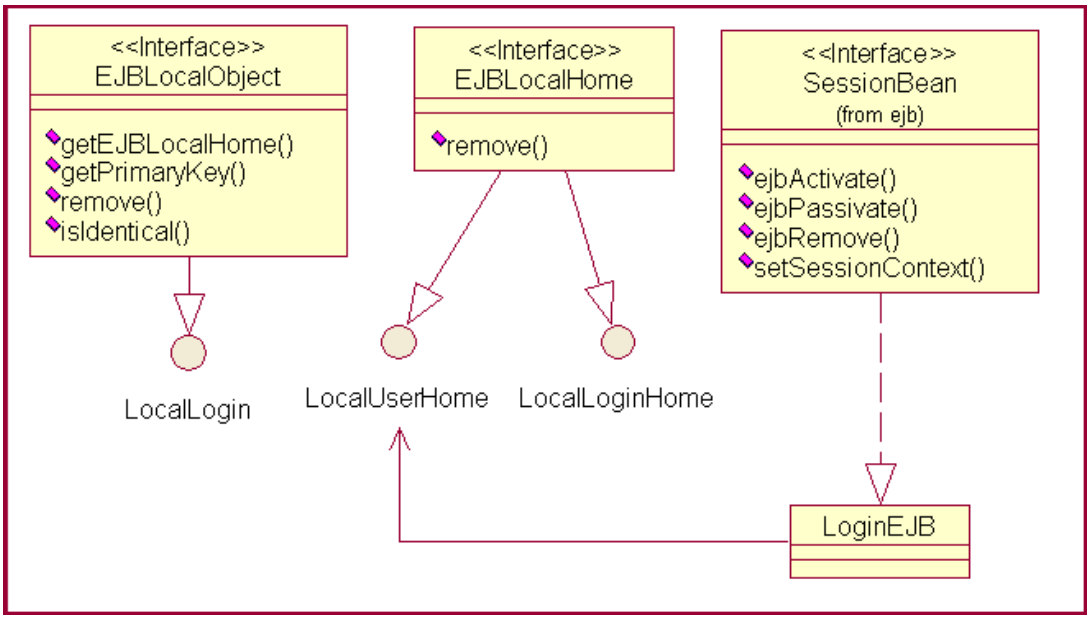


Figura 39 - Diagrama de classes do *session bean Login*

4.1.1.7 Searcher

O módulo funcional *Sarcher* é responsável pelo mecanismo de consulta no repositório centralizado de metadados da biblioteca digital. A requisição do usuário obtida da interface da Dilib será mapeada para uma consulta no banco através do *container* EJB, uma vez que este é responsável pela persistência dos registros de metadados. É da competência do *Searcher* fazer a consolidação e dar suporte à apresentação do resultado da consulta.

A interface com o usuário oferecida pela Dilib para a realização das consultas expõe a lista de coleções acessíveis para o usuário e a lista de atributos de metadados que se pode consultar, além de uma lista de operadores lógicos que permitem a montagem de uma sentença de consulta mais complexa. Este tipo de consulta é denominado no Diliframe de "consulta avançada".

A requisição de uma operação de consulta contém em sua URL, como mencionado na sub-seção 4.1.1.1 a string "search.do". A classe de ação HTML, responsável pela separação dos atributos e a montagem do *SearchEvent*, denomina-se *SearchHtmlAction*. A ação EJB que aciona o módulo *Searcher* para realizar a

funcionalidade requerida pelo usuário é a *SearchEJBAction*, que também é um *session bean*.

O módulo *Searcher* é composto pelo *session bean Searcher* que realiza a consulta e por classes auxiliares utilizadas para viabilizar a paginação do conjunto resultado na sua apresentação para o usuário. Para a realização das pesquisas, o *Searcher* faz acesso ao EJB *Metadata* e conta, ainda, com a classe auxiliar *Metadata Attributes*. O diagrama de classes simplificado, contemplando alguns elementos deste módulo, é apresentado na Figura 40.

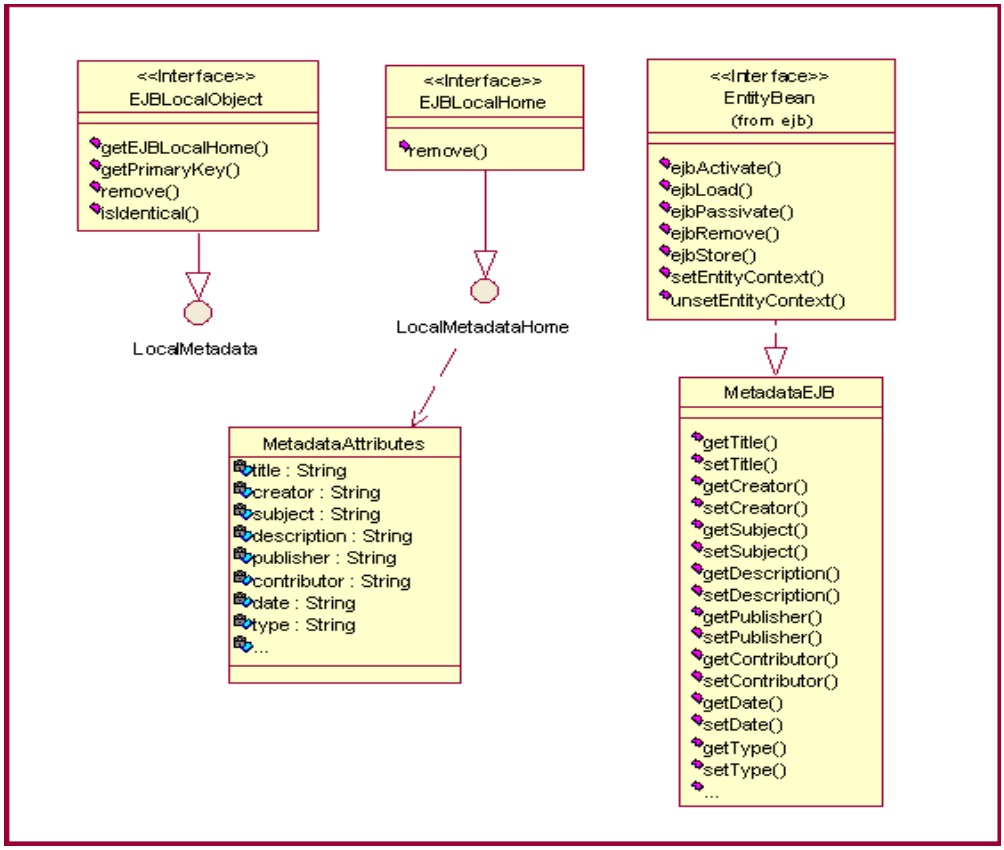


Figura 40 - Diagrama de classes do *entity bean Metadata*

O EJB *Searcher* tem dois métodos de consulta o *simpleSearch* e o *advancedSearch*.

- Através do primeiro item de pesquisa, são realizadas as consultas simples, isto é, que têm apenas um item de pesquisa com restrição a apenas uma coleção. O método tem três parâmetros: o atributo, o valor e a coleção na qual se deseja pesquisar. Através do primeiro parâmetro identifica-se o método correspondente no EJB Metadata para o qual

são passados os dois outros valores. No EJB Metadata, a consulta é realizada pelo container EJB de acordo com a sentença EJB-QL referente ao método "*find*", configurada no *deployment descriptor*. No quadro a seguir, é apresentado um excerto da classe LocalMetadataHome. O resultado desta consulta é devolvido em uma "`java.util.Collection`". O Searcher, tendo em vista a paginação do conjunto resultado na página da aplicação, utiliza as idéias do padrão de projeto "*ValueListHandler*", explicado mais adiante nesta sub-seção.

```
package diliframe.dilib.ejb.Metadata;
import java.util.*;
import javax.ejb.*;
/*
 *  Esta classe contém os métodos de ciclo de vida e finders do EJB "Metadata".
 */
public interface LocalMetadataHome extends EJBLocalHome {

    public LocalMetadata create( MetadataAttributes _ma)
        throws CreateException;
    public LocalMetadata findByPrimaryKey (java.lang.Object _id)
        throws FinderException;
    public Collection findByTitle(String _title)
        throws FinderException;
    public Collection findByCreator(String _creator)
        throws FinderException;
    public Collection findBySubject(String _subject)
        throws FinderException;
     public Collection findByDescription(String _description)
        throws FinderException;
    public Collection findByPublisher(String _publisher)
        throws FinderException;
    public Collection findByContributor(String _contributor)
        throws FinderException;
    public Collection findByDate(String _date)
        throws FinderException;
    public Collection findByType(String _type)
        throws FinderException;
...
}
```

- Através do método *advancedSearch*, pesquisas mais complexas podem ser realizadas sobre os atributos de metadados. As sentenças de pesquisas avançadas podem conter uma lista de diversos pares (atributo, valor), admitindo, inclusive, vários valores para o mesmo atributo. A sentença pode combinar, ainda, a aplicação de restrições em uma ou mais coleções, além da utilização de operadores lógicos: OR, AND, XOR, NOR e NOT. Esta versão do projeto da Dilib deixa em aberto a solução para a implementação desta funcionalidade. Cabe ao desenvolvedor, que deseje contar com a pesquisa avançada implementá-la.

O acesso ao repositório centralizado de metadados, para que sejam realizadas as consultas, é viabilizado pelo *entity bean* CMP Metadata, cujas consultas estão pré-definidas no *deployment descriptor* do *bean*. No caso da consulta avançada, a aplicação exige muita flexibilidade para a formação de suas sentenças. A questão é como contornar, de forma mais prática e eficiente, a dificuldade do mapeamento das consultas dinamicamente montadas pelo usuário, para as estaticamente previstas no *deployment descriptor* do *bean*. Sugere-se a análise de duas soluções para este propósito:

- Uma abordagem baseada nos conceitos do padrão de projeto J2EE "*FastLaneReader*" [Crupi, 2001] que propõe o acesso direto aos dados do banco, via JDBC, quando estes forem tabulares, apenas para leitura, e quando é necessário um melhor desempenho. Os dados do repositório centralizado de metadados atendem às premissas para a utilização do padrão. A idéia central, entretanto, seria a de facilitar a construção das sentenças de pesquisa, quando se estiver tratando de uma consulta avançada. Desta forma, poder-se-ia adotar uma abordagem mista para o processo de consulta: na consulta simples, seriam utilizados os métodos "*find*" geridos pelo container (CMP) para a efetivação da consulta. Na pesquisa avançada, seriam utilizados os métodos de acesso direto ao banco de dados oferecidos por uma classe DAO [Crupi, 2001]. O diagrama de colaboração da Figura 41 mostra um esboço deste processo, contemplando a aplicação do "*Fast Lane Reader*".

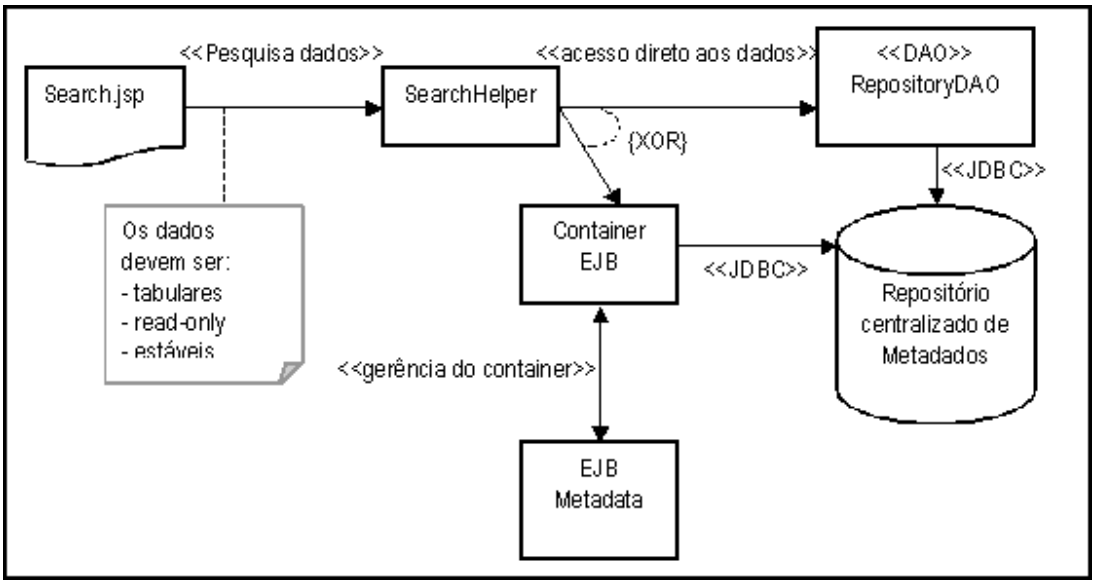


Figura 41 - Diagrama de colaboração *FastLaneReader*

- Uma outra abordagem para aumentar a flexibilidade nas consultas seria a utilização e adaptação dos conceitos do padrão de projeto "*Value List Handler*" [Crupi, 2001]. O texto quadro 123 apresenta uma explicação mais detalhada dos conceitos por ele propostos. A estratégia de implementação seria tornar o *stateful session bean Searcher* em um *Value List Handler.* Ele seria responsável por realizar a consulta e guardar o conjunto resultado como uma variável de estado e, nesta lista, permitir iteração. No *Value List Handler* seriam implementados métodos "*template*" que permitiriam a construção *ad hoc* de sentenças de consulta em tempo de execução. Isto é possível após as aplicações das restrições de consulta que seriam realizadas no próprio *session bean,* após a chamada a um método "*find*" do *entity bean*. "A utilização do *Value List Handler* nestas condições permitiria a implementação de algoritmos inteligentes de consulta e caching, sem as restrições impostas pelos métodos "find" dos entity beans." [Sun Microsystems, 2002].

A manipulação do conjunto resultado na interface com o cliente é realizada através da paginação. Normalmente, a quantidade de registros retornados de uma consulta à biblioteca digital é alta. E para apresentá-los de forma mais eficiente, utiliza-se o conceito de páginas. Por exemplo, na primeira seriam apresentados os dez primeiros registros, na segunda os próximos dez e assim sucessivamente até o final da lista.

O padrão "*Value List Handler*", descrito no quadro 1, apresenta os conceitos que viabilizam a apresentação de grandes conjuntos de resultados de forma eficiente. O módulo *Searcher* da Dilib prevê a utilização deste padrão, e, para tanto, dispõe de classes que implementam seus conceitos. O diagrama de colaboração da Figura 42 representa a interação realizada entre módulos do "*Searcher*" em um processo de consulta simples.

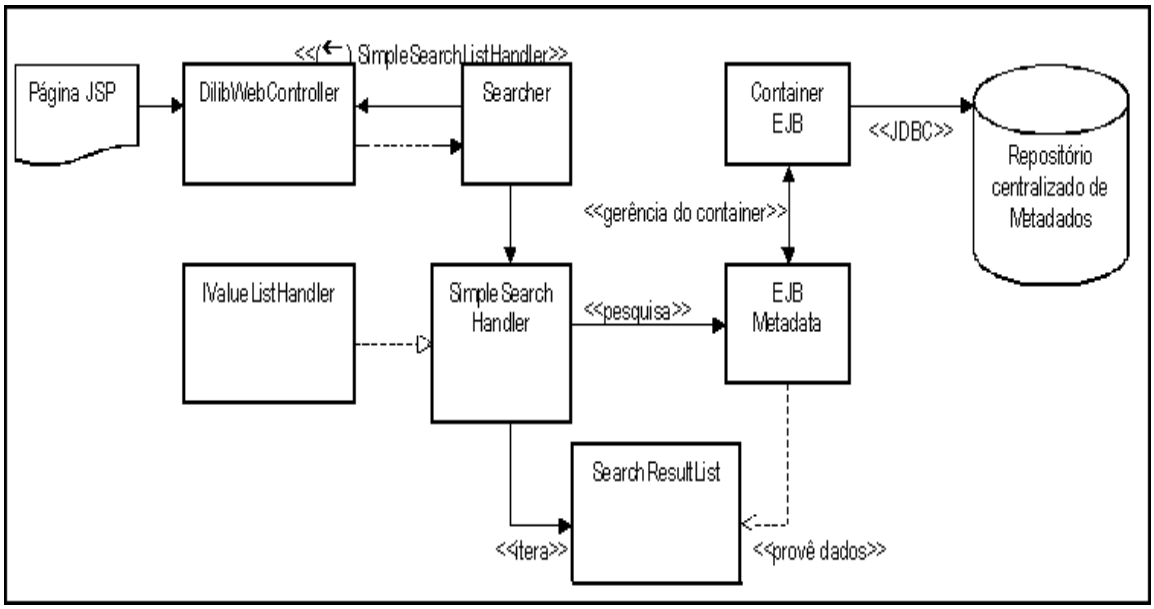


Figura 42 - Diagrama de colaboração *ValueListHandler*

Os passos deste processo são os seguintes:

1. Através da interface adequada o cliente solicita uma consulta na biblioteca digital.
2. O DilibWebController realiza todo o procedimento já conhecido para este módulo até encaminhar a ação EJB de consulta ao *session bean Searcher*.
3. O *Searcher* detecta que a requisição é de uma consulta simples e cria uma instância da classe "*SimpleSearchListHandler*" para executar a consulta.
4. A "*SimpleSearchListHandler*" cria e utiliza o EJB *Metadata* para realizar a consulta requerida.
5. O container EJB, responsável pela persistência no *entity bean Metadata,* faz acesso ao banco de dados e implementa a consulta requerida.
6. Os elementos resultantes da consulta, denominados "*ValueObject*" na linguagem do padrão "*Value List Handler*", irão constituir a "*Value List"* (*SearchResultList*) que está contida na classe "*SimpleSearchListHandler*".

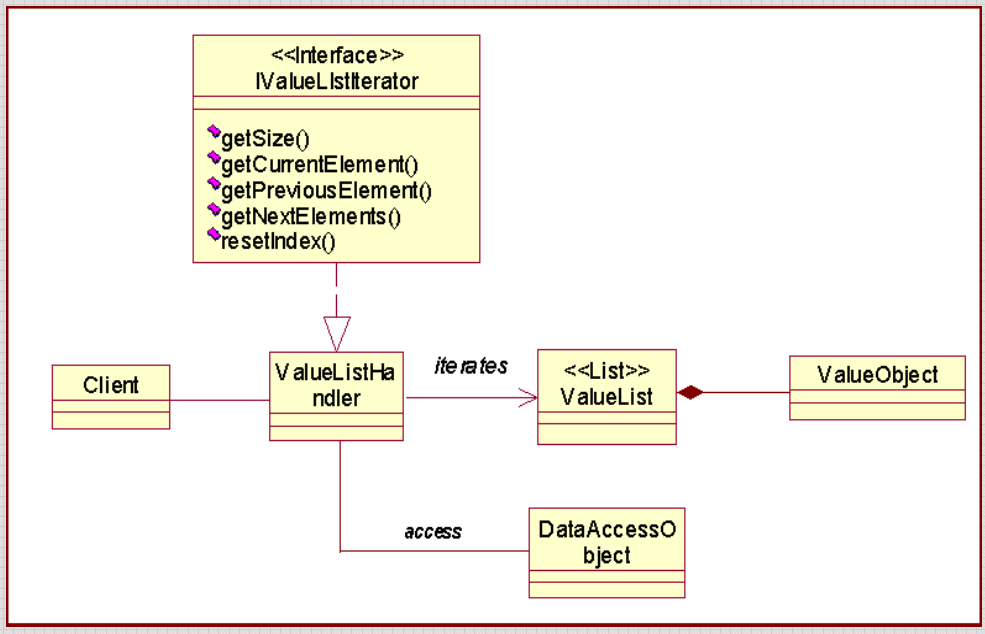
7. A "*SimpleSearchListHandler*", que implementa a interface "*ValueListIterator*", permite que o cliente navegue sobre o conjunto resultante da consulta.
8. É através do "*SearchEventResponse"*, que o módulo *Searcher* envia a instância da "*SimpleSearchListHandler*" à camada Web.
9. Para a montagem e apresentação do resultado, a página JSP deve fazer acesso à "*SimpleSearchListHandler*".

Value List Handler [Crupi, 2001]

O *Value List Handler* é utilizado para controlar a consulta, a cache de resultados e para fornecer os resultados para o cliente em um conjunto resposta cujo tamanho não é previamente conhecido, e tende a ser bastante grande.

Este padrão prevê a criação de uma classe *Value List Handler* para controlar a execução das consultas e o caching de resultados. A *Value List Handler* armazena os resultados obtidos da consulta como uma coleção de Transfer Objects, ou seja, as sub-listas. O cliente referencia a classe *Value List Handler* para oferecer os resultados da consulta sempre que necessário. Esta classe implementa o padrão Iterator [Gamma *et al*, 1994] para viabilizar a solução.

A figura a seguir, representa o diagrama de classes dos elementos envolvidos neste padrão.



4.1.1.8 Retriever

O módulo funcional *Retriever* é responsável pelo processo de recuperação do recurso selecionado. Como visto anteriormente, os recursos da biblioteca digital estão dispersos em repositórios distintos. A consulta, que é feita no repositório centralizado de metadados, retorna registros de metadados contendo informações que permitem a recuperação do recurso. O mapeamento realizado para a identificação do recurso e localização do repositório são feitos com base nestas informações.

O processo de recuperação, empreendido pelo *Retriever*, abrange as atividades de identificação do repositório, geração da sentença de consulta, realização da consulta, preparação e entrega do resultado. Para tanto, são utilizados elementos com atividades bem definidas como: o *session bean retriever*, o arquivo de configuração XML repositoryDAOSQL.xml, as classes utilizadas para implementar o DAO, classes auxiliares para a leitura e *parsing* do XML, além de outras.

O ponto de partida para a recuperação do recurso é na interface da Dilb. Na URL que contém a solicitação de recuperação do usuário está contida a string "repository.do". O *DilibWebController*, quando recebe a solicitação, cria uma ação HTML denominada *RepositoryHTMLAction*. Esta ação envia ao *DilibWebController* o evento *RepositoryEvent* contendo os atributos passados pela camada Web para a sua realização. A ação EJB *RepositoryActionEJB* é criada e, a partir deste ponto, o *session bean retriever* é acionado. O diagrama de classes da Figura 43 representa o *session bean retriever*.

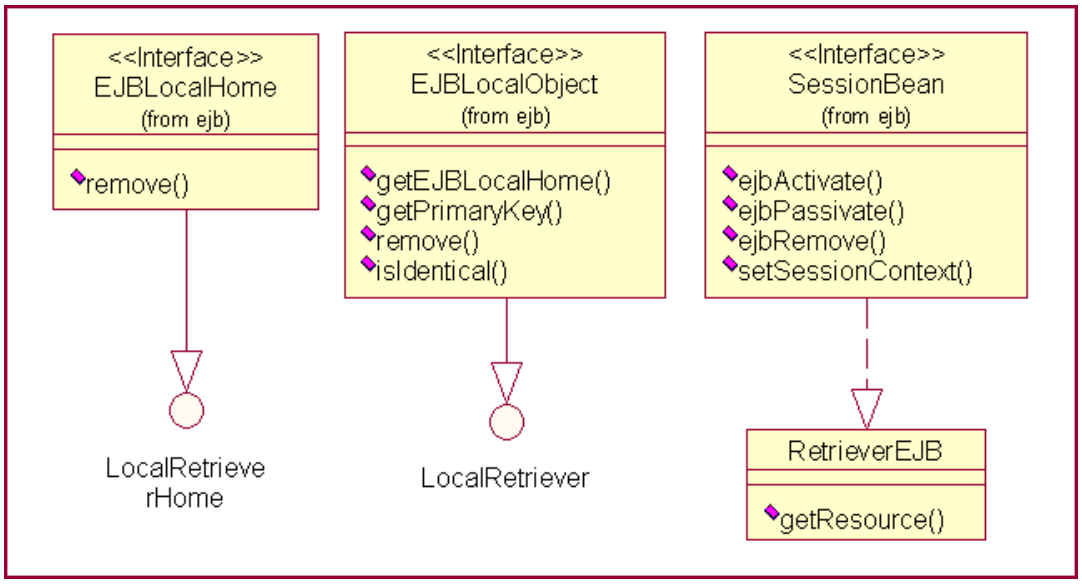


Figura 43 - Diagrama de classes do *session bean Resource*

O primeiro passo para a recuperação é obter a identificação do recurso e do repositório, a partir do *Identifier* do registro de metadados transmitido. O *Identifier* é um elemento composto formado por três partes: o nome do repositório, o nome da coleção e o identificador do recurso em sua tabela. Os nomes do repositório e da coleção devem corresponder às informações cadastradas através do módulo *Repository* no arquivo XML repositoryDAOSQL.xml.

O *session bean Retriever* irá criar uma instância da classe RepositoryDAO e a utilizará para invocar o método de consulta disponibilizado, passando como parâmetro as informações obtidas através do *id* (Veja o excerto do código do *Session Bean* no quadro a seguir). A RepositoryDAO irá ler o arquivo repositoryDAOSQL.xml e obter a sentença de consulta adequada para o banco de dados informado. O resultado da consulta realizada será encapsulada na classe *Resource*. O *Retriever* encaminha o *Resource* criado ao *DilibWebController,* através do evento de resposta *RepositoryEventResponse*.

```
/**
 * Session Bean implementation of Retriever
 */
 public class RetriverEJB implements SessionBean {


    protected RepositoryDAO dao;
```

```
    public void ejbCreate() {
        try {
            dao = RepositoryDAOFactory.getDAO();
        }
        catch (CatalogDAOSysException se) {
            Debug.println("Excecao ao recuperar a DAO " + se);
            throw new EJBException(se.getMessage());
        }
    }
    public void setSessionContext(SessionContext sc) {}
    public void ejbRemove() {}
    public void ejbActivate() { }
    public void ejbPassivate() { }
    public void destroy() { dao = null; }

    public Resource getResource(ResourceID _resourceID ) {
        try {
            return dao.getResource(_resourceID);
        }
        catch (CatalogDAOSysException se) {
            throw new EJBException(se.getMessage());
        }
    }
...
}
```

A Figura 44 apresenta o diagrama de interação que ilustra o processo de recuperação do recurso, a partir do EJB *Retriever*.

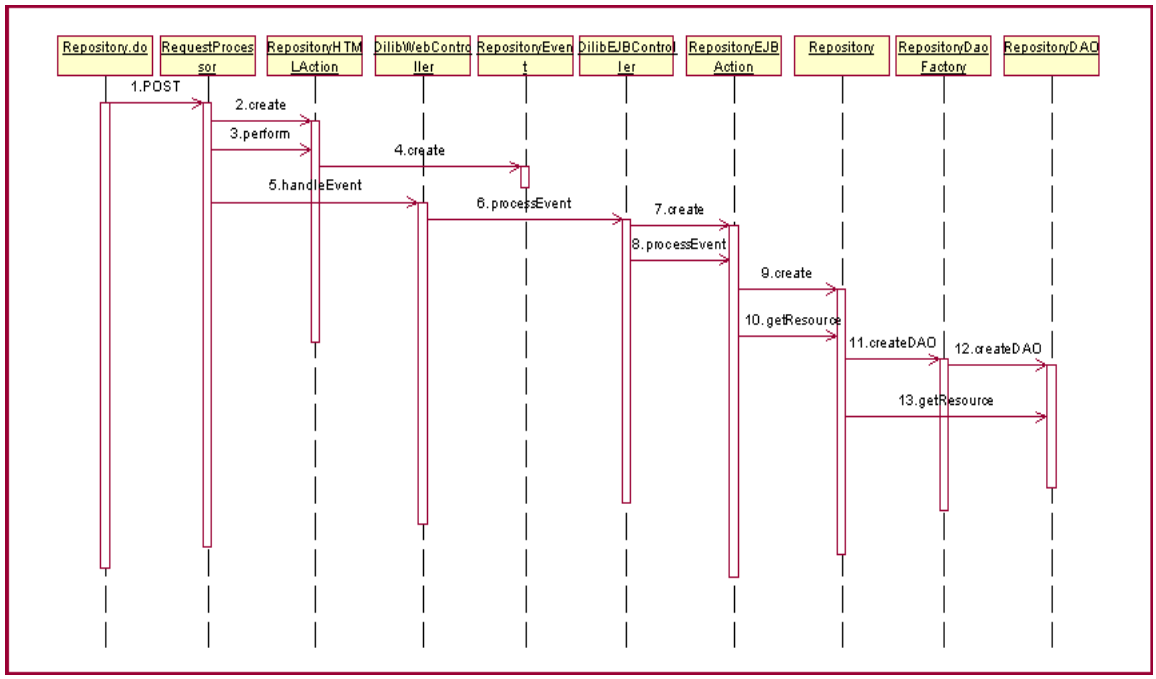


Figura 44 - Diagrama de interação da operação "*getResource*"

A página JSP, que irá apresentar o resultado da consulta ao usuário, terá acesso à classe *Resource* que conterá o conteúdo do recurso recuperado do banco de dados. Neste ponto, o formato do recurso é bastante importante. Se o recurso for do formato texto simples ou html, será carregado em um *buffer* da classe *Resource* que, simplesmente, incorporará este recurso às suas páginas de apresentação.

O projeto deste framework deixa em aberto a solução para o tratamento de recursos em formatos mais complexos. Para tanto, sugerem-se algumas abordagens:

- Determinados formatos de recurso podem ser convertidos para html, por exemplo: .doc, .pdf, .xls, dentre outros. Para tanto, existem meios disponibilizados na Web sob a forma de *plugins*. Estes *plugins* seriam utilizados pelo *session bean Searcher* para a conversão e o carregamento do *buffer* da classe *Resource*.
- Em alguns casos, o usuário necessita do recurso em seu formato original, ou talvez o recurso não seja conversível para html. Neste caso, sugere-se que o recurso seja devolvido ao cliente como em um processo de *download*. Para tanto, o *servlet* responsável por devolver o recurso deve ter o seu *content-type* ajustado para o tipo de dado a ser devolvido.

- Uma outra possibilidade seria o armazenamento apenas de recursos baseados em XML. Desta forma, poderiam ser recuperados dados das mais diversas apresentações: voz, imagens, mapas de forma comum e com um tráfego notavelmente leve. Caberia ao cliente (browsers) a interpretação do recurso recuperado. Maiores informações sobre a utilização de XML para a representação de dados multimídia podem ser encontradas em [W3C, 2002] [W3C, 2001] [W3C, 2001b].

4.1.2 A MetadataMaker

A sub-aplicação *MetadataMaker* é utilizada pelo provedor de metadados para a inclusão e manipulação de registros de metadados no repositório centralizado. Para se ter acesso às suas funcionalidades, o usuário deve ter permissão concedida pelo administrador de usuários em seu perfil.

A *MetadataMaker* deve garantir a consistência das informações que são inseridas ou alteradas nos registros de metadados; para que, por exemplo, não haja um registro de metadados referenciando um recurso inexistente. Ou, também, para que um recurso não seja atribuído a mais de uma coleção, ou a uma coleção não cadastrada em seu repositório. A *MetadataMaker* encaminhará os registros para inserção através de arquivos XML, permitindo assim a utilização de registros de metadados já prontos.

Para a efetivação das funcionalidades oferecidas pela *MetadataMaker* é necessária sua integração com alguns módulos funcionais da Dilib: *Login*, *Collection*, *Retriever* e *Repository*. Diferentemente da Dilib, que é uma aplicação baseada em Web, a arquitetura da *MetadataMaker* prevê que ela seja uma aplicação cliente J2EE. Não obstante, a interação entre os componentes da Dilib seja feita através de interfaces locais, a *MetadataMaker* irá fazê-lo através de interface remota. Isto permitirá que o provedor de metadados possa atualizar o repositório centralizado à distância.

A arquitetura da sub-aplicação *MetadataMaker* reflete as preocupações com a gerência e o acesso aos recursos da Dilib. Seus módulos funcionais estão estruturados de acordo com o padrão MVC e desempenham papel bem definido para a viabilização dos serviços oferecidos pela aplicação. A seguir, são listadas a descrição da estrutura de cada módulo e suas responsabilidades.

4.1.2.1 Cliente

Este módulo funcional é implementado como uma classe tradicional Java. Ele expõe ao usuário a interface adequada para a realização das operações de *Login*, de criação, inserção, atualização e exclusão de registros de metadados. Para a realização destas operações, o Cliente cria uma instância do EJB *MetaMaker* que representa o *Front Controller* da aplicação. O *MetaMaker* disponibiliza o acesso às suas funcionalidades através de uma interface local.

- A interface oferecida para a operação de *Login* tem a forma usual de dois campos: uma para o nome e o outro para a senha. A solicitação de *Login* é encaminhada ao *Front Controller MetaMaker* contendo estes dois argumentos. A Figura 45 apresenta um diagrama de interação que esboça, em alto nível, a operação de *Login* na sub-aplicação *MetadataMaker*.

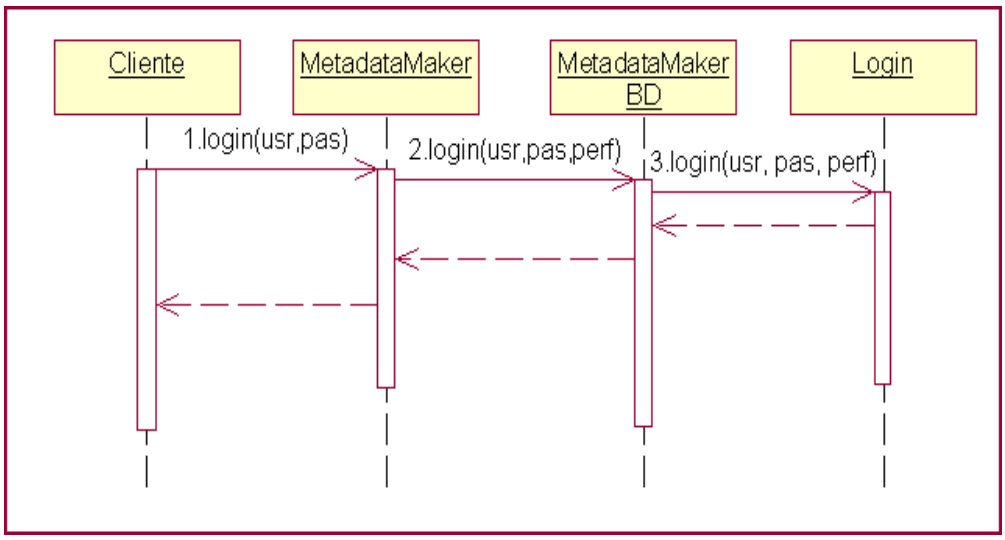


Figura 45 - Diagrama de interação da operação "*Login*"

- A interface apresentada para o processo de **Criação** consiste de um formulário contendo campos referentes aos elementos do padrão de metadados *Dublin Core*. Os elementos de preenchimento obrigatório deverão estar destacados como tal. Os atributos *subject* e *format*, cujos valores estão restritos a vocabulários controlados, são seguidos por uma lista contendo as opções possíveis. O elemento *identifier* já deverá estar semi-preenchido de acordo com a prévia seleção do usuário de repositório e coleção, faltando apenas o identificador único do recurso (chave primária). Por

conseguinte, o campo *relation* deverá, também, estar preenchido. Através de classes auxiliares o formulário contendo os valores do registro de metadados é carregado em um arquivo XML para que, em cima deste, seja realizada a operação de inserção.

- A operação de **Inserção** disparada pelo Cliente pode ser precedida, ou não, por uma operação de criação de registro de metadados. O usuário pode optar por utilizar arquivos XML para a descrição de metadados no padrão Dublin Core já prontos, para alimentar o repositório centralizado. A interface oferecida pelo módulo funcional Cliente deve possibilitar a seleção de um arquivo em uma árvore de diretórios ou até mesmo um diretório completo. Neste último caso, seriam feitas tantas operações de inserção quantos forem os arquivos XML do diretório informado. A solicitação de inserção será encaminhada ao *Front Controller MetaMaker* junto com uma referência ao arquivo de registro de metadados em questão. A Figura 46 apresenta um diagrama de interação que esboça, em alto nível, o processo de inserção de registro de metadados no repositório centralizado de metadados.

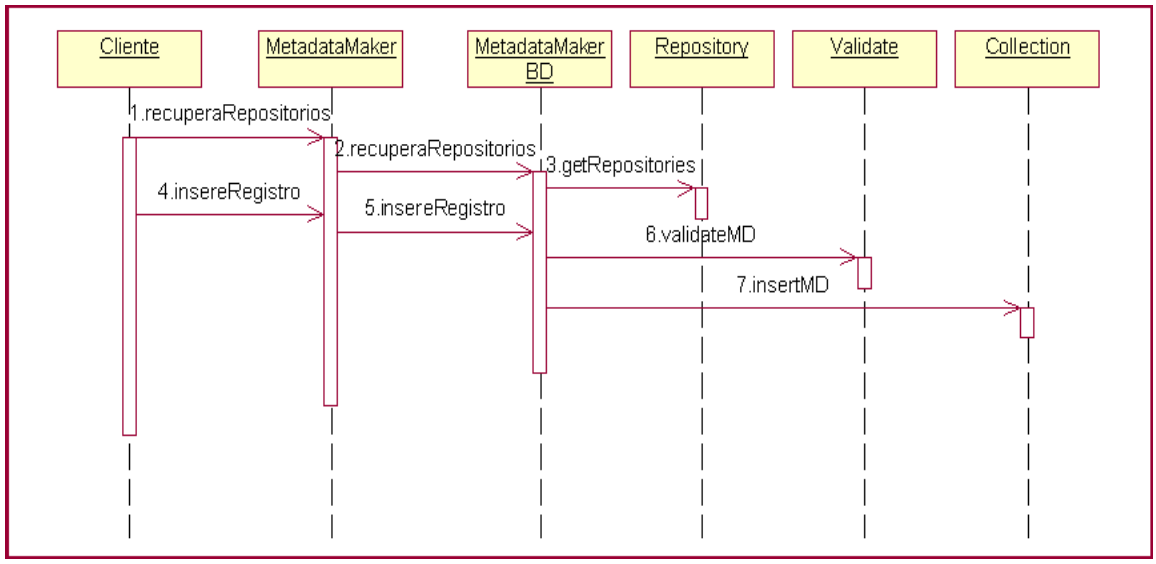


Figura 46 - Diagrama de interação da operação "*InsereRegistro*"

- Para a operação de **Atualização,** a interface apresentada ao provedor de metadados deve conter todos os dados do registro que ele deseja alterar. Para que isto seja possível, o recurso desejado deve ser previamente recuperado. A sub-aplicação *MetadataMaker* não é dotada das funcionalidades de pesquisa existentes na Dilib. Isto significa que o

provedor de metadados deve dispor de informações sobre o registro que deseja alterar. Tais informações, dizem respeito ao repositório, coleção e identificador do recurso. Diante disso, observa-se a necessidade de duas solicitações ao *Front Controller MetaMaker:* uma de recuperação e outra de atualização. De forma análoga à operação anterior, os valores dos registros devem ser carregados em um arquivo XML cuja referência será passada como parâmetro. A Figura 47 apresenta um diagrama de interação que esboça, em alto nível, o processo de atualização de registro de metadados no repositório centralizado de metadados.

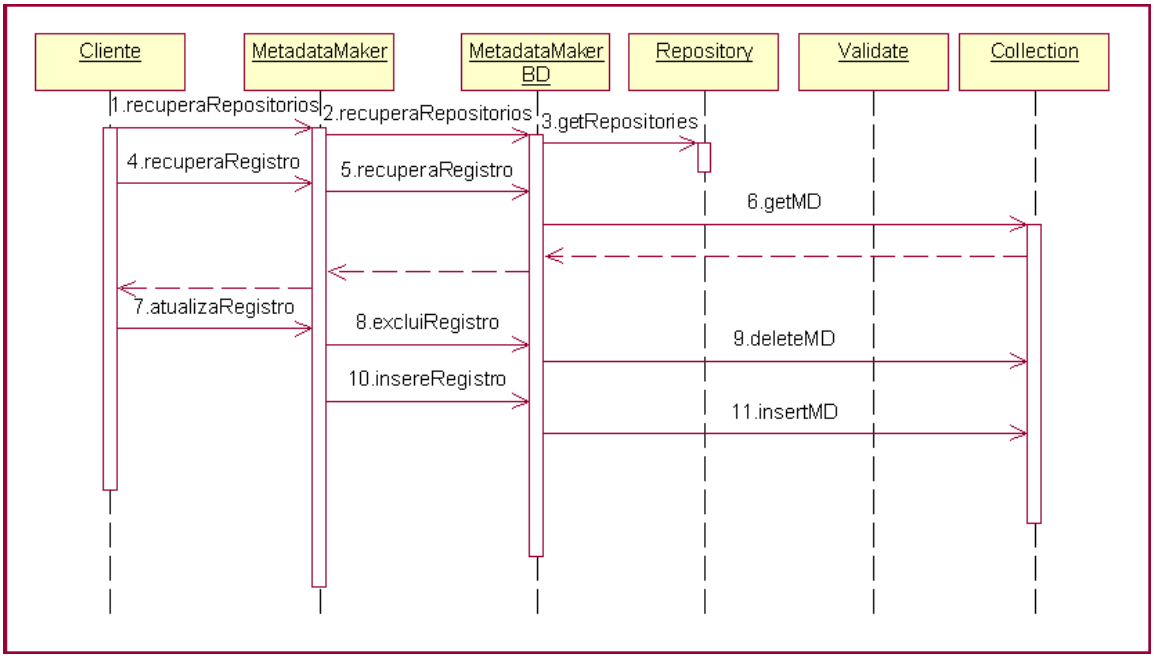


Figura 47 - Diagrama de interação da operação "*AtualizaRegistro*"

- A interface apresentada ao provedor de metadados para a realização de uma **exclusão** de registro é bastante semelhante à apresentada para a operação de alteração. Antes de efetivar a exclusão do registro o provedor de metadados deverá visualizá-lo. Assim, o mesmo procedimento discutido para a operação de atualização, se aplica também neste caso, salvo algumas exceções. Não haverá necessidade de transferir para um arquivo XML o conteúdo do registro. Além disto, a segunda operação invocada ao *Front Controller MetaMaker* é de exclusão, devendo ser passado como parâmetro o *id* do registro. A Figura 48 apresenta um diagrama de interação que esboça, em alto nível, o

processo de exclusão de registro de metadados do repositório centralizado de metadados.

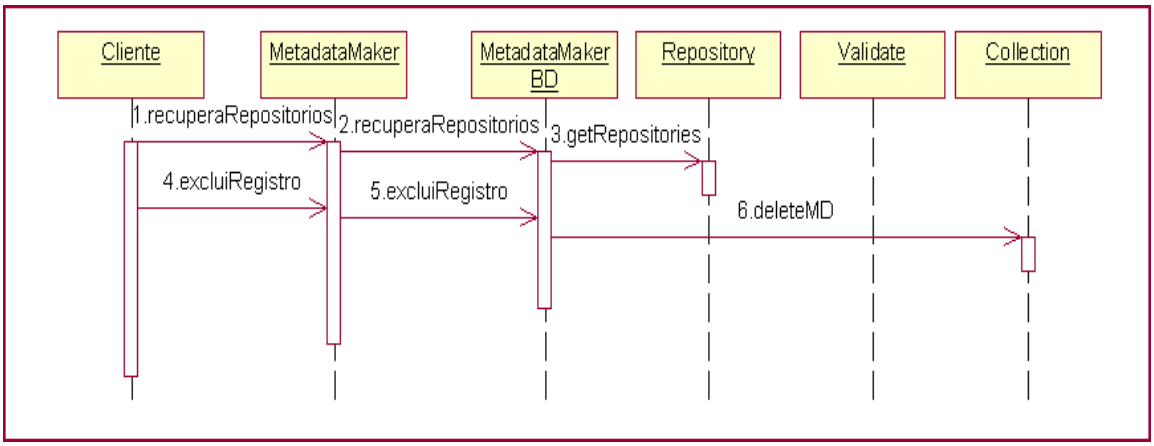


Figura 48 - Diagrama de interação da operação "*ExcluiRegistro*"

4.1.2.2 MetaMaker

O módulo funcional *MetaMaker* é responsável por disponibilizar ao cliente as operações necessárias para o cumprimento das funcionalidades da *MetadataMaker*. O *MetaMaker* implementa o padrão de projeto *Front Controller*. Fazem parte deste módulo o *stateful session bean MetaMaker,* além de classes auxiliares para a manipulação de arquivos XML e serialização de atributos de metadados.

O *MetaMaker* centraliza o recebimento das solicitações do cliente. Todavia, não faz acesso direto aos componentes da camada *Model* para efetivar as tarefas. O *MetaMaker* cria uma instância do EJB *MetaMakerBD* e faz acesso às suas operações através de uma interface remota. O quadro, a seguir, apresenta um excerto do código do *MetaMaker*.

```
...
public class MetaMakerEJB implements SessionBean {

          private SessionContext sc = null;
          private MetaMakerBDHome mmbd = null;

          public MetaMakerEJB() {}
          public void setSessionContext(SessionContext sc) {
             this.sc = sc;
          }
          public void ejbCreate() {
             try {
```

```
                    mmbd = lookupMetaMakerBD();
                } catch (NamingException ex) {
                    throw new CreateException(ex.getMessage());
                }
            }
            public void ejbActivate() {
                try {
                    mmbd = lookupMetaMakerBD();
                } catch (NamingException ex) {
                    throw new CreateException(ex.getMessage());
                }
            }
            public void ejbPassivate() {
                mmbd = null;
            }
...
}
```

As operações oferecidas pelo EJB *MetaMaker* não são idênticas às disponibilizadas remotamente pelo *MetaMakerBD* (sub-seção 4.1.2.3). O *MetaMaker* faz o tratamento e as solicitações necessárias para cada operação. Uma operação de atualização, por exemplo, é traduzida em uma operação de exclusão e inserção. Esta decisão de projeto foi tomada a fim de aproveitar o procedimento de validação já realizado pelo processo de inserção.

Nas operações de inserção, o *MetaMaker* faz a serialização do conteúdo nos arquivos XML. Para tanto, utiliza a classe auxiliar *MetadataAttributes*. Os valores dos elementos são mapeados para atributos daquela classe. A instância montada desta classe será passada como parâmetro para a operação "insereMD" do EJB *MetaMakerBD*.

A Tabela 17 apresenta as operações disponibilizadas pelo MetaMaker.

Operações do MetaMaker
Login
RecuperaRepositorios

RecuperaColecoes
RecuperaRegistro
ExcluiRegistro
AtualizaRegistro
InsereRegistro

Tabela 17 - Operações do *MetaMaker*

4.1.2.3 MetaMakerBD

O módulo funcional *MetaMakerBD* implementa o padrão de projeto *BusinessDelegates*. Ele faz acesso aos elementos de negócio para a efetivação das operações. Compõem este módulo os session beans *MetaMakerBD* e *Validate.* Os módulos funcionais da Dilib que interagem com a sub-aplicação *MetadataMaker*, o fazem com os elementos deste módulo funcional. O diagrama de colaboração da Figura 49 representa um esboço dos processos realizados na sub-aplicação pelos elementos do *MetaMakerBD.*

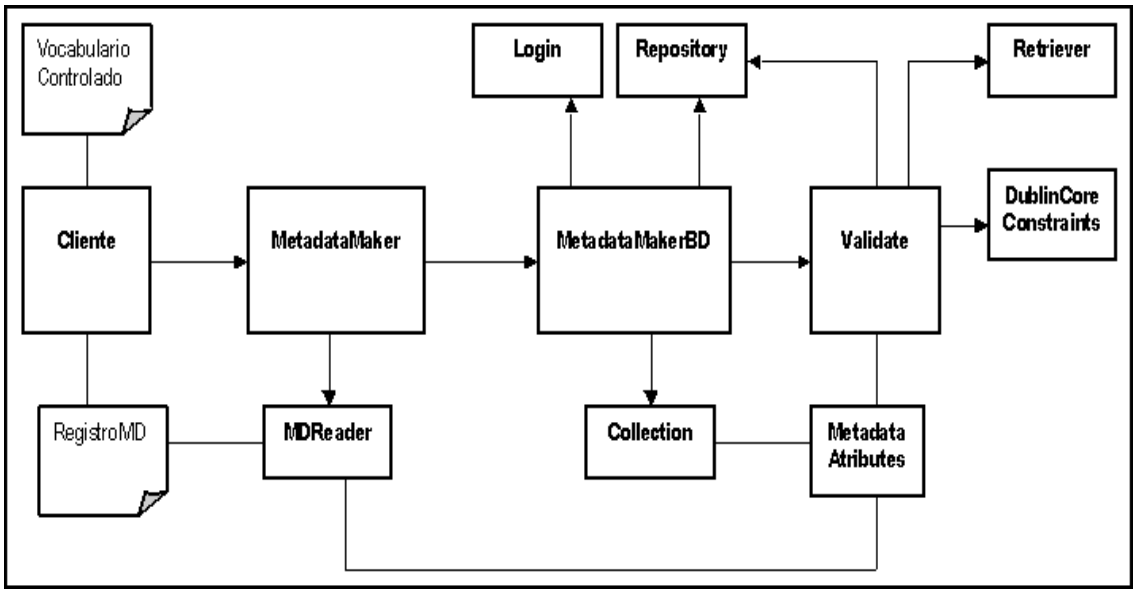


Figura 49 - Diagrama de colaboração do *MetadataMaker*

- *MetaMakerBD*

No momento em que o *MetaMakerBD* é criado, uma instância para cada componente que ele interage também é criada. Entretanto, as instâncias do EJBs *Repository, Login, Collection* e *Validate* só são utilizados nos momentos oportunos.

Na invocação ao método *Login,* o *MetaMakerBD* utiliza a sua referência ao EJB *Login*. Ele envia os parâmetros que lhe foram passados, além de um outro que determina o perfil do usuário requerido. O retorno, denotando o sucesso ou insucesso na operação, será feito ao *FrontController*.

A interação com o módulo funcional *Repository* é realizada em duas ocasiões: no método que recupera os repositórios cadastrados para a biblioteca e no método que recupera as coleções disponíveis no repositório. Estas consultas são feitas através da referência ao EJB *Repository* com a passagem de parâmetros adequada.

O *MetaMakerBD* utiliza o módulo funcional *Collection* para realizar as inserções dos registros no repositório centralizado de metadados. Optou-se por não manipular diretamente o *entity bean Metadata*, mas sim, fazê-lo através do *session bean Collection*. Isto permitiria um maior desacoplamento dos componentes de negócio do projeto. A operação de inserção de um registro de metadados é bastante criteriosa, a fim de cumprir as premissas de consistência determinadas para este módulo. Antes de invocar o método de inserção do EJB *Collection*, o *MetaMakerBD* utiliza outro elemento deste módulo, o EJB *Validate*, para as operações de validação. É passado como parâmetro para o método de validação uma instância da classe *MetadataAttributes*. As outras operações relativas à exclusão e à recuperação do registro de metadados com base no *identifier* são feitas diretamente.

- *Validate*

O *Validate* é responsável por realizar as verificações necessárias para que um registro de metadados seja considerado válido. Para a realização destas verificações o EJB *Validate* recebe uma instância da classe *MetadataAttributes* contendo os dados do registro e interage com os módulos *Repository*, *Retriever* e *Login* da Dilib. No momento da criação do *session bean Validate*, são criadas instâncias a estes componentes. As referências a estas instâncias são utilizadas no momento adequado. A classe auxiliar à *DublinCoreConstraints* também é utilizada no processo de validação.

As verificações devem ser realizadas em cima dos seguintes pontos: repositório, coleção, identificador do recurso e a existência dos atributos obrigatórios convencionados. O diagrama de interação da Figura 50 ilustra, em alto nível, o processo de validação disparado pelo *MetaMakerBD* quando recebe uma solicitação de inserção.

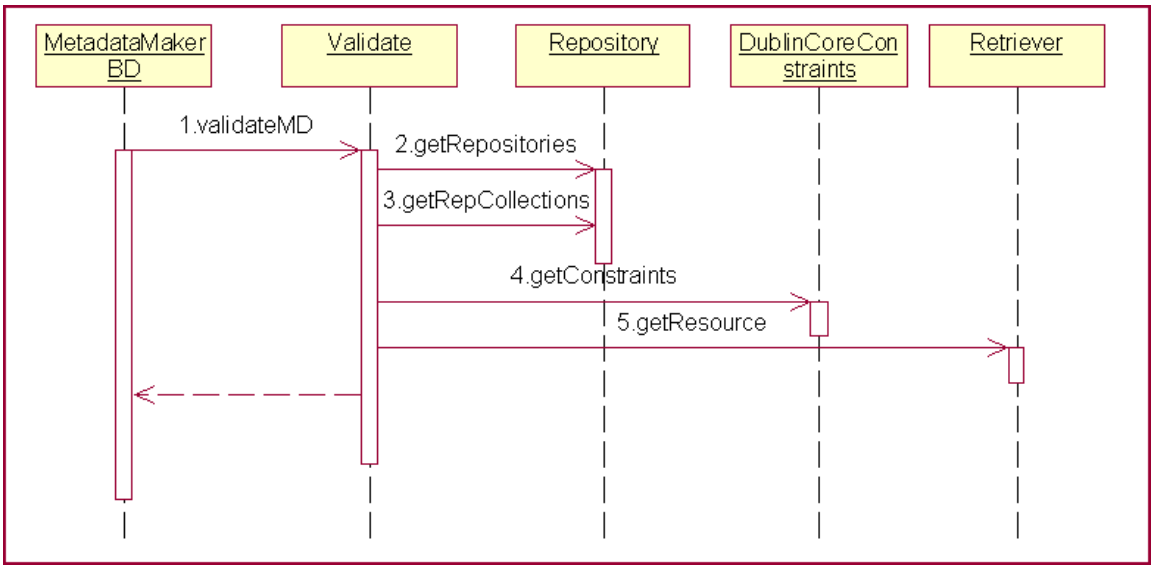


Figura 50 - Diagrama de interação da operação "*Validate*"

A validação do repositório e coleção do registro de metadados é baseada nas informações dos atributos *identifier* e *relation*. O primeiro passo é verificar se a coleção informada no *identifier* é a mesma informada no *relation*. O próximo passo é verificar se o repositório informado em id está cadastrado na Dilib. Isto deve ser feito analisando a lista de repositórios retornados após a invocação de um método do EJB *Repository* que devolve os repositórios cadastrados. Por fim, verifica-se se a coleção informada está cadastrada no repositório informado. Esta análise será feita com base na lista de coleções retornadas após a invocação de um método do EJB *Repository* que devolve as coleções cadastradas em determinado repositório. Se a verificação foi positiva nos três testes, o repositório e a coleção informada são considerados válidos.

A validação do identificador do recurso, terceira parte do atributo *identifier,* é realizada para garantir a consistência entre o recurso e o registro que o descreve. Utilizando o módulo *Retriever,* é feita uma consulta passando o identificador único do recurso. Se a consulta foi bem sucedida, o identificador é considerado válido.

Convencionou-se que, para os registros de metadatos do Diliframe, determinados elementos do Dublin Core seriam obrigatórios na descrição do recurso. Para que o repositório seja considerado válido, os elementos definidos como obrigatórios devem ter valor associado. Para realizar esta verificação o EJB *Validate* conta com a classe auxiliar *DublinCoreConstraints* que contém as informações de quais elementos são considerados obrigatórios e quais são os opcionais. Se a checagem realizada for satisfatória e o repositório, coleção e identificador do recurso também, o registro, finalmente, é considerado válido.

4.1.3 A RepositoryMaker

A sub-aplicação do Diliframe *RepositoryMaker* é utilizada pelo provedor de dados para carregar o banco de dados de recursos. O seu objetivo é facilitar o processo de povoamento do repositório, principalmente quando o acervo da biblioteca digital é formado por dados multimídia.

A *RepositoryMaker* apresentará uma interface simplificada na qual o usuário não deverá escrever nenhum comando SQL para a realização da operação de inserção de um recusro. O repositório deverá ser previamente cadastrado na Dilib pelo administrador de dados. As informações relativas à localização do repositório, fornecedor do banco de dados e comandos SQL para a inserção devem ter sido cadastradas nesta oportunidade. Como visto na sub-seção 4.1.1.4, o arquivo XML repositoryDAOSQL.xml tem esta configuração. É necessário também que a tabela que armazenará os dados esteja criada adequadamente no banco.

De forma análoga à sub-aplicação *MetadataMaker,* a *RepositoryMaker* também é implementado como uma aplicação J2EE cliente. Seus módulos funcionais seguem a arquitetura definida pelo padrão MVC. São eles: Cliente, *RepMaker* e *RepMakerBD.* Para a efetivação de suas funcionalidades, a sub-aplicação interage com os módulos funcionais *Login* e *Repository* da sub-aplicação Dilib. Além destes, as classes DAO utilizadas pelo *Retriever* também são aproveitadas pelo módulo funcional *DBAccess.* A Figura 51 apresenta um diagrama de colaboração de alto nível dos processos realizados na *RepositoryMaker.*

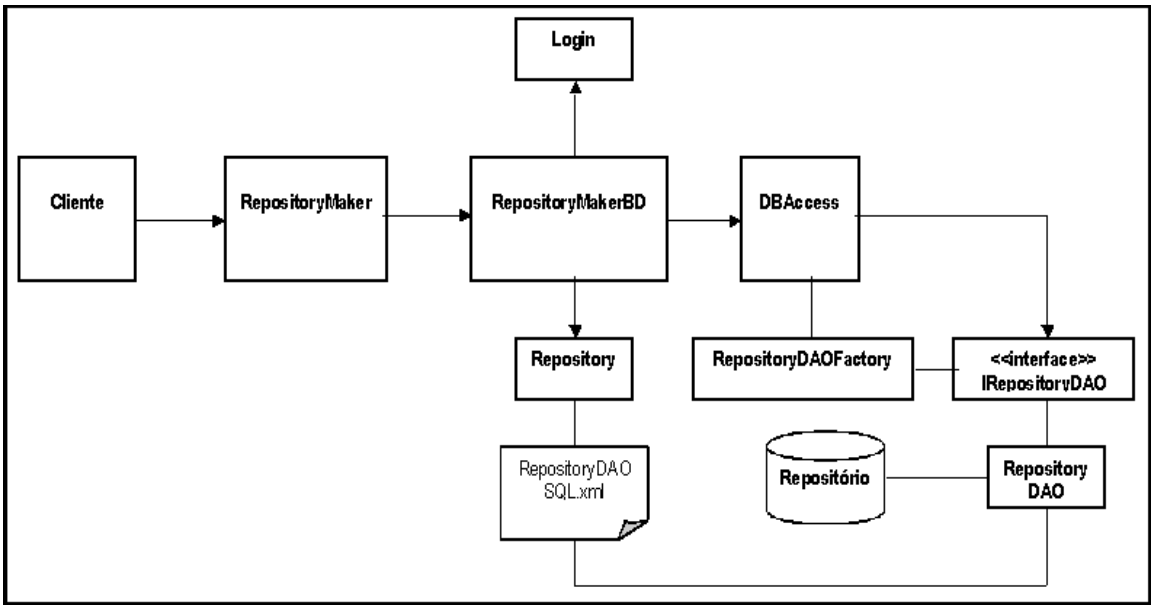


Figura 51 - Diagrama de colaboração da *RepositoryMaker*

4.1.3.1 Cliente

O módulo Cliente é implementado como uma classe tradicional Java. Esta é a porta de entrada da sub-aplicação que oferece a interface adequada para as operações disponibilizadas: *Login* e inserção. É através de uma interface local que o EJB *RepMaker,* que é o *Front Controller* da aplicação, disponibiliza ao Cliente suas operações. Para tanto, o Cliente deve criar uma instância do componente e, através de sua referência, invocar os métodos adequados.

- Antes do acesso às funcionalidades da sub-aplicação, o provedor de dados deve ser autenticado e autorizado pelo processo de **Login**. A interface apresentada pelo Cliente apresenta dois campos onde devem ser informados o nome e a senha do usuário. Esta solicitação é encaminhada ao *Front Controller RepMaker* passando os parâmetros adequados. A Figura 52 apresenta um diagrama de interação que ilustra, em alto nível, o processo de *Login*.

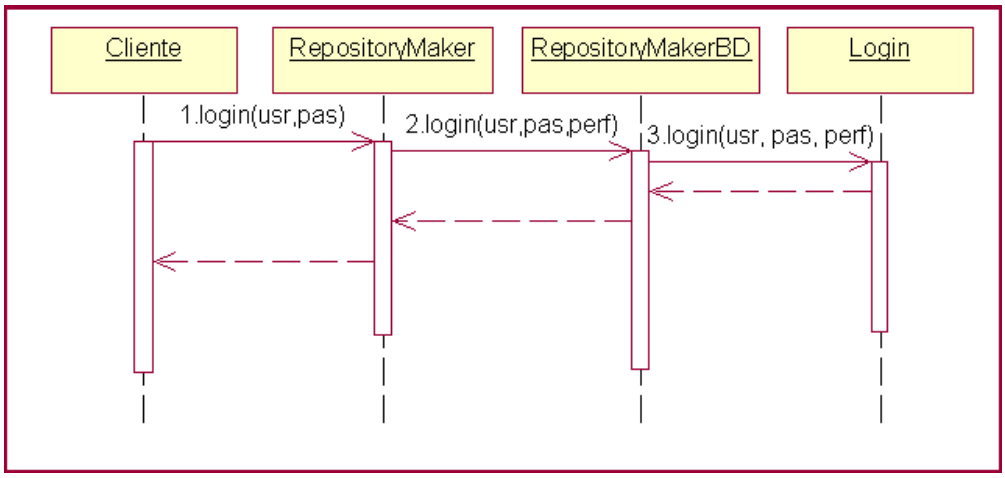


Figura 52 - Diagrama de interação da operação de "*Login*"

- Na operação de **Inserção** o provedor de dados seleciona de uma lista de repositórios ao qual deseja adicionar um recurso. Os recursos inseridos através do *RepMaker* são arquivos de imagem, som, vídeo e texto. A interface apresentada para a escolha dos arquivos é uma árvore de diretórios na qual ele selecionará o recurso desejado ou até mesmo um diretório completo. Neste último caso, serão disparadas tantas inserções quantos forem os arquivos presentes no diretório escolhido. A solicitação da operação de inserção será encaminhada ao *Front Controller RepMaker,* passando como parâmetro uma referência ao arquivo do recurso. A Figura 53 apresenta um diagrama de interação que esboça, em alto nível, o processo de inserção de recurso no repositório de dados.

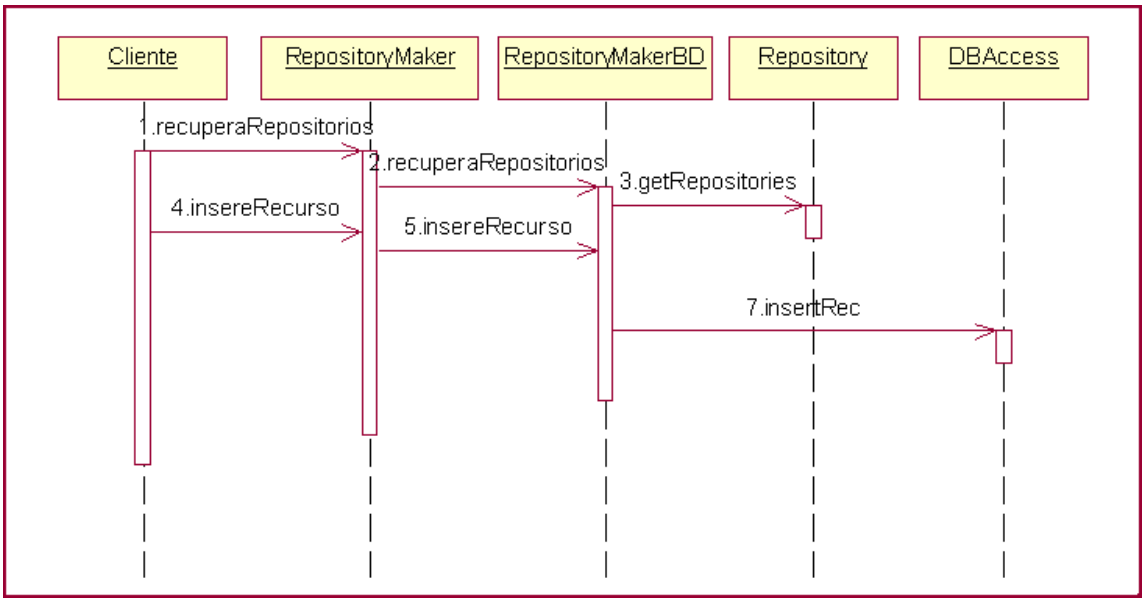


Figura 53 - Diagrama de interação da operação "*insereRecurso*"

4.1.3.2 RepMaker

O módulo funcional *RepMaker* é composto pelo *stateful session bean RepMaker* e algumas classes auxiliares que apóiam o processo. Este módulo implementa o padrão de projeto *Front Controller,* e, assim, disponibiliza ao cliente as operações necessárias para a implementação das funcionalidades da sub-aplicação.

No momento da criação do *RepMaker,* uma instância do EJB *RepMakerBD* também é criada. O *RepMaker* não faz acesso direto aos componentes da *Model* para a efetivação das operações. Ele apenas os encaminha invocando os métodos e fazendo acesso às suas operações, através do EJB *RepMakerBD,* a partir da referência criada. O quadro, a seguir, apresenta um excerto do código do EJB *RepMaker*.

```
...
public class RepositoryMakerEJB implements SessionBean {

          private SessionContext sc = null;
          private RepositoryMakerBDHome rmbd = null;

          public RepositoryMakerEJB() {}
          public void setSessionContext(SessionContext sc) {
             this.sc = sc;
          }
```

```
            public void ejbCreate() {
               try {
                      rmbd = lookupRepositoryMakerBD();
               } catch (NamingException ex) {
                      throw new CreateException(ex.getMessage());
               }
            }
            public void ejbActivate() {
               try {
                      mmbd = lookupRepositoryMakerBD();
               } catch (NamingException ex) {
                      throw new CreateException(ex.getMessage());
               }
            }
            public void ejbPassivate() {
               rmbd = null;
            }
...
}
```

Na operação de inserção, o *RepMaker* recupera o conteúdo do arquivo cuja referência foi passada como parâmetro e o armazena em uma classe Java de armazenamento de bytes (*InputStream* ou *ByteArray*). Para o EJB *RepMakerBD* será encaminhada uma instância desta classe como parâmetro da operação "insereRecurso".

A Tabela 18 apresenta as ações disponibilizadas pelo *RepMaker*.

Ações do RepMaker
Login
recuperaRepositorios
recuperaColecoes
recuperaRegistro
excluiRegistro
atualizaRegistro
insereRegistro

Tabela 18 – Ações do *RepMaker*

4.1.3.3 RepMakerBD

O módulo funcional *RepMakerBD* implementa o padrão de projeto *BusinessDelegates*. É através dele que os componentes de negócio são localizados, criados e acessados. O *RepMakerBD* disponibiliza uma interface remota através da qual serão realizadas solicitações para que ele faça o encaminhamento. Fazem parte deste módulo os *session beans RepMakerBD* e *DBAccess.*

- *RepMakerBD*

As operações de login e de recuperação de usuários cadastrados na Dilib são realizadas através da interação com os módulos *Retriever* e *Login* da Dilib. No momento de criação do EJB *RepMakerBD,* também são criadas instâncias para os EJBs *Login* e *Repository*.

Na invocação ao método *Login,* o *RepMakerBD* utiliza a sua referência ao EJB *Login*. Para esta operação é passada, além dos parâmetros enviados (nome e senha), a característica do perfil de acesso que se deseja validar: se o usuário é provedor de dados. O retorno desta operação, vinda do EJB *Login,* deve ser enviado ao *FrontController RepMaker.*

Na invocação do método "recuperaRepositórios", o *RepMakerBD* utiliza sua referência ao EJB *Repository.* Como resultado, são retornados todos os repositórios cadastrados na Dilib. Este retorno deverá ser enviado ao *FrontController RepMaker.*

- *DBAccess*

O *session bean DBAccess* é responsável por realizar as operações de inserção nos repositórios de dados. Para tanto, ele se utiliza das mesmas classes auxiliares que o módulo funcional *Retriever* da Dilib. Tais classes são as classes DAO que implementam o acesso ao banco de dados do repositório: *RepositoryDAOFactory*, *IRepositoryDAO* e *RepositoryDAO.* Optou-se por não utilizar o EJB *Retriever* para esta tarefa por que ele não dispunha do método de inserção. Esta decisão de projeto merece ser revista, se o interesse for criar um componente mais genérico e reutilizável.

A operação de inserção desencadeia a criação da classe DAO de acesso ao banco de dados. Para ela, devem ser passados como parâmetros o identificador do recurso e os bytes a ele correspondentes. A consistência com relação à chave primária é realizada no

banco de dados. Assim, não será permitida a inserção de dois recursos com o mesmo identificador.

4.2 Estratégia para implementação

O processo de construção do Diliframe foi planejado como sendo iterativo e incremental. A característica evolutiva do framework estaria contemplada em cada ciclo de iteração, ou seja, o framework evoluiria e cada iteração apresentaria a nova versão, após as críticas de seus usuários.

O refinamento da análise, projeto e implementação, a cada ciclo de iteração, permitem fazer o ajuste fino das expectativas de seus usuários. Além disso, mantém o framework atualizado, com relação às inovações tecnológicas que venham contribuir positivamente para seu projeto.

A princípio, o enfoque dado ao Diliframe era bem mais simplista. O framework constituía-se de um conjunto de elementos com os quais o desenvolvedor poderia contar para a construção de sua aplicação. Após os refinamentos dos requisitos dos usuários chegou-se ao projeto apresentado na seção 4.1.

A estratégia definida para a implementação do Diliframe fundamentou-se na composição de produtos finais em cada ciclo de iteração. Ao final da análise do domínio e do projeto do framework, foram divididos os ciclos de iteração necessários para a consecução do projeto e o produto final de cada uma destas fases. Cada ciclo de iteração contemplará alguns dos requisitos funcionais definidos (sub-seção 3.2.1) e algumas funcionalidades determinadas pelos casos de uso.

De acordo com o modelo da arquitetura apresentada, o Diliframe divide-se em três sub-aplicações: *Dilib*, *MetadataMaker* e *RepositoryMaker*. Em torno da Dilib, giram a maioria dos requisitos funcionais e dos casos de uso. A *MetadataMaker* e o *RepositoryMaker* que atendem, respectivamente, ao provedor de metadados e ao provedor de dados, também estão associados a alguns requisitos funcionais e casos de uso, entretanto, em menor número.

A estratégia adotada para a distribuição dos requisitos funcionais e dos casos de uso nas fases de implementação leva em consideração, horizontalmente, as sub-aplicações e, verticalmente, as camadas do MVC. Isto significa que o primeiro passo é determinar qual

a ordem de implementação das sub-aplicações e em seguida, para cada sub-aplicação, qual a ordem de implementação de seus módulos funcionais, de acordo com a sua localização: na *View*, na *Model* ou na *Controller*. Esta estratégia e as escolhas que serão feitas devem se apoiar em princípios já estabelecidos na resolução deste tipo de problema. A Tabela 19 apresenta situações ou qualificações que os módulos, que implementam as funcionalidades retratadas nos casos de uso, devem ter para que sua implementação seja priorizada [Larman, 1998].

PRIORIDADE	SITUAÇÃO DO MÓDULO FUNCIONAL
1	Representa impacto significativo no projeto arquitetural, isto é, implementa vários componentes do negócio ou requer serviços de persistência.
2	Contém funções complexas, arriscadas ou de críticas.
3	Representa as funcionalidades fundamentais do negócio.
4	Envolve pesquisa ou investimento em tecnologia nova ou arriscada.

Tabela 19 - Considerações na priorização de módulos para implementação

Diante do exposto, optou-se por priorizar a implementação da Dilib, e, em seguida, da MetadataMaker e da RepositoryMaker. No escopo das sub-aplicações, será dada prioridade aos módulos funcionais contidos na camada *Model*, seguidos pelos contidos na camada *Controller* e, em seguida, na camada *View*. Para conciliar esta estratégia, com a necessidade de que cada ciclo de iteração apresente a funcionalidade exigida pelos casos de uso a eles associados, é necessária a criação de elementos auxiliares. Estes elementos serviriam para fornecer interface adequada, carga no banco de dados, dentre outros.

A definição dos ciclos de interações levou em consideração as premissas aqui discutidas, bem como a associação aos requisitos funcionais levantados para o Diliframe e seus casos de uso. Com relação ao atendimento ao requisito funcional R7-"Suportar a adoção de um esquema de cobrança", foi definido que os *hot spots* deixados no componente *User* seriam suficientes para dar o suporte necessário. A Tabela 20 apresenta a divisão dos ciclos. As fases são numeradas e indicam qual sub-aplicação será nela

trabalhada. Em seguida, é apresentado o título da fase que indica o seu objetivo. A evolução da aplicação e o aumento de sua complexidade são demonstrados através de um conjunto de oito características da biblioteca. Para finalizar, são informados os casos de uso e os requisitos funcionais contemplados no ciclo, bem como, os elementos resultantes sob a forma de produto final.

Diliframe Fase 1 – Dilib: Funcionamento Mínimo
Pesquisa: Baseada em atributos de Metadados; **Localização dos recursos**: Centralizada; **Localização dos metadados**: Centralizada; **Padrão de metadados**: Padrão único Dublin Core; **Armazenamento**: Homogêneo, fornecedor único; **Tipo de dados**: Texto; **Segurança**: Não implementada; **Cobrança**: Não implementada; **Casos de Uso**: Pesquisar_Item (Simples), Recuperar_Item; **Requisitos**: RF1, RF2 e RF4; **Produtos finais**: Searcher e Retriever.
Diliframe Fase 2 – Dilib: Biblioteca Distribuída
Pesquisa: Baseada em atributos de Metadados; **Localização dos recursos**: Distribuída; **Localização dos metadados**: Centralizada; **Padrão de metadados**: Padrão único Dublin Core; **Armazenamento**: Homogêneo, múltiplos fornecedores; **Tipo de dados**: Texto; **Segurança**: Não implementada; **Cobrança**: Não implementada; **Casos de Uso**: Listar_Coleções, Selecioanar_Coleções; **Requisitos**: RF5; **Produtos finais**: Repository.
Diliframe Fase 3 – Dilib: Administração de usuários, coleções e repositórios
Pesquisa: Baseada em atributos de Metadados;

Localização dos recursos: Distribuída; **Localização dos metadados**: Centralizada; **Padrão de metadados**: Padrão único Dublin Core; **Armazenamento**: Homogêneo, múltiplos fornecedores; **Tipo de dados**: Texto; **Segurança**: Autorização e autenticação de acesso para as funcionalidades administrativas; **Cobrança**: Não implementada; **Casos de Uso**: Cadastrar_Usuário, Listar_Usuários, Excluir_Usuário, Alterar_Usuário, Cadastrar_Coleção, Listar_Coleções, Excluir_Coleção, Alterar_Coleção, Cadastrar_Repositório, Listar_Repositório, Excluir_Repositório, Alterar_Repositório, Efetuar_Login; **Requisitos**: RF9, RF6; **Produtos finais**: User, Collection e Login.
Diliframe Fase 4 – Dilib: Biblioteca Multimídia
Pesquisa: Baseada em atributos de Metadados; **Localização dos recursos**: Distribuída; **Localização dos metadados**: Centralizada; **Padrão de metadados**: Padrão único Dublin Core; **Armazenamento**: Homogêneo, múltiplos fornecedores; **Tipo de dados**: Imagem, texto, vídeo e áudio; **Segurança**: Autorização e autenticação de acesso para as funcionalidades administrativas; **Cobrança**: Não implementada; **Casos de Uso**: Pesquisar_Item (Avançada); **Requisitos**: RF3; **Produtos finais**: Searcher, Controller e View Assembler.
Diliframe Fase 5 – MetadataMaker: Suporte à criação de Metadados
Esta fase deve contemplar a criação completa da MetadataMaker uma vez que os elementos que ele reutiliza da Dilib já estão prontos. **Casos de Uso**: Listar_Repositórios, Listar_Coleções, Manipular_Metadado, Recuperar_Item, Criar_Metadado; **Requisitos**: RF10;

Produtos finais: Todos os módulos funcionais da MetadataMaker, com exceção dos módulos reutilizados da Dilib.
Diliframe Fase 6 – RepositoryMaker: Suporte à carga de repositórios
Esta fase deve contemplar a criação completa da RepositoryMaker uma vez que os elementos que ele reutiliza da Dilib já estão prontos. **Casos de Uso**: Listar_Repositórios, Inserir_Recurso; **Requisitos**: RF10; **Produtos finais**: Todos os módulos funcionais da RepositoryMaker, com exceção dos módulos reutilizados da Dilib

Tabela 20 - Descrição de ciclos de iteração

Capítulo 5

Conclusão

Este trabalho de pesquisa apresentou o empenho no sentido de projetar e especificar uma solução que facilite o desenvolvimento de sistemas de bibliotecas digitais. Para tanto, foi proposto um framework de aplicação construído em cima da plataforma Java para aplicações corporativas. O Diliframe é destinado ao desenvolvedor de bibliotecas digitais e busca aumentar sua produtividade, diminuindo o tempo de projeto através do reuso que proporciona.

A maximização da demanda da construção de bibliotecas digitais é resultado da tendência mundial de valorização da informação e da evolução de bibliotecas tradicionais. Cada vez é mais freqüente a disponibilização de acervos especializados por universidades, institutos e centros de pesquisa. As bibliotecas digitais fazem a ponte entre a informação e o indivíduo que a procura. Esta ligação pode ser feita de forma mais eficiente, flexível e confiável que os mecanismos de busca tradicionais da Web.

Há diversos projetos em todo o mundo, tanto acadêmicos quanto comerciais, na área de bibliotecas digitais. Algumas iniciativas visam a facilitar o trabalho do desenvolvedor na criação deste tipo de sistemas. O Diliframe destaca-se por oferecer uma arquitetura aberta, flexível e modular, na qual serviços são adicionados facilmente, de acordo com a necessidade da nova aplicação. Além disso, destaca-se a experiência no desenvolvimento do projeto em cima de uma plataforma relativamente nova, com o intuito de herdar os prováveis benefícios trazidos por esta tecnologia.

A solução proposta pelo Diliframe, em atendimento ao princípio que norteou a condução deste trabalho: facilitar o desenvolvimento de bibliotecas digitais, determinar o padrão de comportamento da aplicação. Enfim, oferece a base em cima da qual uma nova biblioteca deve ser criada.

A personalização e a adaptação do Diliframe à realidade da nova aplicação é facilitada devido ao projeto de sua arquitetura. O Diliframe é modular, característica obtida pela aplicação do padrão MVC, o que permite a fácil adição de componentes de negócio às suas sub-aplicações. A aplicação de restrição de segurança, através de autorização de acesso e autenticação de usuário, é deixada em aberto para escolha do desenvolvedor. Isto é possível, uma vez que o módulo responsável por esta funcionalidade é implementado seguindo o padrão de projeto "*InterceptingFilter*". A personalização dos serviços oferecidos pela biblioteca, com base no perfil do usuário, já é prevista pelo Diliframe, mesmo que não seja por ele implementada. No componente que representa o usuário, está previsto um atributo que determina o perfil para que tais serviços sejam implementados. Estes serviços são alguns dos *hot spots* do Diliframe identificados sobre a sua arquitetura.

O projeto do Diliframe passou por diversos refinamentos para que atingisse a sua forma final. O estado atual do desenvolvimento do Diliframe, com base nos ciclos de iteração definidos na seção 4.2, é o de conclusão do primeiro ciclo. Os ciclos de iterações ali definidos servem como guia para o desenvolvedor que deseja implementar o Diliframe.

5.1 Perspectivas

Os trabalhos futuros vislumbrados para a evolução do Diliframe têm como ponto de partida sua implementação, com a conclusão de todos os ciclos de iteração. Os pontos que se destacam como sugestões para evolução do Diliframe estão listados a seguir:

- Aplicação do framework em diversos projetos de biliotecas digitais de pequeno e grande porte, a fim de que se coletem mais requisitos com o desenvolvedor. Com estas informações seria possível o correto direcionamento de novas versões do projeto.
- Unificação das sub-aplicações do Diliframe em um único sistema Web. Com isso seria necessário rever as interfaces dos componentes EJBs que, em sua maioria, foram definidas como locais.
- Armazenar os registros de metadados como arquivos XML. Esta mudança envolve diretamente a estrutura do framework. Com relação aos pontos listados anteriormente, esta mudança seria mais complexa de realizar. Entretanto, com isto, seriam lançadas as bases para possiblitar a

interoperabilidade com outras bibliotecas que utilizam o XML para a descrição de recursos.

- Implementação de formas alternativas de pesquisa. Atualmente o Dilib contempla apenas a busca com base em atributos de metadados. Opcionalmente, poderia ser oferecida a busca textual (*full-text retrieval*) [Yu, 1994] ou a recuperação baseada em conteúdo (*content-based retrieval*) [Sheth, 1998].
- Mudança do paradigma do repositório centralizado de metadados para um repositório descentralizado. Isto possibilitaria uma maior independência entre o sistema e os dados, podendo, por exemplo, os metadados serem organizados juntos com os dados.
- Permitir a utilização de outros padrões de metadados para a descrição do recurso, que não o estabelecido pelo Diliframe. Para tanto, seria necessária a utilização de tradutores que transformariam a consulta feita com base no padrão original do Diliframe para o padrão adotado pela coleção.

Estas duas últimas medidas visam a diminuir as restrições impostas pelo Diliframe com relação aos dados.

5.2 Resultados

Considerando os requisitos funcionais e não-funcionais traçados no início deste projeto, constata-se que seus objetivos foram parcialmente atingidos.

Um dos requisitos não-funcionais, considerado como muito importante, não foi atendido.

"RNF5 – Deve oferecer uma rica documentação dos módulos envolvidos"

A documentação de um framework é peça fundamental para a sua reutilização. A única documentação disponibilizada para o Diliframe é esta dissertação, o que dificulta a sua utilização real.

Considera-se que os demais requisitos foram satisfatoriamente atendidos.

Referências Bibliográficas

[Arms, 1995] ARMS, W., Key Concepts in the Architecture of the Digital Library**,** *D-Lib Magazine*, 1995. Acessível por: http://www.dlib.org em Fevereiro de 2001.

[Arms, 2001] ARMS, W., Digital Libraries**,** *MIT Press*, 2001.

[ARXIV, 2002] arXive.org e-Print archive. Acessível por: http://arxiv.org/ em outubro de 2002

[Baptista, 2000] BAPTISTA, C. Steplib: A Digital Library for Spatio-Temporal and Multimedia Data. *Tese de doutorado*, Universidade de Kent em Canterbury, 2000.

[BibSFront, 2002] Acessível por: http://www.bn.br/bibsemfronteiras/index.html, em Maio de 2002.

[Borbinha, 2000] BORBINHA, J. . Elementos do núcleo de metadata "Dublin Core", versão 1.1. *Dublin Core Metadata Initiative*. Acessível por: http://bnd.bn.pt/ed/dcmes/dcmes11-20000518.htm em Outrubro de 2002.

[Biggerstaff, 1989] BIGGERSTAFF, T. *Software Reusability V1. Concepts and Models*. ISBN 0201080176, Addison Wesley Pub, 1989.

[Cockburn, 1997] COCKBURN, A. Structuring Use-Cases with goals. Journal of Object-Oriented Programming 10(7):35-40, 1997 .

[Coleman, 1998] COLEMAN, D. *A use case template: draft for discussion,* 1998. Hewlett-Packard Software Initiative.

[Cosumano, 1999] COSUMANO, M., YOFFIE, D. Software Development on Internet Time. *IEEE Computer*, Outubro 1999.

[Cox, 2000] COX, S. DCMI Period Encoding Schema: Specification of the limits of na interval time and methods for encoding this in a text string. *Dublin Core Metadata Initiative*. Acessível por: http://dublincore.org/documents/2000/07/28/dcmi-period/ em Outubro de 2002

[Cox, 2000b] COX, S. DCMI Box Encoding System: specification of the spatial limits of a place and methods for encoding this in a text string. *Dublin Core Metadata Initiative*. Acessível por: http://dublincore.org/documents/2000/07/28/dcmi-box/ em Outubro de 2002

[Crupi, 2001] CRUPI, J., MALKS, D. e ALUR, D. *Core J2EE Patterns*, ISBN 0130648841, Prentice Hall, 2001

[D'Souza, 1999] D'SOUZA, D. e WILLS, A. . *Objects, Components and Frameworks with UML.* ISBN: 0201310120, Addison-Wesley, 1999.

[DC, 2002] Dublin Core Metadata Intiative. Acessível por: http://dublincore.org/ em Outrubro de 2002

[DDC, 2002] Dewey Decimal Classification. Acessível por: http://www.oclc.org/dewey/ em Outrubro de 2002.

[DESIRE, 2002] Development of a European Service for Information on Research and Education. Acessível por: http://www.desire.org em Outubro de 2002

[deVries, 1998] DeVRIES, A. EBERMAN, B. e KOVALCIN, D. The Design and Implementation of an Infrastructure for Multimedia Digital Libraries. *Proceedings of the 1999 International Database Engineering and Application Symposium, Wales*, 1998. Pp. 103-120

[DLI1, 1994] Digital Library Initiative – Primeira Fase. Acessível por: http://www.dli2.nsf.gov/dlione/ em Fevereiro de 2001

[DTF, 2002] Date Time Format. WorldWideWeb Consortium, Acessível por: http://www.w3.org/TR/NOTE-datetime em Outubro de 2002.

[EELS, 2002] Engeeniring E-Library, Sweden Acessível por: http://eels.lub.lu.se/ em Outubro de 2002.

[Fayad, 1999] FAYAD, M., SCHIMDT, D. and JOHNSON, E. *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, ISBN 0471248754, Wiley, 1999.

[FGDC, 2002] Federal Geographic Data Comitte. Acessível por:

http://www.fgdc.gov/index.html em Outubro de 2002

[Fowler, 1999] FOWLER, M. e SCOTT, K. *UML Distilled – Applying the Standard Object Modeling Language – second edition*. ISBN 020165783X, Addison-Wesley, 1999.

[Fox, 1999] FOX, E., Digital Libraries Initiative (DLI) Projects 1994-1999 Em *Bulletin of the American Society for Information Science* Outubro/Novembro, 1999.

[Fox, 1999b] FOX, E. e SORNIL, O. Digital Libraries. Em *Modern Information Retrieval*. AWL England, 1999

[FBN, 2002] Fundação Biblioteca Nacional. Acessível por: http://www.bn.br/, em Maio de 2002.

[Gamma *et al*, 1994] GAMMA, E., HELM, R.; JOHNSON, R.; VLISSIDES, J., *Design Patterns: Elements of Reusable Software*. ISBN 0201633612, Addison-Wesley, 1994.

[Garcia-Molina, 1995] GARCIA-MOLINA H. e LYNCH, C. Interoperability, Scaling, and the Digital Libraries Research Agenda *IITA Digital Libraries Workshop*, 1995

[Hillmann, 2001] HILLMANN, D. Using Dublin Core. Acessível por: http://dublincore.org/documents/usageguide/ em Outrubro de 2002

[IBM, 2002] IBM. Acessível por: http://www.ibm.com em Dezembro de 2002.

[IEEE, 2000] IEEE Technical Committee on Digital Library, http://computer.org/tab/tclist/tcdl.htm

[IETF, 2002] The Internet Engineering Task Force. Acessível por: http://www.ietf.org/ em Outubro de 2002.

[ISO, 2002] International Organization for Standardization. Acessível por: http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=4767 em Outubro de 2002.

[Informix,2002] Informix. Acessível por: http://www-3.ibm.com/software/data/informix/ em Dezembro.

[Jacobson, 1992] JACOBSON, I. , CHRISTERSSON, M. e JONHSON, P. *Object-Oriented Software Engineering*, ISBN: 0201403471 Addison-Wesley, 1992

[Johnson, 1988] JONHSON, R. e FOOTE, B. Designing reusable classes. *Journal of Object-Oriented Programming*, 1(5), Junho/ Julho. 1988:22-35.

[Johnson, 1992] JONHSON, R, Documenting Frameworks Using Patterns. *Proceedings of OOPSLA* 1992, pp. 63-76, Vancouver, BC , Outubro 1992.

[LCSH, 2001] Library of Congress Subject Headings – Principles of Structure and Policies for Application. Acessível por: http://www.tlcdelivers.com/tlc/crs/shed0014.htm em Outubro de 2002.

[LCC, 2002] Library of Congress Classification Outline, Acessível por: http://www.loc.gov/catdir/cpso/lcco/lcco.html em Ourubro de 2002.

[Kahn, 1995] KAHN, R. e WILENSKY R. *A Framework for*

*Distributed Digital Object Services***.** Corporation for National Research Initiatives. Acessível por: http://www.cnri.reston.va.us/cstr/arch/k-w.html em Outubro de 2002.

[Krasner, 1988] KRASNER, G. e STEPHEN, T. P. A cookbook for using the model-view-controller user interface paradigm in Smaltalk-80. *Journal of Object-Oriented Programming* 1(3):26-49, Agosto/Setembro 1988

[Larman, 1998] LARMAN, C., *Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design*. ISBN 0137488807, Prentice-Hall, 1998.

[Lesk, 1997] LESK, M. *Practical Digital Libraries*. Morgan Kaufmann Publishers, 1997

[MARC, 2002] Library of Congress Network Development and the MARC Standards Office. Acessível por: http://www.loc.gov/marc/ em outubro de 2002.

[Marchionini, 1998] MARCHINIONINI, G. Research and Development in Digital Libraries Acessível por: http://ils.unc.edu/~march/ em Fevereiro de 2002.

[MeSH, 2002] Medical Subject Headings. Acessível por: http://www.nlm.nih.gov/mesh/meshhome.html em Outrubro de 2002.

[Meyer, 1992] MEYER, B. *Applying 'Design by Contract'* IEEE Computer, 25(10): 40-51, Outubro 1992.

[Microsoft.Net, 2002] Microsoft.Net Acessível por: http://msdn.microsoft.com/netframework/productinfo/overview/default.asp em Outubro de 2002.

[MIME, 1996] Multipurpose Internet Mail Extensions (Mime) Part Five: Conformance Criteria and Examples. Acessível por: http://www.ietf.org/rfc/rfc2049.txt?number=2049 em Outubro de 2002.

[MODELS, 2002] MODELS – Moving to Distributed Environments for Library Services. Acessível por: http://www.ukoln.ac.uk/dlis/models/ em Outubro de 2002.

[MPEG7, 2002] Internacional Organisation for Standardisation ISO/IEC JTC1/SC29/WG11 *Coding of Moving and áudio*. Acessível por: http://ipsi.fhg.de/delite/Projects/MPEG7/Documents/W2861.htm em outubro de 2002

[NCSTRL, 2002] Networked Computer Science Technical Reference Library, Acessível por: http://www.ncstrl.org/ em Outubro de 2002

[NTDL, 2002] New Technologies Distributed Learning Project. Acessível por: http://www.ntdl.org em outubro de 2002.

[NTRS, 2002] NASA Technical Report Server. Acessível por: http://library.gsfc.nasa.gov/Databases/NTRS/AboutNTRS.htm em outrubro de 2002.

[NZDL, 2002] The New Zealand Digital Library Project Acessível por: http://www.cs.waikato.ac.nz/~nzdl/ em outurbro de 2002

[OMG, 2000] OMG. *Unified Modeling Language Specification version 1.3*. Mar. 2000. Acessível por: http://cgi.omg.org/cgi-bin/doc?formal/00-03-01.pdf.gz em Outubro de 2002.

[OMG-UML, 2002] Object Management Group, Unified Modeling Language (UML Version 1.3). Acessível por: http://www.omg.org/technology/documents/formal/uml.htm em Outubro de 2002

[Oracle, 2002] Oracle. Acessível por: http://www.oracle.com em Dezembro de 2002.

[Pree, 1995] PREE, W. Design Patterns for Object-Oriented Software Development. Addison-Wesley, 1995

[Reddy, 1999] REDDY, R., WTEC-Digital Information Organization in Japan. International Technology Research Institute, World Technology, (WTEC) Division, *WTEC Panel Report*, 1999.

[Renardus, 2002] Projeto Renardus. Acessível por: http://www.renardus.org em Outubro de 2002

[ROADS, 2002] Resource Organization and Discovery in Subject-Based Services (ROADS). Acessível por: http://www.ilrt.bristol.ac.uk/roads/ em Outubro de 2002

[Roberts, 1997] ROBERTS, D. e JOHNSON, R. *Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks*. 1997. Acessível por: http://st-www.cs.uiuc.edu/users/droberts/evolve.html em Outubro de 2002.

[Roman, 2001] ROMAN, E., *Mastering Enterprise JavaBeans and the Java 2 Plataform, Enterpsise Edition*. ISBN: 0471332291, John Wiley & Sons, 2001.

[Rumbaugh, 1999] RUMBAUGH, J., BOOCH, G. e JACOBSON, I. *The Unified Modeling Language Reference Manual.* ISBN: 020130998X, Addison-Wesley, 1999.

[Rumbaugh, 1994] RUMBAUGH, J. *Getting Started: Using Use Cases to Capture Requirements*. JOOP, pages 8-23, 1994

[RUP, 2002] Rational Unified Process. Acessível por: http://www.rational.com/products/rup/ em Outubro de 2002

[Salton, 1988] SALTON, G. *Automatic Text Processing, The Transformation, Analisys, and Retrieval of Information by Computer*. Addison-Wesley, 1988

[Schaffer, 1994] SCHAFFER, W. e GARLAN, D. *Software Architecture: Perspectives on an emerging discipline.* Prentice-Hall, 1994

[Schäuble, 1998] SCHÄUBLE, P. e SMEATON, A. F. An International Research Agenda for Digital Libraries. *Summary Report of the Series of Joint NSF-EU Working Groups*

on Future Directions for Digital Libraries Research, 1998

[Schneider, 1998] SCHNEIDER, G. e WINTERS, J.P., *Applying Use Cases: A Practical Guide,* Addison-Wesley, 1998.

[Sheth, 1998] SHETH, A., KLAS, W. e BOLL, S. Overview on Using Metadata to Manage Multimedia Data. *Multimedia Data Management – Using Metadata to Integrate and Apply Digital Media*. McGraw-Hill, 1998

[Singh, 2002] SINGH, I., STEARNS, B. e JONHNSON, M. Designing Enterprise Application with the J2EE Plataform, Second Edition, Addison-Wesley, ISBN 0201787903, 2002

[Struts, 2002] The Struts Web Application Frameworks. Acessível por: http://jakarta.apache.org/struts/ em Outubro de 2002.

[Sun Microsystems, 2002] SUN MICROSYSTEMS INC. *Enterprise JavaBeans Specification v2.0.* Out. 2002. Disponível em http://java.sun.com/products/ejb.

[Sun Microsystems, 2002b] SUN MICROSYSTEMS INC. *Simplified Guide to the Java 2 Plataform Edition* Out. 2002. Disponível em http://java.sun.com/products/ejb.

[Sun Microsystems, 2002c] SUN MICROSYSTEMS INC. *Sample Application Design and Implementation* Out. 2002. Acessível por: http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/sample-app/sample-

app1.3.1.html em Outubro de 2002

[Sun Microsystems, 2002d] SUN MICROSYSTEMS INC. *Java Server Face Technology* Out. 2002. Acessível por: http://java.sun.com/j2ee/javaserverfaces/ em Outubro de 2002

[Sun Microsystems, 2002e] SUN MICROSYSTEMS INC. *Java Blueprints Enterprise: Java Pet Store Sample Application.* Out. 2002. Acessível por: http://java.sun.com/j2ee/javaserverfaces/ em Outubro de 2002

[Takagiwa, 2001] TAKAGIWA, O. et al. Programming J2EE APIs with Websphere Advanced, IBM Acessível por: http://ibm.com/redbooks em outubro de 2002

[Tennant, 2000] TENNANT, R. Open Archives: A Key Convergence em *Library Journal Digital*, 2000. Acessível por: http://www.libraryjournal.com/ em 12\03\2001

[Thomas, 1998] THOMAS, A. Java 2 Plataform, Enterprise JavaBean Technology. Server Component Model for the Java plataform, 1998

[Thomas, 1999] THOMAS, A. Java 2 Plataform, Enterprise Edition – Ensuring Consistency, Portability and interoperability, Patricia Seybold Group, EUA, 1999

[Unesco, 2002] Unesco Free Software Portal. Acessível por: http://www.unesco.org/webworld/portal_freesoft/Software/Digital_Library/ em Outubro de 2002.

[Voisard, 1998] VOISARD, A. e GUNTHER, O. Metadata in Geographic and Environmental Data Management. *Multimedia Data Management – Using Metadata to Integrate and Apply Digital Media*. McGraw-Hill, 1998

[Vries, 1998] VRIES, A. P. e BLANKEN, H. M. Database Technology and the Management of Multimedia Data in Mirror. *Proceddings of SPIE The International Society of Optical Engeeniring*, 1998

[W3C, 2001] World Wide Web Consortium. Voice Extensible Markup Language (VoiceXML) Version 2.0. Acessível por: http://www.w3.org/TR/2001/WD-voicexml20-20011023/ em Outubro de 2002. .

[W3C, 2001b] World Wide Web Consortium. Scalable Vector Graphics (SVG) Version 1.0. Acessível por: http://www.w3.org/TR/SVG/ em Outubro de 2002. .

[W3C, 2002] World Wide Web Consortium. XML Signature WG. Acessível por: http://www.w3.org/Signature/ em Outubro de 2002. .

[XML, 2002] W3C Architecture Domain: Extensible Markup Language. Acessível por: http://www.w3.org/XML/ em Outubro de 2002. .

[Yu, 1994] YU, C. e MENG W. Progress in Database Search Strategies**.** *IEEE Software,* 11-19 Maio 1994

Anexo A

Breve Análise de projetos existentes

Aqui serão analisados alguns projetos de bibliotecas digitais, desenvolvidos em diversas partes do mundo. A maioria destes projetos foi iniciada já há alguns anos e continuam evoluindo até os dias atuais. O maior objetivo destes é contribuir com o avanço da área de bibliotecas digitais, lançando novas questões e propondo novas soluções.

A análise de cada projeto foi empreendida visando maior familiarização e vivência com os problemas e idiossincrasias do domínio em estudo. Através das pesquisas e das discussões levantadas na avaliação de cada projeto, foi possível a identificação de padrões estruturais que constituem uma aplicação de biblioteca digital típica. Além do mais, foram observadas as características marcantes de cada sistema, o que diferencia uns dos outros e seus pontos negativos e positivos.

A partir da análise comparativa dos projetos existentes, juntamente com a análise de domínio, realizada com auxílio de especialistas, foi definido o projeto inicial do framework para bibliotecas digitais proposto neste trabalho.

Na primeira seção será apresentado o projeto *New Zealand Digital Library,* desenvolvido pela universidade de Waikato na Nova Zelândia. A segunda seção discute o projeto *Alexandria Digital Library,* um projeto em desenvolvimento desde o ano de 1996 pela Universidade de Santa Bárbara, na Califórnia. O projeto europeu *Renardus,* desenvolvido por um conjunto de instituições será discutido na terceira seção. E finalmente o *Greenstone,* software para a construção de bibliotecas digitais criado, também pela universidade de Waikato, Nova Zelândia, será analisado na quarta seção.

1. New Zealand Digital Library

A *New Zealand Digital Library* (NZDL) foi desenvolvida por membros do departamento de Ciências da Computação da universidade de Waikato, Nova Zelândia. É uma biblioteca digital de acesso público que disponibiliza vários gigabytes de informação através da Internet. O objetivo do projeto é explorar as potencialidades da tecnologia de bibliotecas digitais através do uso de coleções independentes e bastantes distintas [NZDL, 2002].

Tipo de dados: Atualmente, a biblioteca conta com duas coleções: um acervo de músicas com aproximadamente dez mil itens e outra com documentos do tipo texto correspondentes às publicações na área de ciência da computação.

Modelo de Metadados: No projeto da NZDL não há a definição de um padrão de metadados para a descrição dos recursos armazenados. As técnicas utilizadas para a busca e recuperação dos itens são a recuperação textual e a recuperação baseada em melodias. Todavia, as coleções podem ser descritas em termos de metadados. Os atributos de metadados utilizados nesta atividade não são parte de nenhum padrão reconhecido. Eles são, tão somente, uma definição realizada pelos projetistas deste sistema.

Arquitetura da Solução: O maior objetivo da concepção da arquitetura desta biblioteca digital é acomodar diversas coleções de informações com índices distintos de acordo com o tipo de dado armazenado. Para tanto, esta arquitetura é bastante flexível a fim de permitir a composição de mecanismos de busca diferentes, o que se reflete em primeiro plano nas interfaces com o usuário e em segundo plano com os índices que viabilizam as buscas. Esta arquitetura é baseada em camadas. A camada de interface comunica-se com a camada responsável pelas operações de busca, através de um protocolo de comunicação bem-definido. Na camada intermediária, encontram-se os mecanismos de busca e de construção de índices e, por fim, os índices.

Serviços Oferecidos: Como parte da própria solução, a NZDL oferece serviço para a criação dos índices da coleção.

Armazenamento: As coleções são armazenadas em máquinas distribuídas na rede. Os clientes tomam conhecimento de cada uma delas através de uma lista de coleções armazenadas na máquina local. Os recursos das coleções não são armazenados em bancos de dados convencionais.

Interface: A arquitetura desta solução permite que as interfaces sejam bem distintas dos outros módulos do sistema. Desta forma, a interface com o usuário que hoje é realizada via Web poderia ser também baseada em janelas ou mesmo em linha de comando (para fins de depuração).

2. Alexandria Digital Library

A partir de 1996, a agência de fomento a pesquisa americana NSF – *National Science Foundation* – vem financiando pesquisas na área de bibliotecas digitais. O grande projeto da área, que hoje se encontra em sua segunda fase é o *Digital Library Initiative* [DLI1, 1994]. Neste projeto, diversas instituições de nível superior uniram-se na pesquisa e no desenvolvimento tecnológico para a área de bibliotecas digitais.

A universidade da Califórnia, em Santa Bárbara, desenvolveu a *Alexandria Digital Library* – ADL. Este projeto, que na segunda fase passou a chamar-se ADEPT (*Alexandria Digital Earth*) visa ao desenvolvimento e aplicação de inovações na área de bibliotecas digitais geográficas.

Tipo de dados: A *Alexandria Digital Library* tem um nicho de aplicação bem definido: Oferecer informações geo-referenciadas aos seus usuários. Desta forma, os dados armazenados são: imagens de satélite, mapas, vídeos, fotografias aéreas, dentre outros.

Modelo de Metadados: Recursos geo-espaciais são caros para serem adquiridos e de uso extremamente especializado. Os padrões de metadados criados para descrevê-los são bastante complexos e abrangentes. A *Alexandria Digital Library* suporta o uso tanto do CSGDM, desenvolvido pelo FGDC norte-americano – *Federal Geographic Committee*; quanto do USMARC, bastante adotado nas bibliotecas de propósito geral. A evolução do projeto permitiu ainda que um outro padrão, mais simples, inspirado no Dublin Core, fosse adotado. Esta decisão visou a oferecer mais agilidade às pesquisas.

Arquitetura da Solução: A arquitetura da *Alexandria Digital Library* é constituída por três camadas: Cliente, o *middleware* e o servidor.Os clientes são aplicações Web que se comunicam com o *middleware* usando XML. Na camada intermediária encontram-se as regras de negócio como: controle de acesso a coleções e serviços, mecanismos de consulta e gerência de resultados. No lado servidor, encontra-se o banco de dados que armazenam os recursos e outros módulos gerenciais para auxílio à pesquisa. Uma visão detalhada da arquitetura atual deste projeto pode ser vista na Figura 54.

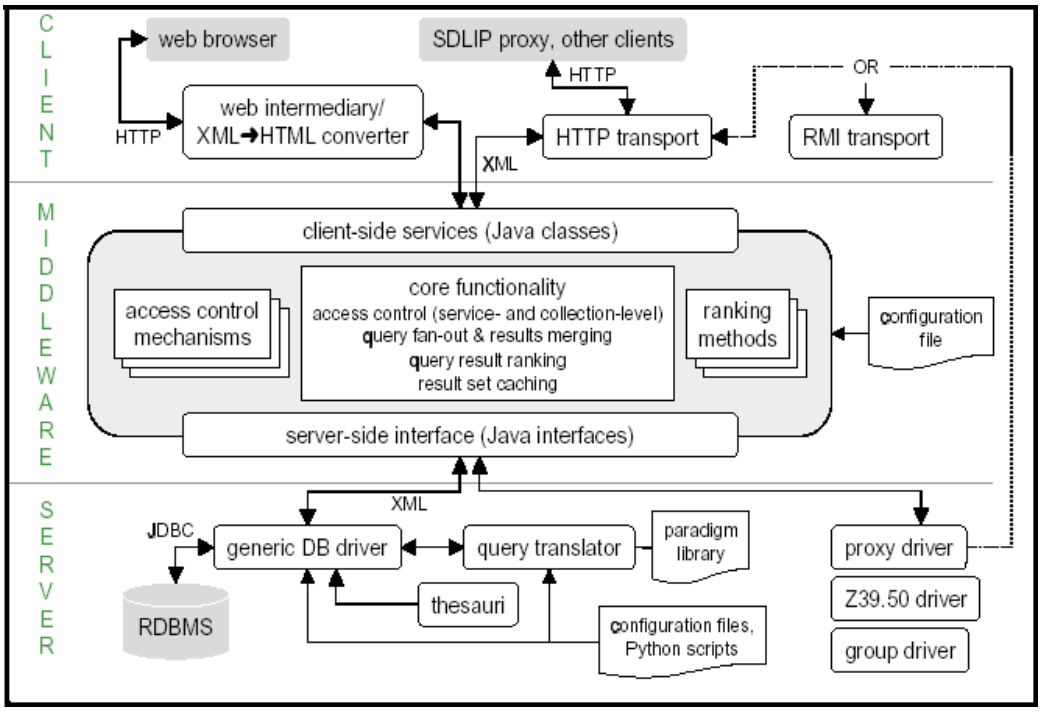


Figura 54 - Arquitetura da ADEPT

Serviços oferecidos: A ADL oferece ao usuário acesso transparente a coleções distribuídas (DDL – *Distributed Digital Libraries*). Além do mais o serviço para pesquisa dos dados geo-referenciados oferece grandes facilidades e poder ao usuário. Na segunda fase do projeto, quando passou a denominar-se ADEPT, o serviço de pesquisa incorporou à sua interface um mapa mundi e a possibilidade de utilização de imagens tridimensionais.

Armazenamento: O armazenamento dos recursos é realizado em bancos de dados convencionais (bancos de dados relacionais). As consultas em cima destes dados passam previamente por um tradutor de consultas.

Interface: Na primeira fase do projeto, a interface oferecida para a ADL era uma aplicação cliente Java. Hoje a ADL pode ser acessada por browsers através da Internet em qualquer lugar do mundo. A interface oferece duas modalidades de pesquisa: a primeira é através de um mapa o mundo e a segunda através de atributos. No primeiro caso, o usuário seleciona a área que deseja consultar no mapa dividido em retângulos (*bounding boxes*) e pode dar *zoom* para aproximar ou afastar a imagem. Na pesquisa em cima de atributos, o usuário pode especificar uma consulta baseada em: uma coleção específica, coordenadas

geográficas (latitude e longitude), operadores de pesquisa geo-referenciais, período de tempo, formato do recurso, mídia dentre outros.

3. Renardus

O projeto *Renardus* é uma iniciativa conjunta de várias instituições européias financiado pela IST – Sociedade de Tecnologia da Informação, da União Européia [Renardus, 2002]. A maioria dos parceiros do *Renardus* já havia desenvolvido projetos na área de bibliotecas digitais, como por exemplo: ROADS [ROADS, 2002], DESIRE [DESIRE, 2002], EELS [EELS, 2002].

A missão desta iniciativa foi criar um projeto piloto que viabilizasse o acesso a recursos de qualidade na Web através de um único ponto. O público alvo é formado por professores, estudantes e pesquisadores de nível universitário na Europa.

O *Renardus* oferece a infraestrutura e os seus parceiros (os *subject gateways*) disponibilizam a informação. Para tanto, o projeto investe na criação de um arcabouço tecnológico adequado e na definição das questões de negócio desde o princípio. Desta forma, foi desenvolvido um serviço viável.

Tipo de Dados: Os dados disponibilizados através do *Renardus* são provenientes de seus parceiros. Dentre os vários tipos de recursos oferecidos têm-se: imagem, som e texto.

Modelo de Metadados: O projeto trabalha com um esquema de metadados baseado na semântica dos campos do padrão *Dublin Core.* Todos os parceiros do projeto utilizam o mesmo padrão. Os elementos descritivos são: título, criador, descrição, identificador, assunto, editor, idioma, tipo, além de outros elementos específicos do *Renardus.*

Arquitetura da solução: O modelo arquitetural do projeto foi desenvolvido a partir de uma investigação inicial dos requisitos do usuário e da tecnologia disponível de forma a suprir tais requisitos. A arquitetura do *Renardus* é uma extensão da MIA – *Models Information Architecture* [MODELS, 2002], baseada em camadas lógicas de acordo com a Figura 55. No *Renardus* as coleções são distribuídas e disponibilizadas pelos parceiros do projeto de forma transparente ao usuário.

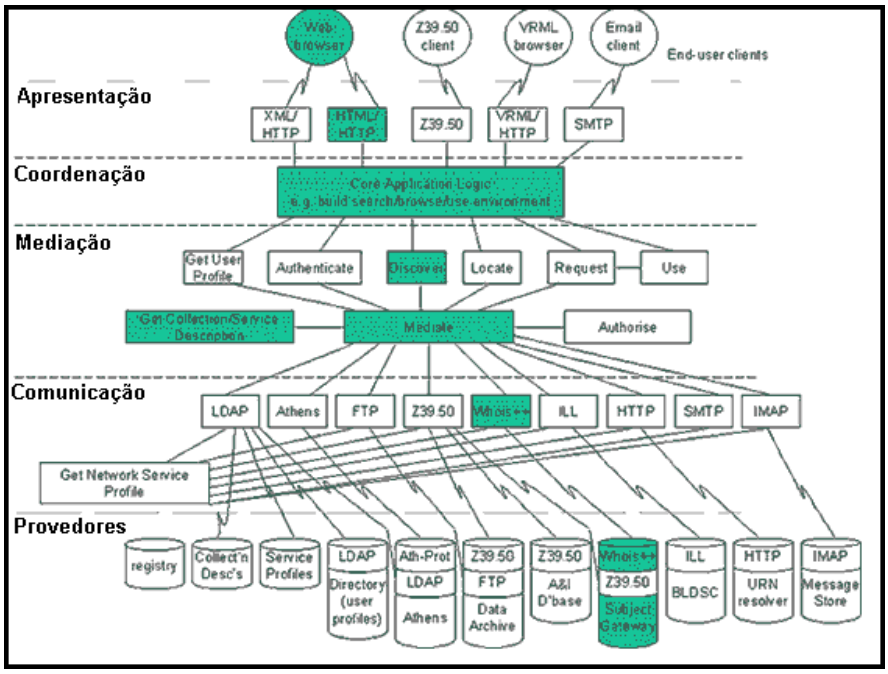


Figura 55 - Arquitetura do Renardus

Serviços oferecidos: São oferecidos os serviços de busca e navegação nas coleções. O projeto prevê, ainda, o serviço de metadados compartilhados. Neste serviço, os parceiros disponibilizariam os metadados de suas coleções para o armazenamento em um catálogo único e acessível a todos.

Armazenamento: O armazenamento dos dados de cada coleção é feito por cada parceiro (*subject gateways*) de forma independente, desde que se adequem às definições feitas no *Renardus*.

Interface: A interface da aplicação é Web e está acessível através da URL http://www.renardus.org.

4. Greenstone

O software para a construção de bibliotecas digitais *Greenstone* foi produzido pelo projeto *New Zealand Digital Library* da universidade de Waikato, na Nova Zelândia e distribuído em parceria com a UNESCO [Unesco, 2002].

A missão deste projeto é permitir que os usuários, principalmente universidades, bibliotecas e outros órgãos públicos construam as suas próprias bibliotecas

digitais. O objetivo do software é incentivar a criação deste tipo de sistema que viabilizem a troca e o intercâmbio de informações de forma pública.

Tipo de Dados: Os tipos de dados armazenados na *Greenstone* podem ser texto, áudio, imagem ou vídeo.

Modelo de Metadados: O uso de metadados não é obrigatório neste software. Entretanto, se eles existirem podem ser criados índices de recuperação textual também para os atributos de metadados. O padrão de metadados utilizado na *Greenstone* é característico do próprio projeto e não segue nenhum padrão pré-definido.

Arquitetura da Solução: As soluções construídas com o *Greenstone* podem funcionar tanto *stand alone* quanto para a Web. A arquitetura é basicamente cliente/servidor, onde os clientes são browsers e no servidor encontram-se tanto os dados quanto o negócio da aplicação.

Serviços oferecidos: A *Greenstone* oferece serviço de administração de coleções onde são apresentadas informações sobre a instalação e criação de coleções. E ainda, capacita outros usuários para a realização destas tarefas. A criação de índices é, também, outro serviço nativo da *Greenstone*. Uma série de *plugins,* que são distribuídos juntamente com o *Greenstone,* pode ser utilizada para a criação de coleções. Neste ponto do projeto, já foram criados *plugins* para os seguintes tipos de documentos: *pdf, email, html*, dentre outros.

Armazenamento: Os recursos disponibilizados por uma biblioteca construída com este software não são armazenados em bancos de dados convencionais. São acessíveis via URN.

Interface: A interface de acesso à aplicação já é pré-definida e pode ser facilmente personalizada. Ela pode ser acessada a partir de browsers tanto em clientes Windows quanto em Unix.

Anexo B

Modelo para descrição de Casos de Uso

O modelo adotado para a descrição dos casos de uso do Diliframe é apresentado na Tabela 21. As linhas em destaque indicam os títulos de cada etapa, informando, ainda, se são elas opcionais ou não; as linhas subseqüentes apresentam uma breve descrição do propósito de cada uma.

1. Descrição de Caso de Uso
Título do caso de uso.
2. Finalidade
Objetivo a ser atingido ou ação a ser desempenhada com a realização do caso de uso.
3. Atores
Lista de atores envolvidos no caso de uso.
4. Pré-condições (opcional)
Condições que devem ser satisfeitas para que o caso de uso tenha início.
5. Fluxo Principal
Seqüência típica de eventos na interação entre atores e sistema.
6. Fluxos Alternativos (opcional)
Qualquer decisão ou variação alternativa ocorrida na seqüência do fluxo principal.
7. Exceções (opcional)

Possíveis situações de erro ocorridas durante o fluxo principal.
8. Pós-condições (opcional)
Resultados obtidos após a realização do caso de uso.
9. Testes sugeridos (opcional)
Sugestão de ações a serem realizadas na fase de testes.

Tabela 21 – Modelo de Caso de Uso

A seguir, a semântica e a forma de apresentação do modelo proposto serão discutidas com mais detalhes:

- **Descrição de casos de uso**

Nesta etapa, é identificado o título do caso de uso. É importante que o nome seja único e sugestivo a fim de informar, prontamente, o que será descrito pelo caso de uso. Recomenda-se que sejam usados verbos no infinitivo no título do caso de uso.

- **Finalidade**

A etapa relativa à finalidade do caso de uso deve descrever o objetivo de negócio a ser atingido com a realização do mesmo. De forma simples e direta, informar ao leitor qual o cenário de interação com o sistema que este caso de uso pretende descrever.

- **Atores**

Nesta etapa, são listados os atores envolvidos no caso de uso. Recomenda-se grafar o nome dos atores com suas iniciais maiúsculas para que seja facilitada a sua identificação no texto [Larman, 1998]. Opcionalmente os atores podem ser classificados como primários e secundários [Coleman, 1998]. Um ator primário atinge determinado objetivo contando com a assistência do sistema. Já um ator secundário atua na assistência do sistema para que este cumpra alguma tarefa.

- **Pré-condições**

Diz-se das pré-condições de um caso de uso, os requisitos para que este possa ter início. Estes pré-requisitos têm origens bastante distintas e podem não depender da ação dos atores do caso de uso em questão. Uma lista de pré-condições pode conter, por

exemplo: acionamento de uma opção de menu, finalização de uma certa transação, finalização de determinado período de tempo, alimentação de alguns dados de entrada, etc.

- **Fluxo Principal**

Nesta etapa, é relatada a interação dos atores com o sistema. Todos os passos necessários a fim de que o objetivo seja atingido são listados na ordem em que acontecem: a ação do ator e a resposta do sistema. Havendo a necessidade de iteração dos passos o autor deve fazê-lo em linguagem natural.

Cada passo do fluxo deve ser identificado por um número seqüencial. Esta identificação pode ser usada nos fluxos alternativos para o restabelecimento da seqüência de ações. Em seguida, vem a identificação do evento que pode ser uma ação do ator ou uma resposta do sistema. Os passos podem ou não ser acompanhados de exceções ou passos alternativos. Passos alternativos serão descritos na etapa de fluxo secundário. As exceções serão aprofundadas na sua seção correspondente. O caso de uso é finalizado quando não há mais passos a serem executados no fluxo principal ou quando um passo direciona o fluxo para outro caso de uso.

O formato de apresentação de um item do fluxo principal é o seguinte:

P = P*.<número identificador><ação do ator | resposta do sistema ><E>*|<PA>**

onde:

PA = "Passo Alternativo"

P = "Passo"

E = "Exceção"

- **Fluxos Alternativos**

A etapa de fluxos alternativos lista a seqüência de passos alternativos que podem ser gerados para cada item do fluxo principal. Um passo alternativo é criado quando existe uma tomada de decisão no fluxo principal. Os passos alternativos têm o mesmo formato dos passos no fluxo principal, acrescidos da opção de informar para qual passo se deseja retornar a fim de dar continuidade ao processo.

Não obstante, é recomendado não gerar passos alternativos de passos alternativos. Isto comprometeria o bom entendimento do caso de uso. Além do mais, o objetivo desta etapa não é esgotar todas as possibilidades de interação do sistema, mas, tão

somente, realizar uma análise do comportamento do sistema e de seus requisitos de forma mais detalhada.

O formato de apresentação de um item do fluxo alternativo é o seguinte:

PA = PA.<número identificador>

<número sequencial><ação do ator | resposta do sistema > <E>|<PA>*|<P>**

onde: PA = "Passo Alternativo"

P = "Passo"

E = "Exceção"

- **Exceções**

Nesta etapa, são listadas as exceções que podem ocorrer em cada um dos passos dos fluxos principal e alternativo. As exceções são caracterizadas por comportamentos diferentes dos esperados. Isto pode ter origem a partir de dados de entrada, ações do ator ou sub-sistemas com funcionamento anormal.

Prever exceções, na fase de análise de casos de uso, é interessante e muito útil. As exceções, identificadas nesta fase do processo de desenvolvimento, não estão relacionadas à estrutura interna de implementação, e sim ao domínio da aplicação. Isto sugere que o projetista deve reservar-lhes uma atenção especial nas fases subseqüentes do processo.

Na descrição de uma exceção, deve constar a situação que a desencadeia. Em seguida, é listada uma seqüência de passos de interação podendo resultar em uma das seguintes situações: retorno ao fluxo alternativo, retorno ao fluxo principal, ou ainda a finalização do caso de uso.

O formato de apresentação de um item de exceção é o seguinte:

E = E.<número identificador><descrição da situação de exceção>

<número sequencial><ação do ator | resposta do sistema > <PA>|<P>**

onde: PA = "Passo Alternativo"

P = "Passo"

E = "Exceção"

- **Pós-condições**

Nesta etapa, são apresentadas as atividades cumpridas ao final do caso de uso. É retratado o resultado final obtido do processo de interação do ator com o sistema. Esta etapa é considerada opcional, pois, normalmente, os casos de uso descrevem ações pontuais cujos resultados esperados já foram descritos na etapa relativa à finalidade. Recomenda-se, então, listar pós-condições de um caso de uso, apenas quando isto não for redundante.

- **Testes Sugeridos**

Nesta etapa, são listadas algumas situações críticas que devem ser consideradas na fase de testes. Assim como a etapa de exceções, a etapa de testes sugeridos não tem pretensão de esgotar todas as possibilidades de testes a serem empreendidos.

O objetivo desta etapa é antecipar comportamentos que podem originar erros no sistema. A previsão de testes, nesta fase do processo de desenvolvimento, denota a preocupação com a qualidade do sistema em desenvolvimento. Os testes sugeridos nesta etapa da descrição dos casos de uso podem ser utilizados como base na elaboração dos planos de teste da aplicação.

Anexo C

O padrão de Metadados Dublin Core

Aqui será discutido o padrão de metadados adotado no projeto do Diliframe para a descrição de seus recursos. Serão abordados o histórico, as particularidades e os benefícios obtidos pela escolha deste padrão.

O que é o Dublin Core?

O Dublin Core (DC) é um conjunto de elementos simples e efetivos para a descrição de diversos tipos de recursos digitais [DC, 2002]. Este padrão conta com um conjunto de quinze elementos cuja semântica foi definida por um corpo internacional e multidisciplinar de profissionais com o objetivo de estabelecer um padrão de referência. Cada um destes elementos é usado para descrever uma característica do recurso. As características positivas inerentes a este padrão estão sumariadas na Tabela 22 [Hillmann, 2001].

Simplicidade de criação e manutenção.
O conjunto de elementos do DC foi mantido pequeno e simples a fim de que fosse possível a criação de registro de metadados por não-especialistas. Ao mesmo tempo em que mantinha o conjunto completo e coerente para viabilizar a sua utilização por mecanismos de busca.
Semântica fácil de ser entendida
O DC define um conjunto de elementos cuja semântica é facilmente entendida e aceita. O objetivo é condensar a descrição de recursos em cima destes elementos. E, assim, permitir que pessoas das mais diversas áreas de atuação os reconheçam e realizem as suas consultas, eficientemente, em cima destes termos. Quer seja, por exemplo, para um pesquisador buscando um artigo científico, quer seja para um estudante em idade escolar; o elemento

"título" deverá ser entendido da mesma forma.
Escopo internacional
Originalmente, o DC foi criado em inglês, mas hoje já há versões em diversos idiomas tais como: alemão, grego, japonês, além de outros, inclusive o português.
Capacidade de extensão
A versatilidade do DC é garantida por sua capacidade de extensão. Os desenvolvedores deste padrão estabeleceram mecanismos para que outros padrões de metadados, definidos para um nicho de recursos bem definidos, pudessem ser acoplados ao DC a fim de prover um registro de metadados mais completo. Sendo assim, as mais diversas aplicações podem usar DC via descrição de seus recursos optando por uma versão simples ou estendida.

Tabela 22 - Características do padrão Dublin Core

Princípios do Dublin Core e seus elementos

Todos os elementos do Dublin Core são opcionais e podem ser repetidos de acordo com a necessidade. Por exemplo, se um artigo foi escrito por três autores o elemento "*creator*" deve ser repetido três vezes. Cada elemento conta, ainda, com um conjunto limitado de qualificadores que podem ser referentes à sua semântica ou a sua sintaxe. Diante disso classificam-se os qualificadores em dois grupos [DC, 2001]:

- Refinamento de elementos: Os qualificadores usados para refinamento têm como objetivo fazer com que o valor do elemento seja mais específico e focado. Um elemento refinado tem o mesmo significado de seu elemento raiz (sem qualificação) só que aplicado num contexto mais restrito.
- Esquema de codificação: Estes qualificadores identificam esquemas de acordo com os quais os valores dos elementos devem ser expressos. Estas convenções podem agregar valor ao significado do elemento.Os esquemas de codificação dizem respeito a vocabulários controlados ou notação para descrição formal.

A Tabela 23, lista os quinze elementos do Dublin Core agrupados segundo o conceito ao qual os elementos se referem. Alguns elementos dizem respeito ao conteúdo do

recurso que descrevem; outros se referem a questões relativas à propriedade intelectual do recurso; já os demais caracterizam uma instância do recurso em particular. Nesta dissertação e no projeto do Diliframe optou-se por utilizar o conjunto de elementos do Dublin Core em seu idioma original. Não obstante, já exista tradução oficial destes elementos para o português formalizada e mantida por José Luís Borbinha da biblioteca Nacional de Portugal [Borbinha, 2000].

Conteúdo	Propriedade intelectual	Instância
Coverage • Description • Type • Relation	Contributor • Creator • Publisher • Rights	Date • Format • Identifier • Language

Tabela 23 - Elementos do Dublin Core

A seguir, cada elemento Dublin Core será descrito e algumas considerações sobre ele serão tecidas de acordo com a especificação em sua versão 1.1 [DC, 2002].

Title
È o nome dado ao recurso. Geralmente, o título é o nome pelo qual o recurso é formalmente conhecido. Este elemento admite um qualificador do tipo "refinamento de elemento" e não admite qualificador do tipo "esquema de codificação". **Qualificador:** *Alternative*: Refere-se à um título que pode ser substituído pelo principal. Normalmente é utilizado para conter abreviações ou traduções dos títulos.

Creator
Representa a entidade responsável por fazer o conteúdo do recurso. Este elemento corresponde a uma pessoa, organização ou serviço. Este elemento não admite qualificadores.

Subject

Representa o assunto retratado no conteúdo do recurso. Normalmente, será expresso em termos de palavras chave, frases chave ou códigos de classificação que descrevem o tópico do recurso. Este elemento admite qualificadores do tipo "esquema de codificação" e não admite qualificador do tipo "refinamento de elemento".

Qualificadores:

LCSH – *Library of Congress Subject Headings* [LCSH, 2001], MeSH – Medical *Subject Headings* [MeSH, 2002], DDC – *Dewey Decimal Classification* [DDC, 2002], LCC – *Library of Congress Classification* [LCC, 2002].

Description

Representa um sumário do conteúdo do recurso. Pode, por exemplo, ser um *abstract,* tabela de conteúdos, dentre outros. Este elemento admite qualificadores do tipo "refinamento de elemento" e não admite qualificador do tipo "esquema de codificação".

Qualificadores:

Table of contents: Uma lista de elementos com os tópicos do conteúdo.

Abstract: Um resumo com o conteúdo do recurso.

Publisher

Representa uma entidade responsável por tornar o conteúdo do recurso disponível. Exemplos deste são: uma pessoa, uma organização ou serviço. Este elemento não admite qualificadores.

Contributor

Representa uma entidade responsável por realizar contribuições ao conteúdo do recurso. Exemplos deste são: uma pessoa, uma organização ou serviço. Este elemento não admite qualificadores.

Date

Representa uma data associada a um evento no ciclo de vida do recurso. Geralmente, este elemento pode ser associado à data da criação ou da disponibilização do recurso.

Recomenda-se utilizar um formato padrão para o valor deste elemento, como por exemplo, o definido pela ISSO 8601: "AAAA-MM-DD". Este elemento admite qualificadores do tipo "refinamento de elemento" e também admite qualificadores do tipo "esquema de codificação".

Qualificadores:

Refinamento de elemento –

Created: A data de criação do recurso.

Valid: Data ou intervalo de validade do recurso.

Available: Data ou intervalo que o recurso tornou-se ou tornar-se-á disponível.

Issued: Data da publicação formal do recurso.

Modified: Data na qual o recurso foi modificado.

Esquema de codificação –

DCMI *Period* [Cox, 2000] e W3C-DTF [DTF, 2002].

Type

Representa o tipo do recurso, a natureza do conteúdo do recurso. Este elemento admite termos que descrevem categorias, funções, gêneros, nível de organização de conteúdo. É recomendado utilizar um vocabulário controlado como o especificado pelo Dublin Core *Type* [DC, 2002]. Este elemento admite qualificador do tipo "esquema de codificação" e não admite qualificador do tipo "refinamento de elemento".

Qualificador:

DCMI *Type Vocabulary* [Cox, 2000].

Format

Representa o formato da manifestação física ou digital do recurso. Geralmente, este elemento representa o tipo da mídia ou o tamanho do recurso. Este elemento pode ser usado, também, para informar o software, hardware ou outro tipo de elemento necessário para apresentar ou operar o recurso. É recomendado que o valor deste elemento seja obtido a partir de um vocabulário controlado. Este elemento admite qualificador do tipo "esquema de codificação" e qualificadores do tipo "refinamento de elemento".

Qualificadores:

Refinamento de elemento –

Extent: Representa o tamanho ou duração do recurso.

Medium: O material ou meio físico que constitui o recurso.

Esquema de codificação –

IMT – *Internet Media Type*.

Identifier

Representa uma referência única para o recurso em determinado contexto. É recomendado identificar o recurso através de uma string ou número de acordo com um sistema de identificação formal. Exemplos de utilização deste elemento são URI, ISBN, dentre outros. Este elemento admite qualificador do tipo "esquema de codificação" e não admite qualificadores do tipo "refinamento de elemento".

Qualificador:

URI – *Uniform Resource Location*.

Source

Representa uma referência ao recurso ao qual o recurso que está sendo descrito é derivado. O recurso pode derivar deste completamente, ou, apenas em parte. É recomendado representar este elemento através de uma string ou número de acordo com um sistema de identificação formal. Este elemento admite qualificador do tipo "esquema de codificação" e não admite qualificadores do tipo "refinamento de elemento".

Qualificador:

URI – *Uniform Resource Location*.

Language

O idioma do conteúdo intelectual do recurso. O formato de representação deste elemento é definido pela RFC 1766. Exemplos de utilização deste são: "en-uk", para representar um recurso escrito em inglês britânico. Este elemento admite qualificadores do tipo "esquema de codificação" e não admite qualificadores do tipo "refinamento de elemento".

Qualificadores:

ISSO 639-2 [ISO, 2002] e a RFC 1766 [IETF, 2002].

Relation
Representa uma referência a um recurso que é relacionado ao recurso que está sendo descrito. É recomendado identificar este elemento através de uma string ou número de acordo com um sistema de identificação formal. Este elemento admite qualificadores do tipo "refinamento de elemento" e também admite qualificadores do tipo "esquema de codificação". **Qualificadores:** Refinamento de elemento – *Is version of*: O recurso é uma versão, uma adaptação ou edição do recurso referenciado. *Has version*: O recurso descrito tem uma versão, adaptação ou edição do recurso referenciado. *Is replaced by*: O recurso descrito é substituído pelo recurso referenciado. *Replaces*: O recurso descrito substitui o recurso referenciado. *Is required by*: O recurso descrito é necessário para o recurso referenciado seja fisicamente ou logicamente. *Requires*: O recurso descrito necessita do recurso referenciado seja fisicamente ou logicamente. *Is part of*: O recurso descrito é parte lógica ou física do recurso referenciado. *Has part*: O recurso descrito inclui o recurso referenciado seja fisicamente ou logicamente. *Is referenced by*: O recurso descrito é referenciado, citado, ou, de alguma outra forma, apontado pelo recurso referenciado. *References*: O recurso descrito referencia, cita, ou, de alguma outra forma, aponta o recurso referenciado. *Is format of*: O recurso descrito tem o mesmo conteúdo intelectual do recurso referenciado, mas, apresentado em outro formato. *Has format*: O recurso descrito é anterior ao recurso referenciado, que é essencialmente o mesmo conteúdo intelectual apresentado em outro formato. Esquema de codificação – URI – *Uniform Resource Location*.

Coverage

Representa a extensão ou escopo do conteúdo do recurso. Este elemento engloba localização espacial, período de tempo ou jurisdição. Este elemento admite qualificadores do tipo "refinamento de elemento" e também admite qualificadores do tipo "esquema de codificação". **Qualificadores:** Refinamento de elemento – *Spatial*: Características espaciais do conteúdo intelectual do recurso. *Temporal*: Características temporais do conteúdo intelectual do recurso. Esquema de codificação para o refinamento *Spatial* – DCMI *Period* [Cox, 2000], ISO 3166 [ISO, 2002], DCMI *Box* [Cox, 2000b], dentre outros. Esquema de codificação para o refinamento *Temporal* – DCMI *Period* [Cox, 2000], e W3C-DTF [DTF, 2002].

Rights
Representa informações sobre os direitos que atuam sobre o recurso. Geralmente, este elemento conterá informações sobre os direitos de propriedade intelectual, *copyright*, dentre outros. Este elemento não admite qualificadores.

Anexo D

Arquitetura de alto nível do Diliframe

Aqui será apresentada e discutida a arquitetura de alto nível, com ênfase na tecnologia, adotada no projeto Diliframe. Após análise superficial de outras plataformas para aplicações distribuídas, tais como Microsoft.Net [Microsoft.Net, 2002] e CORBA, definiu-se pela utilização da J2EE – *Java 2 Plataform Enterprise Edition* para a implementação do Diliframe. Esta escolha fundamentou-se, basicamente, nos seguintes pontos:

- Diversas fontes de pesquisadores e desenvolvedores na Web disponibilizam gratuitamente ferramentas necessárias para a implementação de aplicações que seguem esta plataforma, uma vez que a especificação J2EE da Sun é aberta.
- Uma vasta gama de material didático como: tutoriais, livros, códigos exemplo, listas de discussão, dentre outros; está acessível na Web para desenvolvedores interessados em investir na plataforma. Além do suporte que pode ser fornecido pela vasta comunidade Java de desenvolvedores.
- O interesse em investir em uma nova tecnologia para a implementação de bibliotecas digitais foi fator decisivo na escolha do J2EE. Além do mais, as características de seu modelo de aplicação favorecem sobremaneira a flexibilidade, adaptabilidade e modularização buscadas na concepção do framework.

Neste anexo será dada uma visão geral da plataforma J2EE, considerando que esta tecnologia é relativamente recente, enfatizando os principais conceitos de sua especificação. Ademais, serão abordadas características presentes nas aplicações construídas em cima desta plataforma, tais como, a divisão em camadas e os padrões de projetos nos quais se baseiam o seu modelo de programação. A importância deste estudo

está no embasamento teórico fornecido, visando um maior entendimento do mecanismo de extensão e adaptação do framework proposto.

Contexto Histórico

Os sistemas corporativos, que demandam uma computação extremamente robusta, sempre foram grande fonte de preocupação de empresas e desenvolvedores. Além do mais, com o advento da internet e o espaço aberto para o comércio eletrônico, estes sistemas atingiram um grau de importância ainda maior. As aplicações tornaram-se mais complexas demandando maiores exigências tais como [Thomas, 1999]:

- Escalabilidade: O número de usuários concorrentes para aplicações disponibilizadas na internet pode variar de dezenas a milhares. Dependendo do escopo da aplicação o desenvolvedor deve prever um alto grau de escalabilidade.
- Alta disponibilidade: As aplicações na Internet devem estar constantemente disponíveis, ao contrário dos antigos sistemas nos quais podia-se contar com o final do expediente para a realização de atividades de manutenção.
- Segurança: Aplicações corporativas na internet são uma porta para a empresa aberta em qualquer browser. É necessário valer-se de diversos mecanismos de segurança para garantir a privacidade dos dados dos usuários, quanto da própria empresa.

O modelo arquitetural exigido por este tipo de sistema não se esquadra mais no tradicional esquema cliente-servidor, no qual a lógica de aplicação está localizada na aplicação cliente e é feito acesso direto aos sistemas de informações da empresa – EIS (Enterprise Information Systems), Figura 56. O novo modelo exigido para as aplicações corporativas requer uma maior estratificação em camadas. Neste modelo, a camada intermediária comporta a lógica de negócio e ainda fornece diversos serviços, dentre eles: viabilização do acesso ao EIS, gerência de transações, segurança e integração com serviços de mensagens; Figura 57.

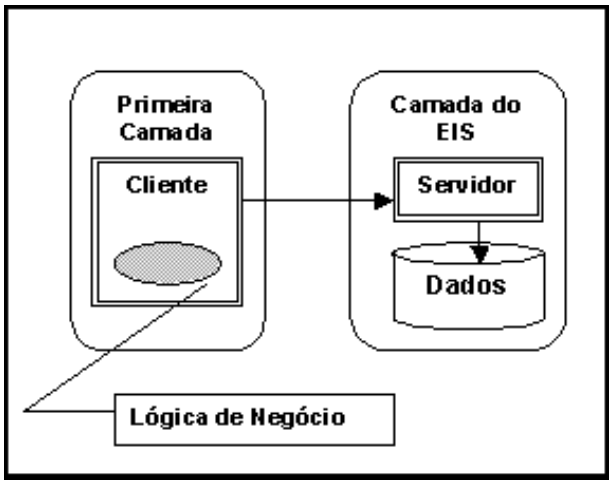


Figura 56 – Modelo de aplicação Cliente/Servidor

Todavia, o desenvolvimento de aplicações corporativas, distribuídas e em múltiplas camadas, tornava-se bastante complexo: cabia ao desenvolvedor a implementação do sistema, além da lógica de negócio, própria da aplicação, preocupar-se, também, com a implementação dos serviços da camada intermediária.

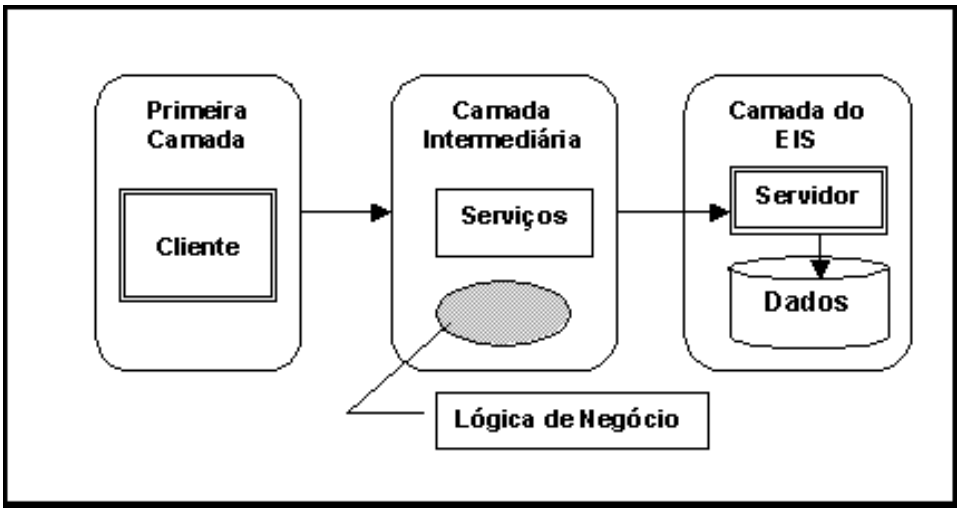


Figura 57 – Modelo de Aplicação Multi-Camadas

Para suprir a necessidade do desenvolvimento deste tipo de aplicações, a *Sun Microsystems* desenvolveu a plataforma Java para corporações. Os esforços para seu desenvolvimento tiveram início em 1997, quando a *Sun* lançou esta iniciativa e encorajou o desenvolvimento de um conjunto de APIs corporativas Java [Thomas, 1999]. Estas APIs forneceriam a infraestrutura necessária para a implementação dos serviços da camada

intermediária, cada uma implementando um tipo de serviço de infraestrutura. Este conjunto é considerado uma extensão da Java e está disponível na *Java Standard Extension*. A Tabela 24 lista as APIs Java para corporações e sua respectiva descrição [Takagiwa, 2001].

API	Descrição
EJB	*Enterprise Java Bean* é modelo para componentes do lado servidor que viabiliza portabilidade entre servidores de aplicações.
JNDI	*Java Naming and Directory Interface* viabiliza o acesso a serviços de nome e diretórios tais como: DNS, NDS e LDAP.
RMI/IIOP	*Remote Method Invocation* cria interfaces remotas para a comunicação entre aplicações Java distribuídas. Utiliza, ainda, o protocolo de comunicação padrão do CORBA IIOP – *Internet Inter-ORB Protocol*.
Java IDL	*Java Interface Definition Language* cria interfaces remotas para viabilizar a comunicação entre aplicações Java e CORBA.
Servlets e JSP	*Servlets* Java e páginas JSP são componentes do lado servidor que rodam em um servidor Web que permita a geração dinâmica de HTML e a gerência de sessões para clientes Web.
JMS	*Java Message Service* viabiliza a comunicação assíncrona usando os modelos de programação *publisher/subscriber* ou enfileiramento.
JTA	*Java Transaction API* permite a demarcação das fronteiras das transações. Esta API é utilizada pelos *containers* na gerência das transações.
JTS	Especifica um gerenciador de transações que suporta JTA e baseia-se no serviço de gerência de transações OTS – *Object Transaction Service*, da OMG [OMG,2000].
JDBC	*Java Database Access* fornece acesso uniforme a sistemas de gestão de dados bancos de dados.
JavaMail	Fornece um protocolo independente para a construção de aplicações de correio eletrônico e de mensagens.

Tabela 24 – APIs corporativas Java

Passados dois anos do lançamento destas APIs, a *Sun* consolidou a sua plataforma para a construção de aplicações corporativas com a J2EE – *Java 2 Plataform Enterprise Edition*. A J2EE constitui-se, formalmente, por um conjunto de especificações inter-relacionadas com o intuito de determinar um padrão para a implementação e implantação de aplicações para corporações [Takagiwa, 2001]. Com a definição desta plataforma é possível garantir um ambiente de execução Java integrado capaz de garantir uma certa qualidade de serviço e assegurar a portabilidade e interoperabilidade das aplicações. As especificações definidas pela J2EE incluem [Sun Microsystems, 2002b]:

- Um modelo para a programação de aplicações;
- Uma plataforma para a execução das aplicações;
- Uma suíte de testes de compatibilidade, a fim de assegurar que as aplicações estão, fielmente, de acordo com o padrão;
- Uma implementação referência que exemplifica as potencialidades da J2EE.

A arquitetura J2EE

A plataforma J2EE especifica uma arquitetura para aplicações corporativas, baseada em componentes e multi-camadas. Neste contexto, definem-se componentes como blocos de códigos pré-desenvolvidos que podem ser montados para a composição de sistemas. De acordo com as camadas da arquitetura, é possível visualizar componentes do lado cliente, isto é processos executados na máquina cliente; e componentes do lado servidor[7], componentes que executam no servidor da aplicação.

O *container* é o elo de ligação entre os componentes e os serviços J2EE. Cada componente do modelo de programação especificado pela J2EE executa dentro de *containers*. Os *containers* são definidos pela especificação da J2EE com a responsabilidade de oferecer um ambiente de execução completo para os seus componentes. Dentre suas atribuições está o pooling de recursos, a gerência de estado dos componentes, que responsabiliza-se por serviços de segurança, além de questões de gerência de transações e resolução de nomes. A seguir, serão listadas as principais características dos *containers* do

[7] Ao longo desta dissertação, será utilizado, preferencialmente, o termo "componente do servidor" em referência a este conceito.

modelo de componentes da J2EE. Na Figura 58 é apresentado o modelo de componentes J2EE [Singh, 2002].

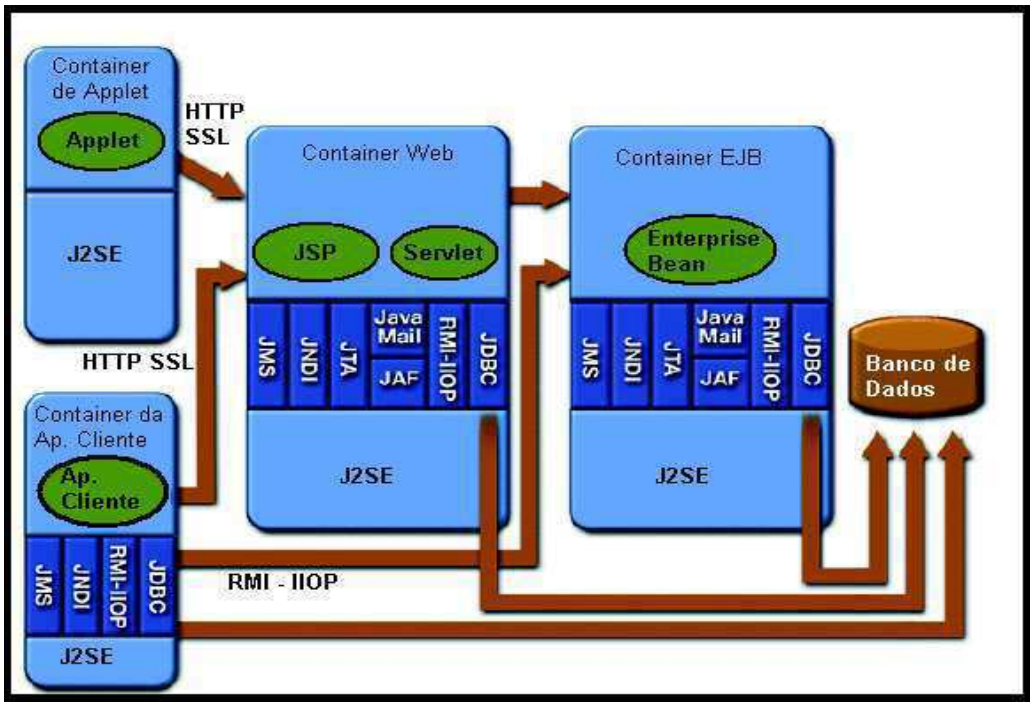


Figura 58 – Modelo de componentes J2EE

- *Container* de aplicações cliente: Este *container* dá suporte aos componentes das aplicações clientes. Deve fornecer acesso ao conjunto de serviços especificados pela J2EE, com exceção da gerência de transações.
- *Container* Web: Este *container* é usado para a execução de JSP e *servlets* que são os componentes Web. Ele interage com o servidor Web de onde recebe as requisições para os servlets e para onde encaminha as suas respostas.
- *Container* EJB: É neste *container,* localizado no servidor de aplicação, onde executam os EJBs que foram implantados. A especificação da J2EE define um contrato através do qual o *container* viabiliza o acesso aos EJBs. Isto é feito através das interfaces "Home" e "Remote" dos componentes. O *container* gerencia todas as características em tempo de execução do EJB: acesso remoto, transações, persistência, segurança, concorrência, dentre outras.

São considerados componentes do lado cliente, no modelo de componentes J2EE, os *applets* e as aplicações clientes.

- As aplicações clientes são programas Java com interface gráfica *Graphic User Interface* - GUI. Os componentes clientes que constituem estas aplicações, os *Java Beans*, têm acesso aos EJBs através do protocolo RMI-IIOP.
- Os *applets* também são utilizados como interface para aplicações J2EE. Eles são componentes de interface para páginas HTML que são transportados através da Internet e executados no browser do cliente ou em qualquer outro *container* de *applet*.

Consideram-se componentes do lado servidor, no modelo de componentes J2EE, os *servlets* e páginas JSPs e os EJBs.

- Os *servlets* e as páginas JSP são componentes do lado servidor que possibilitam uma ligação entre vários clientes e a lógica da aplicação [Thomas, 1999]. Estes componentes são executados no Web *container*, recebendo requisições via HTTP e gerando os resultados que serão devolvidos aos clientes. *Servlets* são classes Java que permitem que a lógica da aplicação se encaixe no processo de requisição/resposta do protocolo HTTP, dentro de páginas HTML. Há muitas restrições com relação ao uso de *servlets* dentro de páginas HTML. A principal delas diz respeito à apresentação dos resultados das requisições, cujo código HTML deve ser montado linha a linha. Já o JSP permite que conteúdo dinâmico seja adicionado às páginas Web através inserção de *tags*. Quando a página JSP é processada pelo *container,* as *tags* originam *servlets* que são carregados e executados dentro do servidor, como qualquer outro *servlets* [Takagiwa, 2001].
- Os EJBs – Enterprise Java Beans são componentes do lado servidor que implementam a lógica das aplicações corporativas do modelo de componentes J2EE. A arquitetura para a construção de componentes do lado servidor, para aplicações multi-camadas baseada em objetos distribuídos, definida pela API corporativa Java denominada *Enterprise Java Beans,* é peça fundamental de toda arquitetura J2EE. Cabe aos EJBs armazenarem a lógica de negócio da aplicação e, para tanto, acessar os dados e outros sistemas legados da organização (EIS) [Crupi, 2001]. Por fazer parte da plataforma J2EE, os EJBs asseguram a característica de

transportabilidade, ou seja, podem ser implantados em várias plataformas, desde que estas sejam certificadas J2EE, sem a necessidade de nova compilação ou modificação do código fonte. Os componentes EJBs executam dentro de servidores de aplicações Java. Tais servidores combinam tecnologia OLTP (*Online Transaction Processing*) com novas tecnologias para objetos distribuídos, a fim de oferecer alto desempenho, alta escalabilidade e robustez [Thomas, 1999]. Os servidores de aplicações simplificam o desenvolvimento de sistemas corporativos distribuídos, uma vez que fornecem serviços típicos de camada intermediária. Estes servidores, também chamados de servidores EJB, devem disponibilizar um *container* EJB onde, efetivamente, são implantados os EJBs. A função do *container* EJB, como frisado anteriormente, é gerir e controlar os serviços oferecidos para uma ou mais classes de EJBs, através da interceptação das requisições feitas a eles. Os *containers* EJB são responsáveis pela gerência do ciclo de vida, controle de persistência e serviços de segurança dos EJBs [Thomas, 1998]. A Figura 59 ilustra um *enterprise JavaBean* implantado no container de um servidor EJB. Os clientes têm acesso às operações do EJB através de suas interfaces que são expostas: Home e Remote. A primeira destina-se às operações do ciclo de vida do componente: criar, localizar e remover instâncias. A segunda oferece acesso aos métodos de negócio do EJB. Em tempo de implantação, o *container* gera, automaticamente, uma interface EJB do tipo *Home* para representar a classe, e uma interface EJB do tipo *Remote* para cada instância.

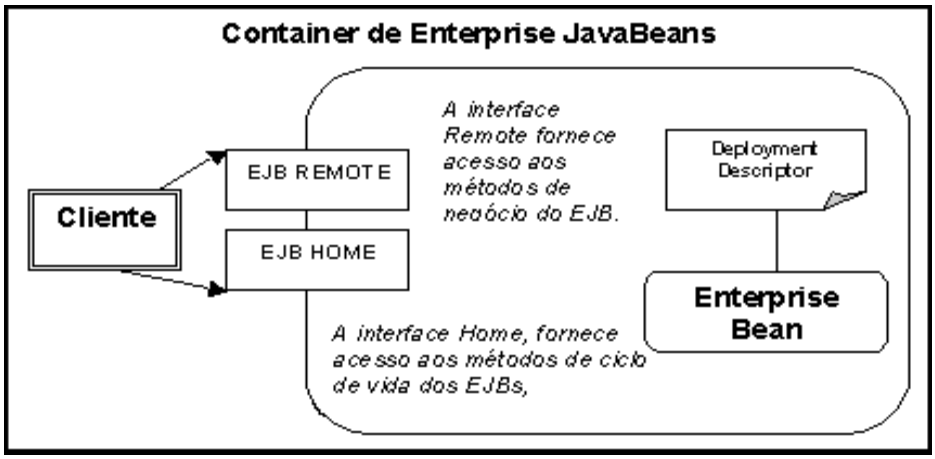


Figura 59 – EJB implantado em um container

Foge ao objetivo deste anexo esgotar um tema tão vasto quanto o aqui abordado. Deseja-se, tão somente, fornecer um embasamento tecnológico que facilite o entendimento do projeto desta dissertação, como anteriormente ressaltado. Mais detalhes técnicos sobre a tecnologia de *Enterprise JavaBeans* podem ser encontrados no Anexo A e nos documentos referenciados ao final desta dissertação [Roman, 2001], [Sun Microsystems, 2002] e [Sun Microsystems, 2002b].

Os componentes J2EE são agrupados em módulo. As aplicações podem ser constituídas de um ou mais módulos. A aplicação completa é empacotada em um arquivo denominado *Enterprise Archieve* (EAR). Os componentes EJBs e os módulos a ele relacionados são empacotados em arquivos do tipo *Java Archieve* (JAR). Os itens que compões o módulo Web, tais como, classes de *servlets*, páginas HTMLs, imagens e JSPs, são empacotados em um arquivo denominado Web Archieve (WAR). Módulos constituintes de aplicações clientes Java são empacotados em arquivos JAR [Takagiwa, 2001].

Cada empacotamento deve ser acompanhado por um arquivo do tipo XML [XML, 2002] que descreve as regras que regem a interação do módulo com o *container* no qual será implantado. O *deployment descriptor* é um contrato estabelecido entre o desenvolvedor do componente e o *container,* estabelecendo, por exemplo, as necessidades de recursos externos, parâmetros de ambiente, requisitos de segurança, dentre outras.

Os *deployment descriptors* podem ser criados e editados manualmente. No entanto, os servidores de aplicação normalmente dispõem de ferramentas que facilitam esta tarefa, freqüentemente, causadora de erros. As regras especificadas, de forma declarativa, e em tempo de desenvolvimento, são, em tempo de implantação, implementadas pelo *container*. A Figura 60 ilustra o empacotamento de uma aplicação J2EE com todos os seus módulos e descritores.

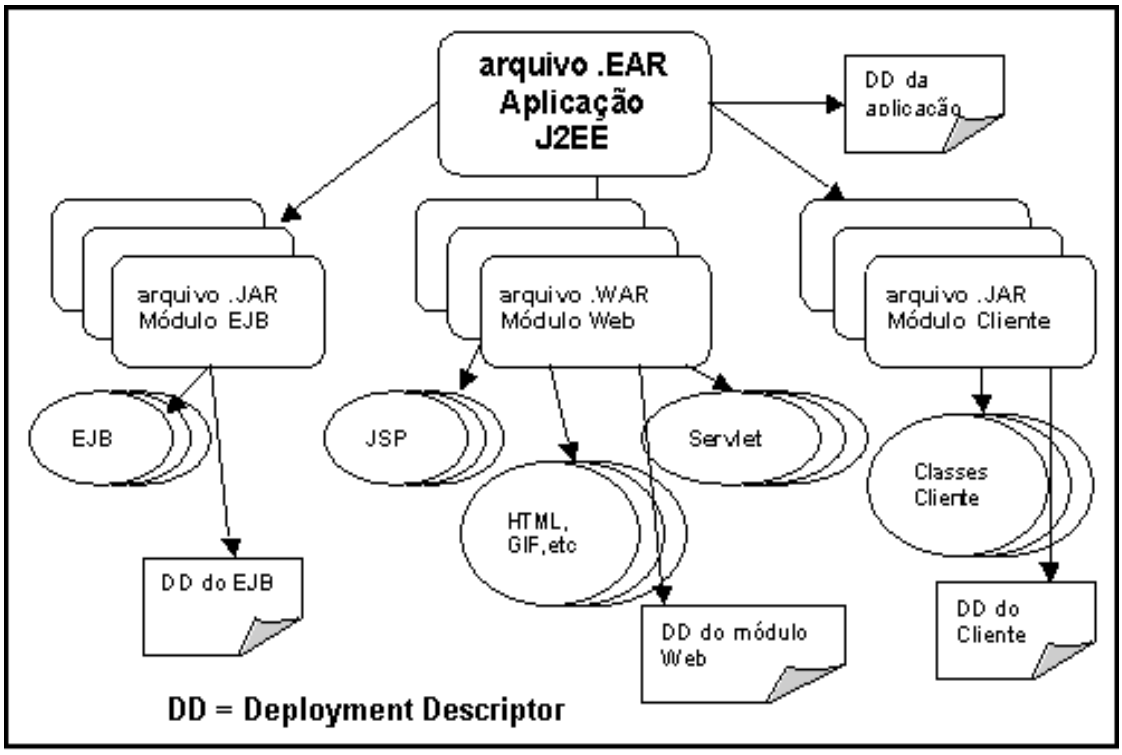


Figura 60 – Empacotamento de módulos de aplicação J2EE

A Figura 61 mostra um exemplo de *deployment descriptor* para um EJB. Neste tipo de componente, o descritor especifica como criar e manter um objeto *enterprise bean*. Ele define dentre outras coisas:

- O nome JNDI que representa o componente;
- Os nomes das interfaces Home e Remote;
- O nome do EJB;
- Se o *bean* é do tipo "*Session*", especifica o tipo da sessão;
- Se o *bean* for do tipo "*Entity*", especifica o tipo da persistência;
    - Caso a persistência seja gerida pelo *container,* devem ser especificados a chave primária e os campos geridos.
- O Tipo de transação;
- Outras referências EJBs.

```
<?xml version="1.0" encoding="UTF-8"?>

<ejb-jar>
     <description>Aplicacao Exemplo - Cesta de
compras</description>
     <display-name>Cesta EJB</display-name>
     <enterprise-beans>
       <session>
         <ejb-name>Cesta</ejb-name>
         <home>CestaHome</home>
         <remote>Cesta</remote>
         <ejb-class>CestaBean</ejb-class>
         <session-type>Stateless</session-type>
         <transaction-type>Bean</transaction-type>
       </session>
     </enterprise-beans>
</ejb-jar>
```

Figura 61 – Deployment Descriptor

Desde o lançamento da especificação J2EE, a Sun vem demonstrando interesse e preocupação com a disseminação de boas práticas de projeto e implementação de aplicações corporativas em cima de sua plataforma. A aplicação referência, sugerida na especificação [Singh, 2002], implementa diversas técnicas que podem ser reutilizadas. Além disso, a definição de padrões de projeto J2EE, elaborados por seu grupo pesquisa, visa a disseminação de estratégias de projeto comprovadamente eficientes para outros desenvolvedores.

A seguir, serão descritos alguns padrões de projeto que determinam características arquiteturais que devem estar presentes nas aplicações J2EE de forma a garantir a qualidade do projeto:

- **Arquitetura MODEL – VIEW – CONTROLLER**

O MVC, ou Modelo-Visualização-Controlador, é um padrão de projeto que objetiva a decomposição de uma aplicação em entidades lógicas, i.e., componentes que têm função de negócio bem definida. Esta divisão permitirá uma fácil divisão em camadas, incrementando a flexibilidade e a reusabilidade de código [Gamma *et al*, 1994].

O uso do MVC minimiza o grau de acoplamento entre os componentes que mantêm e os que apresentam os dados. Com isto, o mesmo sistema pode suportar vários tipos de interface, sem necessidade da realização de alterações ou adaptações no código dos componentes de negócio. O MVC foi originalmente desenvolvido e utilizado em projetos

Smalltalk, onde era aplicado para a realização do mapeamento das tradicionais tarefas de entrada, processamento e saída de dados, para o módulo de interface gráfica – GUI do usuário. No plano das aplicações corporativas J2EE, este mapeamento é feito para aplicações multi-camadas baseadas em Web.

O MVC determina a divisão arquitetural das aplicações em três camadas – *Model*, *View* e *Controller* – desacoplando respectivamente suas responsabilidades. Aplicar a arquitetura MVC aos sistemas J2EE, permite a separação das funcionalidades básicas de acesso aos dados, da lógica de apresentação e de controle que faz uso destas funcionalidades. Esta separação permite que várias interfaces compartilhem o mesmo modelo de dados corporativos [Crupi, 2001]. Os benefícios obtidos são evidenciados nos testes, implementação e manutenção de várias interfaces clientes facilitadas por esta arquitetura.

As funções de cada camada da arquitetura MVC estão detalhadas a seguir [Crupi, 2001]:

- *Model:* Representa os dados e as regras de negócio aplicadas aos dados para acessá-los ou modificá-los. O *Model* encerra a funcionalidade efetiva da aplicação. Ele deve alertar a *View* quando sofre modificações e deve disponibilizar ao *Controller* acesso às suas funcionalidades de negócio que estão encapsuladas.
- *View:* Apresenta o conteúdo do *Model.* Ela tem busca os dados da *Model* e determina como eles devem ser apresentados. É responsabilidade da *View*, atualizar os dados apresentados quando a *Model* muda. A *View* interage com a *Controller* repassando a ele as requisições de entrada do usuário.
- *Controller:* Define o comportamento da aplicação, uma vez que encaminha as requisições da *View* à *Model*, após adequado mapeamento da entrada do usuário. A partir daí, a *Model* desencadeia suas ações. A *Controller* também seleciona a *View* a ser utilizada para a apresentação dos resultados obtidos. Uma aplicação geralmente tem um *Controller* para cada grupo de funcionalidade relacionada.

A Figura 62 retrata o relacionamento entre as camadas *Model*, *View* e *Controller* do MVC, bem como as suas principais características.

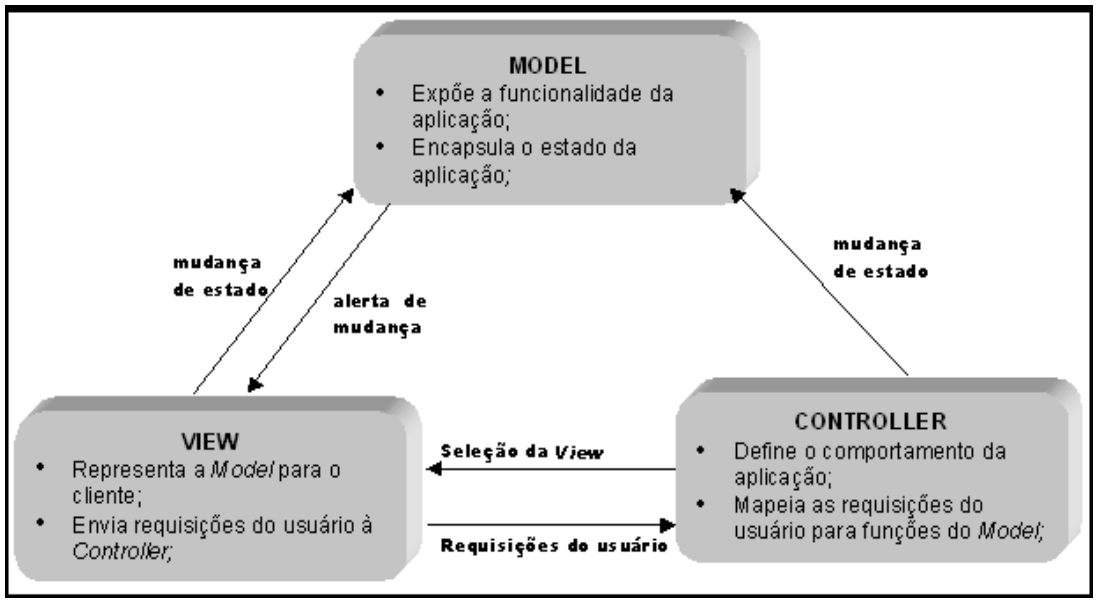


Figura 62 – Arquitetura MVC

- **Padrões de projeto J2EE**

Os padrões de projeto são soluções comprovadas para problemas recorrentes. Eles ajudam a disseminar o conhecimento de especialistas para a comunidade da área. Os padrões de projeto já vêm sendo utilizados com bastante sucesso em diversos campos de atuação. No domínio de aplicação J2EE, alguns padrões já foram publicados por diferentes autores.

A seguir, serão apresentados alguns dos principais padrões de projeto J2EE, [Crupi, 2001] suas descrições e a camada a que sua solução se aplica.

- *Intercepting Filter:* Este padrão se aplica às situações onde há a necessidade de pré ou pós processamento. Pode ser usado, por exemplo, na implementação de um mecanismo centralizado de autenticação. Localizado na camada *Controller*. Veja a Figura 63.

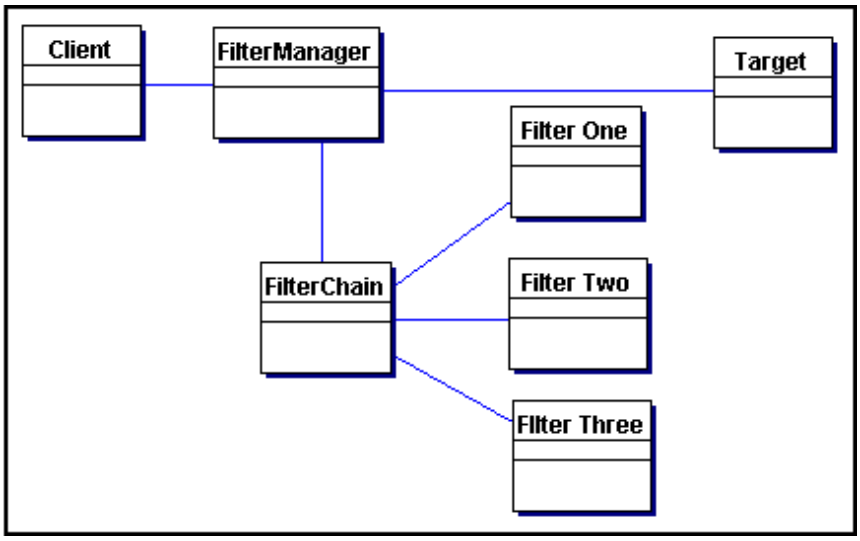


Figura 63 – Padrão de projeto J2EE: Intercepting Filter

- *View Helper:* Sugere o refinamento da *View*, tornando-a mais simples, através da separação da lógica de apresentação dos dados de sua lógica de acesso. Um exemplo prático de implementação deste padrão é a utilização de JSP para formatar e apresentar os dados e *JavaBeans* para recuperá-los. Localizado na camada *View*. Veja a Figura 64.

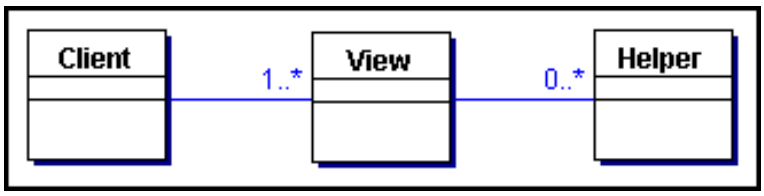


Figura 64 – Padrão de projeto J2EE: View Helper

- *Composite View:* Sugere a criação de um molde para a manipulação de elementos comuns da *View,* a fim de torná-la mais fácil de ser trabalhada. Este padrão é aplicado a *Views* baseadas em Web e partem do princípio que páginas tem elementos dinâmicos e estáticos. Localizado na camada *View (Web-Tier)*. Veja aFigura 65.

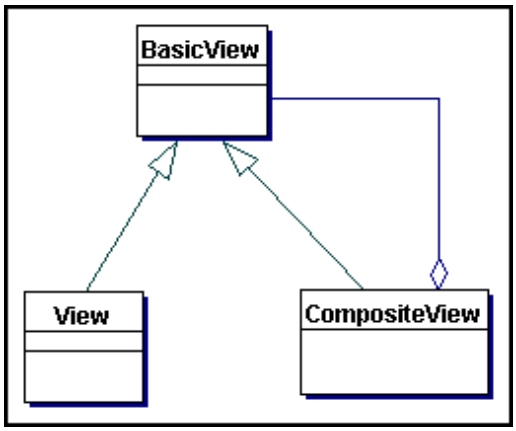


Figura 65 – Padrão de projeto J2EE: CompositeView

- *Session Façade:* Este padrão unifica as funcionalidades da aplicação sob uma interface, facilitando o acesso e as invocações dos métodos pelos clientes do negócio. A session *façade* não implementa a funcionalidade, mas encaminha as requisições para quem efetivamente a realiza. Geralmente, a *Session Façade* é implementado por *session beans* que interagem com outros *enterprise beans*. Localizado na camada *Model(EJB-Tier)*. Veja a Figura 66.

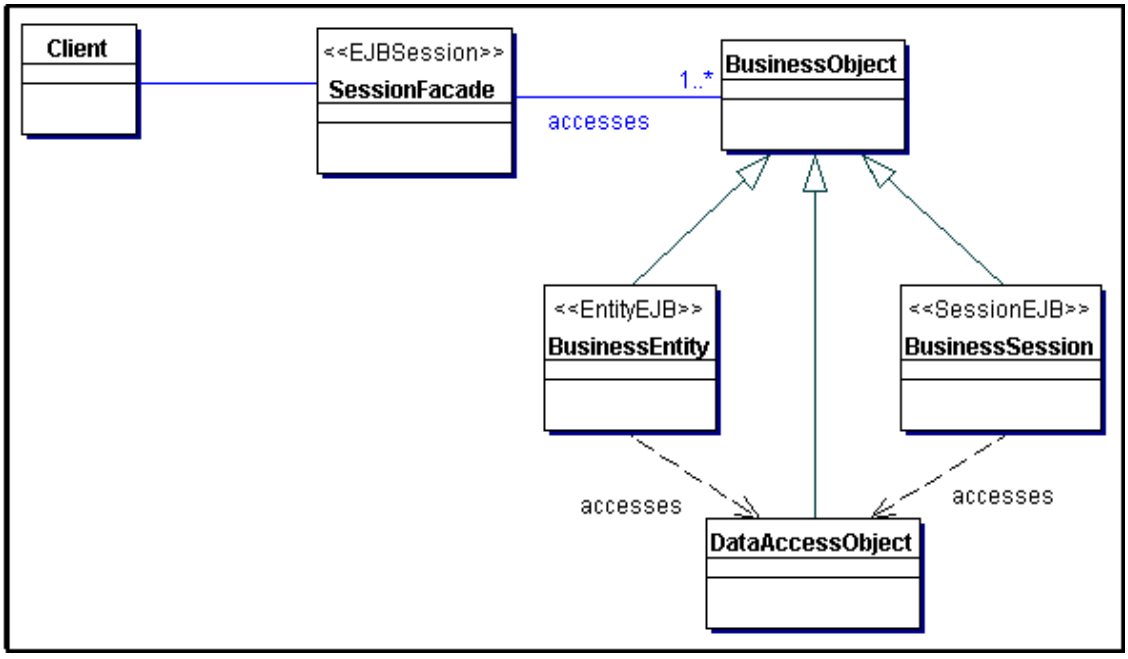


Figura 66 – Padrão de projeto J2EE: Session Façade

- *Business Delegate:* Procura minimizar o acoplamento entre a camada de negócio *(EJB-Tier)* e seus clientes *(Web-Tier).* Propõe a criação de um *proxy*, ou camada, que

viabiliza o acesso aos serviços de negócio encapsulando os detalhes de implementação intrínsecos à API corporativa EJB, tais como: localização de objetos remotos e manipulação de exceções remotas. Além da possibilidade de implementação de outros serviços de melhoria de desempenho, como, por exemplo, o *caching* de dados, a fim de reduzir a quantidade de chamadas remotas necessárias para a realização de determinada funcionalidade. Aplicado entre camadas. Veja a Figura 67.

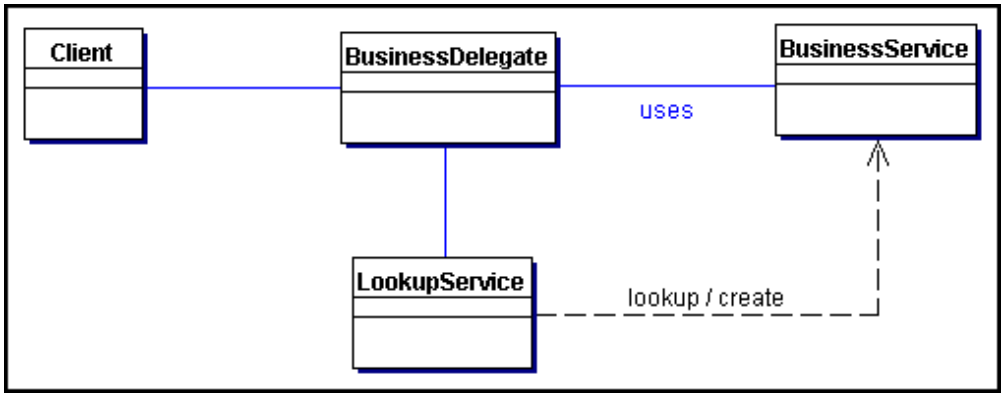


Figura 67 – Padrão de projeto J2EE: Business Delegate