UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

"FrameEST: Um *Framework* de componentes, no padrão MVC, para o domínio de Biologia Molecular"

Luiz Roberto Lombardo

SÃO CARLOS
Agosto/2006

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

L842ff

Lombardo, Luiz Roberto.
FrameEST: um framework de componentes, no padrão MVC, para o domínio de biologia molecular / Luiz Roberto Lombardo. -- São Carlos : UFSCar, 2006.
69 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2006.

1. Arquitetura de computador. 2. Framework (Programa de computador). 3. Componentes de software. 4. MVC – Model view control. 5. Biologia molecular. I. Título.

CDD: 004.22 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"FrameEST: Um Framework de componentes, no padrão MVC, para o domínio de Biologia Molecular"

LUIZ ROBERTO LOMBARDO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Prof. Dr. Mauro Biajiz
(Orientador – DC/UFSCar)

Prof. Dr. Antonio Francisco do Prado
(DC/UFSCar)

Profa. Dra. Rosângela Ap. Dellosso Penteado
(DC/USCar)

Prof. Dr. Júlio César Sampaio do Prado Leite
(PUC/Rio)

São Carlos
Agosto/2006

Dedico este trabalho a todas as pessoas da minha família, em especial ao meu avô Luiz Lombardo (in memorian).

AGRADECIMENTOS

A DEUS, pela proteção, luz e por ter colocado em meu caminho pessoas maravilhosas, meus familiares e amigos.

Aos meus pais João e Aldiva, que sempre me apoiaram em todas as minhas escolhas e que me educaram para a vida.

A minha namorada Gislaine, pelo estímulo, companhia e solidariedade, principalmente nos finais de semana e feriados.

A empresa CESTARI, pelo apoio e confiança, e aos amigos do departamento de informática, pelo incentivo e ajuda, especialmente ao Mohamed, que além de amigo, também é como um irmão.

A todos os amigos e colegas que fiz na UFSCar, em especial aos professores Dr. Mauro Biajiz e Dr. Antônio Francisco do Prado, pois além da orientação, me ensinaram coisas que sempre levarei comigo.

RESUMO

Atualmente, vários projetos de genomas de diferentes seres vivos estão sendo mapeados gerando um grande volume de dados, os quais são armazenados em fontes de dados heterogêneas e distribuídas. Além disso, existem ferramentas disponíveis no domínio de genomas que também necessitam serem integradas. Outro problema é que os sistemas desenvolvidos para este fim não atendem aos pesquisadores, pois na sua grande maioria não possuem flexibilidade e são de difícil expansão.

A proposta deste trabalho é o desenvolvimento de um *framework* de componentes de *software*, denominado FrameEST, desenvolvido com as mais recentes tecnologias de reuso, que estrutura e orienta o desenvolvimento de diferentes aplicações do domínio de biologia molecular. O FrameEST está disponível para reuso das aplicações: na fase de modelagem em uma ferramenta *CASE* e na fase de implementação como um *plug-in* no ambiente *Eclipse*. Um estudo de caso é utilizado para ilustrar o reuso do FrameEST.

ABSTRACT

Nowadays, some projects of genomes of different organisms are being analyzed generating a great volume of data, which are stored in heterogeneous and distributed data sources. Moreover, there are available tools in the genome domain that also need to be integrated. Another problem is that the systems developed for these objectives do not offer all the support to the researchers, therefore in their majority do not possess flexibility and are of difficult expansion.

The proposal of this work is the development of a software component framework, called FrameEST, developed with the most recent technologies of reuses, that structures and guides the development of different applications of molecular biology domain. The FrameEST is available for reuses of the applications: in the phase of modeling in a CASE tool and the phase of implementation as one plug-in in the Eclipse enviroment. A case study is used to illustrate the FrameEST reuse.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API	Application Programming Interface
CASE	Computer Assited Software Engineering
EJB	Enterprise JavaBeans
EST	Expressed Sequence Tags
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Enviroment
JCP	Java Community Process
JDBC	Java DataBase Connectivity
JNDI	Java Naming and Directory Interface
JSF	JavaServer Faces
JSP	Java Server Page
JVM	Java Virtual Machine
MVC	Model View Control
OMT	Object Modeling Technique
RMI	Remote Method Invocation
SGBDR	Sistema Gerenciador de Banco de Dados Relacional
SQL	Structure Query Language
UML	Unified Modeling Language
URL	Uniform Resource Locator
XMI	XML Metadata Interchange
XML	eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

As pesquisas sobre Genomas [35] [39] tiveram seus inícios por volta de uma década atrás. Inicialmente o objetivo era seqüenciar todo o Genoma Humano, caracterizando um grande projeto de pesquisa básica para posteriormente os cientistas poderem almejar o desenvolvimento de curas para doenças através de tratamentos genéticos e produção de novos medicamentos. Com o passar dos anos, o genoma de outras espécies também começou a ser seqüenciado, com objetivos similares, ou seja, o de fornecer conhecimento para obtenção de melhorias genéticas para os mais diversos fins. Essa fase inicial foi denominada era pré-genômica, uma fase de pesquisa básica.

Com o avanço das pesquisas, grandes volumes de dados foram sendo produzidos e armazenados em fontes de dados heterogêneas e distribuídas em diferentes bancos de dados e arquivos texto, gerando forte demanda por *softwares* para o tratamento dos mesmos. Iniciou-se então uma nova fase que vem sendo denominada pós-genômica.

Para a fase Pós-Genomica, deve-se integrar esse grande volume de dados que estão distribuídos em diferentes fontes de dados, necessitando serem qualitativamente trabalhados, em um processo conjunto entre Biólogos e profissionais em computação.

1.1 MOTIVAÇÃO

As várias fontes de dados de genoma armazenadas em bancos de dados e arquivos texto, e suas aplicações, representam hoje um dos principais recursos para apoiar as pesquisas dos biólogos moleculares e geneticistas. Para que essas fontes de dados possam ser utilizadas se fazem necessários mecanismos seguros de extração, processamento, análise e armazenamento desses dados. Outra dificuldade é que essas aplicações muitas vezes utilizam diferentes ferramentas, já disponíveis nesse domínio, para analisar e formatar as informações extraídas dessas fontes de dados. Além disso, os sistemas desse domínio são construídos com diferentes padrões de desenvolvimento, não existindo preocupação com o reuso de *software*, o

que pode acarretar redundâncias e inconsistências. Para a solução dessa questão, são necessários técnicas, métodos e ferramentas para padronizar e dar suporte à construção de sistemas, que possibilite a integração dessas fontes de dados, permitindo seu uso amigável e adequado.

A integração de todos esses dados e ferramentas em um único *framework* é a grande motivação para a construção deste projeto de pesquisa.

1.2 OBJETIVO

Este trabalho apresenta um *framework* com o objetivo de facilitar o desenvolvimento e reuso de aplicações para a integração, extração e o processamento de informações de ferramentas e fontes de dados heterogêneas e distribuídas no domínio de biologia molecular.

O *framework* possibilita uma abordagem de reuso, desde a modelagem até a implementação, através de componentes de *software*, que atende aos requisitos de flexibilidade e extensibilidade, na construção das diferentes aplicações, possibilitando dessa forma o desenvolvimento de aplicações cada vez mais padronizadas.

1.3 ORGANIZAÇÃO

Esta dissertação está estruturada da seguinte forma: no capítulo 2 serão apresentados os conceitos básicos e definições sobre o domínio de biologia molecular. O capítulo 3 apresenta os conceitos e características dos Componentes de *Software* e as tecnologias relacionadas tais como Padrões de *Software* e *Frameworks*. Neste capítulo também são apresentados os conceitos das tecnologias *JavaServer Faces, JavaBeans* e *Enterprise JavaBeans*. O capítulo 4 apresenta as arquiteturas de integração de bioinformática coletadas na literatura, apresentando suas principais características. O capítulo 5 descreve a arquitetura, bem como o processo de desenvolvimento e implementação do *framework* FrameEST. O capítulo 6 apresenta um estudo de caso, a partir da reutilização dos

componentes do *framewok*. E finalmente o capítulo 7 apresenta as conclusões, contribuições e trabalhos futuros.

2 BIOLOGIA MOLECULAR

Neste capítulo serão apresentados os conceitos de biologia molecular que devem ser entendidos para a correta compreensão dos componentes do *framework* proposto e implementado neste projeto. Irão ser explanados alguns conceitos básicos da área de biologia celular e molecular [35] [39].

2.1 CONCEITOS

O estudo dos seres vivos mostra que a evolução produziu uma imensa variedade de formas. Existem aproximadamente quatro milhões de espécies diferentes de bactérias, protozoários, vegetais e animais, que diferem em sua morfologia, função e comportamento. Entretanto, sabe-se agora que, quando os organismos vivos são estudados nos aspectos celular e molecular, observa-se um plano único principal de organização. O objetivo da biologia celular e molecular é precisamente esse plano unificado de organização, isto é, a análise das células e moléculas, que constituem as unidades estruturais de todas as formas de vida. A célula é a unidade estrutural e funcional básica dos organismos vivos.

Todos os processos, realizados por uma célula, são determinados por seu conteúdo genético, mais precisamente, pelas instruções contidas em uma coleção de genes. Estes genes são passados de uma geração para a próxima, de modo que uma prole herda uma série de características de seus pais. A engenharia genética envolve o transporte de genes de sua localização normal em um organismo para outro ou retornando ao organismo original após manipulação. Dessa forma, os cientistas podem retirar genes importantes de plantas e animais e transferi-los para microrganismos tais como bactérias. Um exemplo são bactérias manipuladas geneticamente para produzir insulina humana para tratar a diabetes. Um segundo benefício está relacionado com a transferência de características de uma planta, animal ou microrganismo para uma outra espécie não relacionada. Desta forma, as características

poderão ser incorporadas sem a necessidade de métodos convencionais, na sua grande parte demorados e nem sempre bem sucedidos.

Os cientistas possuem todo esse entendimento agora, graças a compreensão dos pontos que envolvem uma molécula chamada DNA. A molécula de DNA contém subunidades chamada nucleotídeos. Cada nucleotídeo é composto por uma molécula de açúcar (pentose), bases nitrogenadas (purinas e piridiminas) e ácido fosfórico. As pentoses são de dois tipos: ribose no RNA e desoxirribose no DNA. Quanto às bases nitrogenadas, existem as piridiminas que são timina (T) e citosina (C) e as purinas que são adenina (A) e guanina(G).

O DNA localiza-se principalmente no núcleo da célula, dentro dos cromossomos e é formado por duas fitas de nucleotídeos, mantidas unidas por pontes de hidrogênio entre as bases das fitas opostas. Como mostra a figura 1, a estrutura é como uma escada, na qual os corrimãos são formados pelos grupamentos fosfatos e açúcares e, os degraus pelas bases. As bases são como peças de um quebra-cabeça, encaixadas em pares e com as possibilidades únicas de ligação, que são A com T e C com G. As duas fitas se enrolam formando uma hélice.

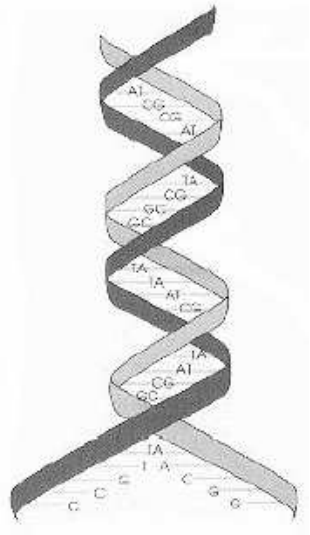

FIGURA 1 – ESTRUTURA EM DUPLA FITA DO DNA [41]

Semelhante ao DNA existe o RNA, diferenciando-se pela substituição da ribose pela desoxirribose e da timina (T) pela uracila (U). A composição de bases do RNA não é

similar à do DNA, pois as moléculas de RNA são compostas por uma única cadeia, e são encontradas tanto no núcleo da célula, onde é sintetizada, quanto no citoplasma.

Para a tradução do código genético, além das moléculas de DNA e RNA, também são necessárias as proteínas. Proteínas são moléculas extremamente versáteis que podem possuir dezenas ou centenas de aminoácidos. Ao contrário do DNA, que assume a forma de uma hélice regular, as proteínas assumem uma enorme variedade de formas tridimensionais.

O processo de tradução do código genético é da seguinte forma: o DNA é o principal armazenador da informação genética. Essa informação é copiada ou transcrita para moléculas de RNA, cujas seqüências de nucleotídeos contêm o "código" para a ordenação específica de aminoácidos. As proteínas são, então, sintetizadas num processo que envolve a tradução do RNA. De forma resumida, a figura 2 ilustra esse processo.

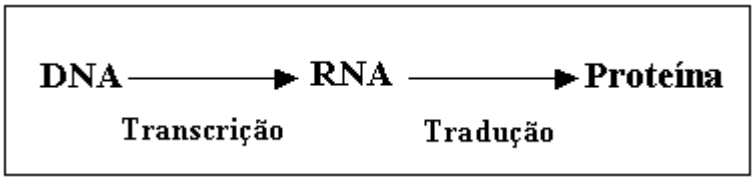

FIGURA 2 - PROCESSOS TRANSCRIÇÃO E TRADUÇÃO.

O conteúdo de todo DNA presente em uma célula, incluindo todos os genes e todas as regiões intergênicas são chamadas de Genoma.

2.2 SEQÜÊNCIAS

O termo "seqüência finita de caracteres", ou simplesmente "seqüência" ou "cadeia", será usado no sentido restrito de uma seqüência finita de caracteres de um dado alfabeto S. Assim, se S = {A,C,T,G}, então ATTCCG e CCGA são seqüências.

As seqüências (que também podem ser chamadas de *biosequências*) podem ser tratadas como cadeias de texto. Por esse motivo, os biólogos moleculares podem coletá-las e guardá-las em arquivos texto. Foi isso o que foi feito no início dos processos de sequenciamento. No entanto, com o avanço da tecnologia, a produção de seqüências aumentou e, conseqüentemente, os dados armazenados em arquivos texto cresceram muito,

tornando sua manutenção e a dos programas de aplicação relacionados muito trabalhosa. Diante disso os biólogos moleculares começaram a usar Sistemas Gerenciadores de Bancos de Dados (*SGBD*), mais apropriados para gerenciar grandes volumes de dados.

Uma das tecnologias utilizadas para sequenciamento genético é a técnica chamada Etiquetas de Seqüências Expressas (*EST, tradução de Expressed Sequence Tags*), uma técnica que identifica as porções dos genes que codificam proteínas, os chamados genes funcionais. Através do sequenciamento, os pesquisadores "decifram" a ordem das bases químicas que constituem o código genético de um ser vivo.

No próximo capítulo serão apresentados os principais conceitos e definições sobre as tecnologias utilizadas na construção do *framework*.

3 *CATALYSIS*, PADRÕES, *FRAMEWORKS* E COMPONENTES DE *SOFTWARE*.

Este capítulo irá descrever os conceitos e características das tecnologias utilizadas no desenvolvimento deste trabalho de pesquisa. Todo o processo de desenvolvimento do FrameEST, desde a modelagem até a implementação, foram baseados nas mais recentes tecnologias e técnicas disponíveis na engenharia de *software*, conforme descritas a seguir.

3.1 *CATALYSIS*

Catalysis [10] é um método de desenvolvimento de *software* baseado em componentes que integra Técnicas, Padrões e *Framework*. Esse método é uma iniciativa de pesquisa desenvolvida na Universidade de *Brighton*, Inglaterra, por *D' Souza e Wills* [10]. Começou em 1991 como uma formalização do *OMT* (*Object Modeling Technique*) [37] e tem como princípios a Abstração, a Precisão e os Componentes *Plug-In*. O princípio Abstração orienta o Engenheiro de *Software* na busca dos aspectos essenciais do sistema, dispensando detalhes que não são relevantes para o contexto do sistema. O princípio Precisão objetiva descobrir erros e inconsistências na modelagem e o princípio Componentes "*Plug-In*" visam a reutilização de componentes para construir outros componentes.

O Processo de Desenvolvimento de *Software* em *Catalysis*, como mostra a Figura 3, segue as características do modelo evolutivo de desenvolvimento de *Software* e está dividido em três níveis lógicos: Domínio do Problema, Especificação dos Componentes e Projeto Interno dos Componentes, correspondendo às atividades tradicionais do ciclo de vida do *software*: Planejamento, Especificação, Projeto e Implementação, que são executadas de forma incremental e evolutiva, resultando na geração de uma nova versão de um protótipo a cada ciclo realizado.

No nível Domínio do Problema é dada ênfase na identificação dos requisitos do sistema, especificando "o que" o sistema deve fazer para solucionar o problema. Identificam-se os tipos de objetos e ações, agrupando-os em diferentes visões por áreas de negócio. Nesse

nível utilizam-se técnicas de *entrevistas* com o usuário, para encontrar pontos importantes no domínio do problema. Feito isso, os requisitos coletados são transportados para o *Modelo de Colaboração*, representando a coleção de ações e os objetos participantes, que após um refinamento serão mapeados para o *Modelo de Casos de Usos*, que representam os atores e suas interações com o sistema.

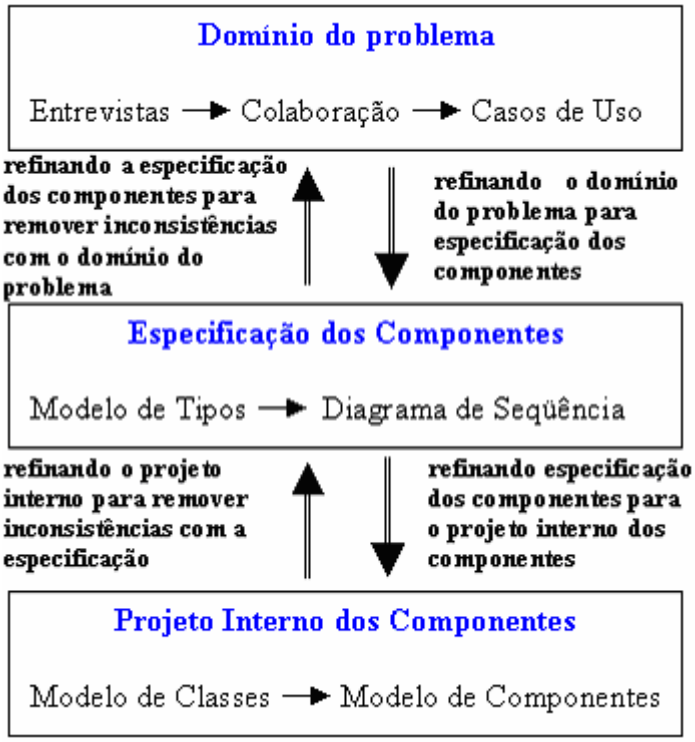

FIGURA 3 – OS TRÊS NÍVEIS DE DESENVOLVIMENTO DE *CATALYSIS*

No nível Especificação dos Componentes dá-se ênfase na identificação, comportamento e objetivos dos componentes. Nesse nível faz-se o refinamento dos modelos de negócios, do domínio do problema, através do mapeamento dos diagramas obtidos no *Domínio do Problema* para o *Modelo de Tipos* que especifica o comportamento dos objetos. Este último modelo é então refinado para listar as ações de cada tipo através do *Diagrama de Seqüência*, que apresenta a seqüência de interações entre os objetos ao longo do tempo.

No nível do Projeto Interno dos Componentes dá-se ênfase na implementação dos requisitos especificados para os componentes do sistema, preocupando-se com suas distribuições físicas. Nesse nível, utiliza-se a técnica de *Diagrama de Classes* para representar

as classes com seus atributos, operações e relacionamentos e técnicas, como *Diagrama de Componentes*, para representar a arquitetura das plataformas física e lógica.

Catalysis apresenta dois modelos específicos para representar *Frameworks*, o ***Modelo do Framework*** e o ***Modelo da Aplicação***. O *Modelo do Framework* representa a especificação estática do *framework*, identificando os tipos com os seus atributos e relacionamentos. Nesse modelo são mostrados os tipos que podem ser estendidos, indicando-os com os sinais < > (*placeholders)*. O *Modelo da Aplicação* representa a dependência dos tipos do *framework* com os tipos estendidos na aplicação.

Todos estes níveis de abstração do método *Catalysis* foram empregados no desenvolvimento do FrameEST.

3.2 PADRÕES DE PROJETO

A chave para habilitar toda a reusabilidade de um *framework* é sua documentação, que deve ser clara e objetiva [6], pois é a partir dessa que, qualquer pessoa consegue entender e utilizar o *framework*. Para tanto, foram utilizados padrões de projeto para essa finalidade.

A origem dos padrões de projeto deu-se com o trabalho feito pelo arquiteto *Christopher Alexander* [2], no final dos anos 70, para construção de arquiteturas e planejamento urbano. O trabalho de *Alexander*, apesar de ser voltado para a arquitetura, possui uma fundamentação básica, que pode ser abstraída para a área de *software*.

A utilização de padrões em sistemas complexos de *software* permite que soluções, previamente testadas, sejam reutilizadas, tornando o sistema mais compreensível, flexível, e fácil de desenvolver e manter. O objetivo do uso de padrões de *software* é a disseminação das soluções de desenvolvimento de *software* já existentes [22]. Projetistas, familiarizados com certos padrões, podem aplicá-los imediatamente a problemas de projeto, sem ter que redescobri-los [15], fornecendo uma linguagem comum, que facilita a comunicação entre desenvolvedores e projetistas.

Alguns padrões são bastante conhecidos destacando-se os do Catálogo de *Gamma* [15] e o catálogo *J2EE Pattern* [3]. Na construção do FrameEST foram utilizados diversos padrões destacando-se o *MVC* (*Model View Control*) [17], *Abstract Factory* [15], *Factory Method* [15], *Business Delegate* [3], *Session Facade* [3] e o *Composity Entity* [3]. O padrão *MVC* estrutura as aplicações em 3 camadas (*Model, View* e *Control*). Nas camadas *View* e *Control*, o padrão *Abstract Factory* fornece uma interface para a criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas. O padrão *Factory Method* permite a instanciação de objetos através da redefinição de subclasses. Na camada *Model*, o padrão *Business Delegate* oculta detalhes de implementação e de acesso à lógica de negócio. O padrão *Session Facade* encapsula e gerencia a complexidade de interação entre os objetos de dados e objetos de negócios, provendo seus serviços através de interfaces, e o padrão *Composity Entity* possibilita modelar, representar e gerenciar o relacionamento entre os objetos de persistência dos dados.

Além dos padrões apresentados, as seções seguintes apresentam outros dois padrões muito utilizados em *frameworks*, o padrão *Template Method* e *Strategy* [15] [16].

3.2.1 PADRÃO *TEMPLATE METHOD*

O padrão *Template Method* ou Método Esqueleto permite que subclasses redefinam determinadas etapas de um algoritmo sem alterar a estrutura do algoritmo, utilizando operações abstratas através de herança. Este padrão possui uma classe abstrata que contém apenas parte da lógica necessária para realizar um propósito. A organização da classe é de tal forma que os métodos concretos chamam os métodos abstratos onde a lógica restante será implementada. Esse padrão é a base para a construção de *frameworks*.

A Figura 4 ilustra a estrutura do padrão *Template Method*. Nela, a classe abstrata define operações abstratas que subclasses concretas irão implementar. Nessa classe, também é

implementado um *Template Method*, definindo o esqueleto de um algoritmo. O *Template Method* chama várias operações, entre as quais, incluem-se as operações abstratas da classe.

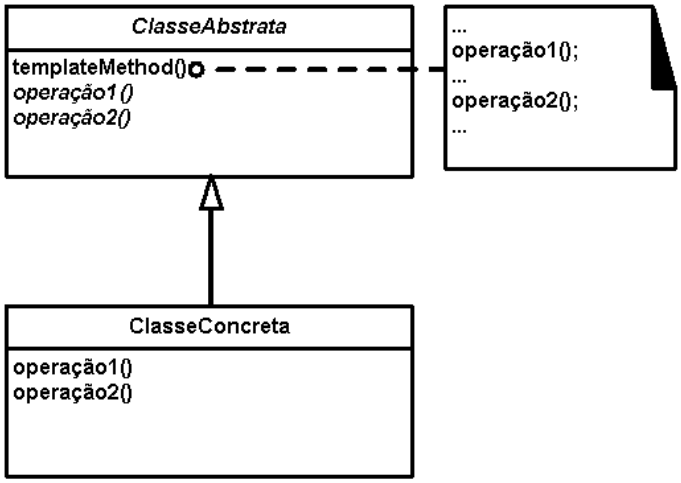

FIGURA 4 - ESTRUTURA GENÉRICA DO PADRÃO *TEMPLATE METHOD* [15]

Já a classe concreta implementa as operações abstratas para desempenhar as etapas do algoritmo, que tenham comportamento específico à esta subclasse.

3.2.2 PADRÃO *STRATEGY*

O padrão *Strategy* ou Estratégia permite definir uma família de algoritmos, encapsulá-los em objetos e disponibilizá-los para reuso. Esse padrão permite encapsular algoritmos relacionados em classes, que são subclasses de uma superclasse comum. Isso permite a seleção de algoritmos que possam variar por objeto e por tempo de execução.A figura 5 apresenta a estrutura do padrão *Strategy*.

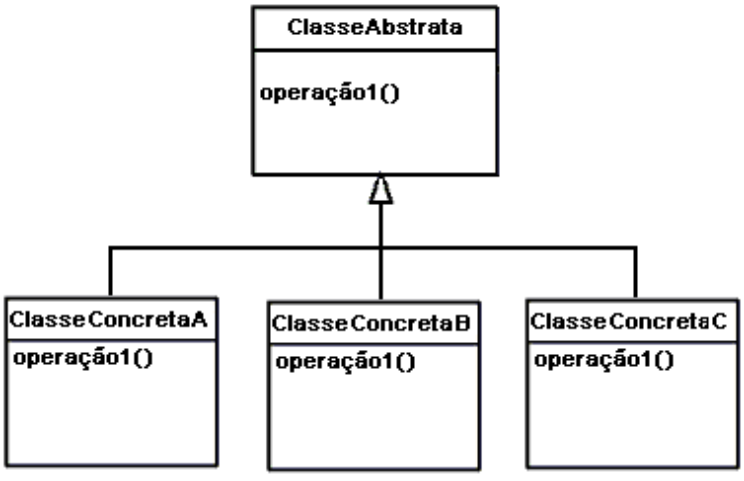

FIGURA 5 - ESTRUTURA GENÉRICA DO *PADRÃO STRATEGY* [15]

Nesse padrão, uma hierarquia de classes define uma família de algoritmos. A classe abstrata declara uma interface comum para suportar todos os algoritmos. Esta interface é responsável por acessar os algoritmos definidos nas classes concretas, as quais possuem uma implementação do algoritmo.

Ambos os padrões, *Template Method* e *Strategy* são muito utilizados no desenvolvimento de *frameworks*, considerando que a construção de *frameworks* se inicia com o padrão *Template Method*, e à medida que este vai evoluindo, o padrão utilizado passa a ser o *Strategy*.

Na seção seguinte, é apresentado o conceito relacionado à estrutura, benefício e desenvolvimento de um *framework*.

3.3 *FRAMEWORKS*

O termo *framework* [21], bastante genérico na língua inglesa, é utilizado neste trabalho conforme definido em engenharia de *software*, isto é, como uma arquitetura flexível e extensível para a construção de uma família de sistemas, aplicada a um determinado domínio. A idéia de *framework* orientado a objetos, depois de sua concepção no final de 1980, tem atraído a atenção de muitos pesquisadores e engenheiros de *software* [7].

Framework é definido por *Johnson at el.* [23] como um conjunto de classes que incorpora um projeto abstrato para a solução de problemas relacionados e por *Coad* [8] como um esqueleto de classes, objetos e relacionamentos agrupados para construir aplicações específicas. Essas classes podem fazer parte de uma biblioteca de classes ou podem ser específicas da aplicação. *Frameworks* possibilitam reutilizar não só componentes isolados, como também toda a arquitetura de um domínio específico.

Uma das principais características de um *framework* é a sua capacidade de reuso, que não somente envolve o aproveitamento de trechos de códigos de programas, mas, também, a reutilização de esforços de todas as fases de desenvolvimento de *software*, desde a análise de requisitos, passando pelo projeto do *software*, implementação e testes [5]. Ele reusa análise, porque descreve os tipos de objetos importantes e como um problema maior pode ser

dividido em problemas menores. Ele reusa projeto, porque contém algoritmos abstratos e descreve a interface que o programador deve implementar e as restrições a serem satisfeitas pela implementação. Ele reusa código, porque torna mais fácil desenvolver componentes compatíveis e porque a implementação de novos componentes pode herdar grande parte de seu código das superclasses abstratas. Em resumo, *frameworks* facilitam o reuso, capturando estratégias de desenvolvimento de *software* bem sucedidas [14].

Existem dois tipos de *frameworks*: caixa-branca e caixa-preta [23]. No *framework* caixa-branca o reuso é provido por herança, ou seja, o usuário deve criar subclasses das classes abstratas contidas no *framework* para criar aplicações específicas. Para tal, ele deve entender detalhes de como o *framework* funciona para poder usá-lo.

Já no *framework* caixa-preta o reuso é por composição, ou seja, o usuário combina diversas classes concretas existentes no *framework* para obter a aplicação desejada. Assim, ele deve entender apenas a interface para poder usá-lo.

Um ponto variável de um domínio de aplicação é chamado de “ponto de especialização” (em inglês, “*Hot spot*”). Diferentes aplicações dentro de um mesmo domínio são distinguidas por um ou mais *hot spots*. Eles representam as partes do *framework* de aplicação que são específicas de sistemas individuais.

Os *hot spots* são projetados para serem genéricos e podem ser adaptados às necessidades da aplicação. A Figura 6 ilustra um *framework* caixa-branca com um único *hot spot R*. Para utilização do *framework* é necessário fornecer a implementação, através de especialização, referente a esse *hot spot*, que no caso da Figura 6 é denominado *R3*.

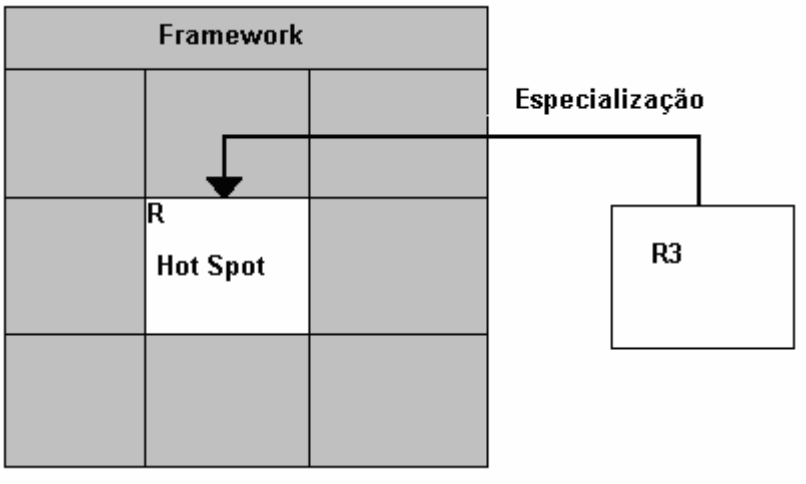

FIGURA 6 - *FRAMEWORK* CAIXA-BRANCA

Já a figura 7 ilustra um *framework* caixa-preta, também com um único *hot spot R*. Nesse *framework*, existem três alternativas (*R1*, *R2* e *R3*) para implementação da responsabilidade *R*. O usuário deve escolher uma delas, através de instanciação, para obter sua aplicação específica. Note que *R1, R2* e *R3* fazem parte do *framework* e são as únicas alternativas possíveis de implementação do *hot spot*. Já no caso da Figura 6, *R3* não fazia parte do *framework*, mas, em compensação, qualquer alternativa de implementação do *hot spot* seria possível.

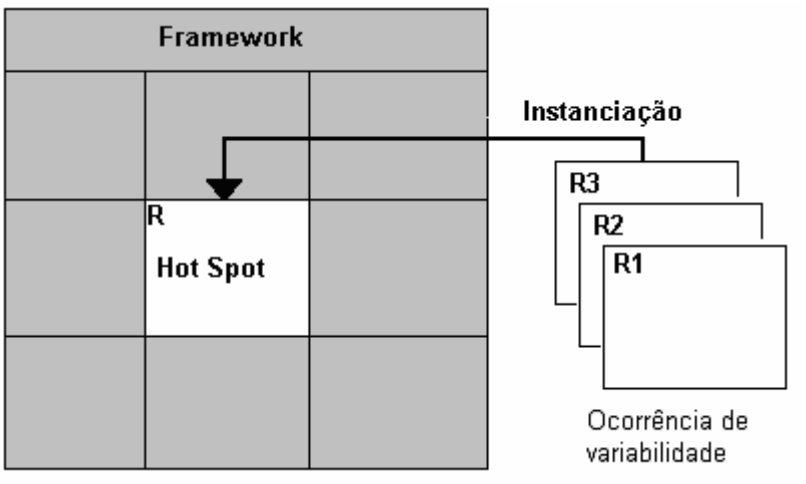

FIGURA 7 - *FRAMEWORK* CAIXA-PRETA

Existem, também, os chamados “Pontos fixos” (em inglês, “*Frozen spots*”), que definem a arquitetura geral de um sistema de *software*, seus componentes básicos e os relacionamentos entre eles. Os *frozen spots* permanecem fixos em todas as instanciações do *framework* de aplicação.

Dessa forma, um *framework* caixa-branca é mais fácil de projetar, pois não há necessidade de prever todas as alternativas de implementação possíveis. Já o *framework* caixa-preta é mais difícil de projetar por haver a necessidade de fazer essa previsão. Por outro lado, o *framework* caixa-preta é mais fácil de usar, pois basta escolher a implementação desejada, enquanto no caixa-branca é necessário completar essas implementações. *Frameworks* caixa-branca podem evoluir a tal ponto, passando a ser considerados caixa-preta [36]. Isso pode ser conseguido de forma gradativa, implementando-se várias alternativas que, depois, são aproveitadas na instanciação do *framework*. Ao mesmo tempo, não se fecha totalmente o *framework*, permitindo ao usuário continuar usando-o como caixa-branca. Após um certo tempo, estarão disponíveis diversas alternativas e, então, se pode decidir tornar o *framework* caixa-preta de fato. Os *frameworks*, que possuem as duas características citadas (caixa-branca e caixa-preta), são denominados caixa-cinza.

Segundo um levantamento realizado por *Yassin at el.* [45], a tendência atual é pelo desenvolvimento de *frameworks* caixa-cinza. Apesar dos *frameworks* caixa-branca serem mais fáceis de desenvolver, a exposição do código aos desenvolvedores pode causar problemas. Já os *frameworks* caixa-preta são mais abstratos e menos flexíveis, tornando difícil a manutenção das aplicações derivadas. *Frameworks* caixa-cinza bem projetados superam os problemas impostos pelos outros dois tipos, pois possuem flexibilidade e extensibilidade, sem expor desnecessariamente informações que não interessam ao desenvolvedor de sistemas.

Neste trabalho, o *framework* proposto é do tipo caixa-branca, com possibilidade de evoluir para caixa-cinza e, depois, caixa-preta, à medida que novas aplicações forem sendo desenvolvidas.

3.3.1 *JAVASERVER FACES*

A tecnologia *JavaServer Faces (JSF)* [26] foi tornada operacional através de um *framework* baseado em componentes, que define uma arquitetura e um conjunto de bibliotecas

para criação de aplicações *Java* para *web* no lado do servidor. Para isso, disponibiliza um padrão de *tags JSP* e classes *Java* que facilitam o desenvolvimento de formulários *HTML* e interfaces gráficas usando *Servlet* e *JSP* [29].

JSF dispõe de bibliotecas de "*tags libraries*" que são utilizadas na composição de páginas *JSP*. As *tags libraries* auxiliam na geração de conteúdo dinâmico em uma página *JSP*. Elas aparecem de duas formas: *html-basic* e *jsf-core*. A primeira define *tags* para representar componentes *HTML* de interface com o usuário e a segunda define outras *tags*, como para tratamento de eventos e validação de componentes [29].

A tecnologia *JSF* é baseada no padrão *MVC* (*Model*, *View*, *Control*), o qual permite estruturar a aplicação em três camadas: modelo, visualização e controle. A visualização representa a interface com o usuário e é realizada pelas páginas *JSP*. O controle é o ponto de entrada do *JSF*. É realizado por um *servlet* chamado *FacesServlet*, o qual recebe requisições *web* dos clientes, através de páginas *JSF*, delega-as para os componentes da camada modelo, e envia as respostas para os clientes [11]. Na camada modelo tem-se os componentes do modelo da aplicação.

3.4 COMPONENTES DE *SOFTWARE*

Segundo *D'Souza at el.* [10], componentes são artefatos de *software* construídos e disponibilizados de forma independente e que podem ser compostos com outros componentes para construir sistemas maiores e complexos.

Um componente de software pode ser reutilizado para a construção de diferentes sistemas de *software* [27]. A reutilização de *software* pode ser obtida através do desenvolvimento com componentes, ou seja, um novo sistema pode ser construído utilizando partes previamente construídas e testadas.

Componentes são estruturas de *software* que incluem implementação, com uma especificação de interfaces providas e requeridas. Uma interface provida especifica os

serviços realizados pelo componente. Um componente pode realizar várias interfaces, fornecendo um conjunto de métodos capazes de implementar adequadamente os serviços especificados na interface. Portanto, a conexão de um componente, para acesso aos serviços providos por outro componente se dá por meio das interfaces. Componentes são construídos de forma a atender uma "linha de montagem" de *software*, tal qual ocorre na indústria. Portanto, devem atender aos requisitos de padronização estabelecidos, atuando como unidades funcionais auto-suficientes, capazes de se comunicar com outras unidades e disponibilizar seus serviços somente através de suas interfaces-padrão [16].

Por se tratar de "blocos" pré-testados e reutilizáveis, os componentes podem ser conectados conforme ilustra o modelo da Figura 8, usando a notação *UML 2.0* [43]. Assim, por exemplo, *Componente1* disponibiliza seus serviços através da interface *Interface_11* para *Componente2*, e *Interface_12* para *Componente3*. Pela semântica da *UML 2.0*, *Interface_11* e *Interface_12* são interfaces requeridas por *Componente2* e *Componente3*, respectivamente.

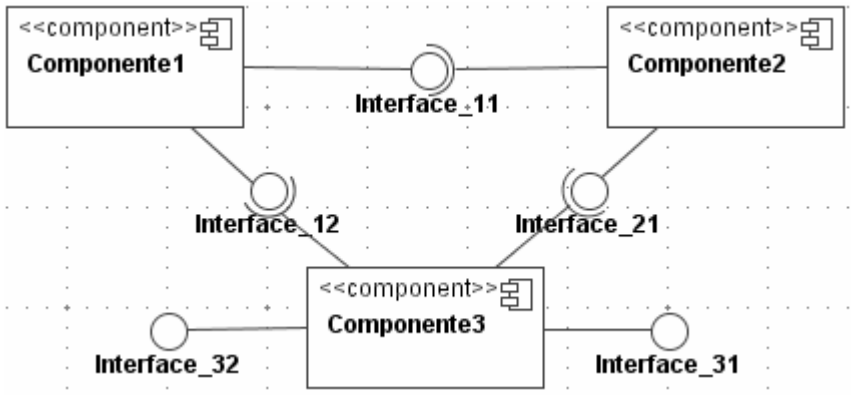

FIGURA 8 - CONEXÕES DE COMPONENTES DE *SOFTWARE*

Portanto, com os componentes pode-se construir sistemas complexos, que visam ao atendimento de uma necessidade de desenvolvimento em menos tempo e com melhores resultados. Considerando que os componentes são blocos de código previamente testados, suas utilizações podem reduzir, de forma significativa, os custos de manutenção e evolução contínua do *software*. Através da experiência com o reuso, podem-se refinar os componentes e também obter novos componentes.

3.4.1 *JAVABEANS* E *ENTERPRISE JAVABEANS*

JavaBeans [18] [19] são componentes de *softwares* escritos na linguagem de programação *Java*, especificado pela *Sun Microsystems* [19]. Um componente *JavaBeans* deve obedecer a certas convenções de construção, comportamentos e nomeação de métodos. Essas convenções possibilitam o uso e reuso desses componentes. *JavaBeans* não devem ser confundidos com *Enterprise JavaBeans* [18].

Segundo *White at el.* [44], *Enterprise JavaBeans* representa um modelo para criação de componentes distribuídos, do lado servidor, especificado pela *JCP* [24]. Enterprise *JavaBeans 3.0 (EJB3.0)* [24], simplifica a implementação das camadas intermediárias e fornece suporte para o controle transacional, escalabilidade, conexão com banco de dados, segurança, portabilidade e reutilização de componentes.

Existem três tipos de *Enterprise JavaBeans 3.0*: "*Entity Beans*", que representam os objetos persistentes em um meio de armazenamento como um banco de dados, "*Session Beans*", que implementam as lógicas de negócios e são utilizados para tarefas de processamento isolado e distribuído, e finalmente os "*Message-Driven Beans*", responsáveis por processar mensagens assíncronas [24].

De acordo com a especificação *EJB3.0* um componente possui os seguintes elementos: Classe Abstrata (não pode ser instanciada) e uma ou mais Interfaces. Esses elementos são construídos utilizando anotações *Java* (*java metadata annotations*) [25], responsáveis por especificar as semânticas e os requerimentos de serviços ao *EJB container*, e de prover informações estruturais e de configurações para as aplicações.

No próximo capítulo, serão apresentados alguns trabalhos correlatos, fazendo uma análise comparativa entre os mesmos.

4 SISTEMAS E FRAMEWORKS – BIOLOGIA MOLECULAR

Neste capítulo serão apresentados sistemas e *frameworks* disponíveis no domínio de biologia molecular. São apresentadas suas arquiteturas, uma descrição sucinta de suas principais funcionalidades e uma análise das suas características.

Os bancos de dados de genoma representam hoje uma das principais ferramentas de suporte para os biólogos moleculares e geneticistas. Para que esses bancos de dados possam ser realmente utilizados na prática, é necessário que existam sistemas que façam análises, filtragens e tratamentos desses dados bem como mecanismos de extração e pesquisas, incluindo as necessidades de processamento, as análises e controles semânticos dos dados, meios de acesso e o problema da integração dessas bases de dados. Os sistemas e *frameworks* apresentados a seguir tentam solucionar a maioria desses problemas.

4.1 SISTEMAS

Existem diversos sistemas voltados para o acesso e manipulação de informações do domínio de biologia molecular, tais como:

- *SRS* (*Sequence Retrieval System*) [40]

O sistema *SRS* integra diversas fontes de dados textuais, contendo informações biológicas através da implementação de *links* entre objetos que compõem estas fontes. O sistema permite, ainda, o uso de algumas poucas aplicações para análise de dados (por exemplo, para comparação de seqüências via *BLAST*), todas elas referentes aos objetos: seqüência de nucleotídeos e seqüências de aminoácidos.

A ferramenta não fornece transparência de esquema e nem de localização, obrigando os usuários a formular consultas sobre atributos de uma dada fonte de dados. O sistema disponibiliza um reduzido conjunto de aplicativos de suporte à pesquisa.

- *CPL/Kleisli* (*Collection Programming Language*) [9]

O sistema *CPL*/*Kleisli* é resultante de um projeto acadêmico que consiste de um sistema de consultas a múltiplos bancos de dados. O *CPL/Kleisli* tem primitivas que implementam chamadas ao *wrapper* do banco de dados a ser acessado, no sentido de executar operações do tipo seleção, projeção e junção. O acesso a fontes de dados textuais é bem mais difícil de ser realizado uma vez que não há uma linguagem para tal. Assim, foi criada uma sintaxe para extração de dados através de sucessivos acessos a registros de forma a implementar as operações.

De forma resumida, o sistema *CPL/Kleisli* oferece as seguintes funcionalidades:

- Disponibiliza os tipos de dados necessários para a integração de recursos de dados distribuídos e heterogêneos da biologia molecular;
- Provê uma interface uniforme para os recursos de dados heterogêneos;
- Permite a formulação de consultas complexas ao ambiente distribuído e heterogêneo;
- Provê a otimização de consultas distribuídas, incluindo paralelismo;
- Provê tratamento uniforme para os algoritmos de análises empregados na biologia (por exemplo, *BLAST*);
- Sua arquitetura permite a modularização dos *wrappers* para acesso aos sistemas distribuídos.

- *TAMBIS* (*Transparent Access to Multiple Bioinformatics Information Sources*) [42]

O projeto *TAMBIS* desenvolvido na Universidade de *Manchester*, consiste de um sistema de recuperação de informações biológicas que é orientado por uma ontologia construída para o domínio da biologia molecular e bioinformática. O sistema foi construído em camadas: a primeira é a conceitual, a segunda trata do mapeamento entre as camadas conceitual e física e a terceira, a camada física.

O sistema *TAMBIS* tem as seguintes características:

- Provê total transparência para os usuários na formulação de consultas, ou seja, não é exigido dos usuários o conhecimento dos esquemas e da localização das fontes de dados;
- Provê acesso de leitura às fontes de dados através de *wrappers*;
- Permite a formulação de consultas complexas sobre múltiplas fontes de dados heterogêneas. Os usuários dispõem de ferramenta gráfica de forma a facilitar a formulação de consultas por usuários não especialistas. A ferramenta transforma a consulta para a linguagem *CPL*;
- Dispõe de uma ferramenta visual para consulta aos dados;
- Dispõe de uma ontologia composta de mais de mil e oitocentos conceitos biológicos e respectivos relacionamentos e que é capaz de inferir outros. A ontologia engloba conceitos referentes a ácidos nucléicos, proteínas, estruturas de proteínas, funções biológicas e processos biológicos.

- *ESTIMA* (*Expressed Sequence Tag Information Management and Annotation*) [28]

ESTIMA é um sistema que provê um esquema de banco de dados para gerenciamento de dados e anotações *EST (Expressed Sequence Tag)*, juntamente com um conjunto de ferramentas baseadas na *internet* que facilitam vários aspectos de consulta, visualização, pesquisa via *Blast* e classificação funcional baseado em um vocabulário definido por uma Ontologia de genes. O esquema de banco de dados do sistema *ESTIMA* provê aos usuários a flexibilidade de referenciar vários bancos de dados externos, para fazer anotações *EST*. Esta flexibilidade no esquema é crítica, frente ao crescente número de bancos de dados de seqüências genômicas.

As principais características do sistema *ESTIMA* são:

- É independente do *pipeline* de processamento *EST*;

- Possibilita múltiplas instalações, pois cada instalação possui seus próprios arquivos de configuração; e
- Mantém múltiplos projetos e suporta múltiplas bibliotecas em um mesmo projeto.

- *ESTAP* (*Expressed Sequence Tag Analysis Pipeline*) [31]

O sistema *ESTAP* é um conjunto de procedimentos analíticos que automaticamente verifica, purifica, armazena e analisa dados *EST´s*. Utiliza uma base de dados relacional para armazenar dados de seqüências e resultados de análises, os quais facilitam as pesquisas para informações específicas e dados analíticos.

As principais características do sistema *ESTAP* são:

- Uma base de dados relacional – utiliza o banco de dados *Oracle 9i* para armazenar os dados;
- Conjunto de programas (*pipeline*) – os programas são desenvolvidos com a linguagem *C/Pro*C,* utilizando a plataforma *linux*. Estes programas é que são responsáveis pela análise e purificação dos dados *EST´*s;
- *Interface web* – o sistema *web* foi desenvolvido utilizando a tecnologia *Java* (*servlets* e *applets*), é através desta interface que o biólogo acessa os dados analisados e também pode submeter seqüências *EST´*s para serem processadas.

Contudo, esses sistemas têm limitações nas formas de acesso às fontes de dados heterogêneas, e oferecem reduzida flexibilidade para atender às necessidades dos biólogos.

4.2 *FRAMEWORKS*

Além dos sistemas apresentados, existem também *frameworks* que foram desenvolvidos para o domínio de biologia molecular, tais como:

- *Bio-AXS* [38]

É uma Arquitetura para Integração de Fontes de Dados e Aplicações de Biologia Molecular. Neste projeto propõe-se a utilização de um *framework* orientado a objetos para acesso e manipulação desses dados. O *framework* foi implementado na linguagem *Java*, especificamente *Java Beans*, por ser uma linguagem orientada a objetos e por possibilitar a portabilidade de código entre diferentes plataformas.O banco de dados utilizado é o *SGBD Oracle 9i*, pois possui um tipo de dado especial, chamado *XMLType*, que é utilizado para consultar e armazenar dados em *XML* [1] no banco. O acesso à ferramenta é disponibilizado através de uma interface *Web*.

O *Bio-AXS* é subdividido em 4 módulos conforme mostra a figura 18:

- Módulo Administrador – módulo de interação entre os usuários e a arquitetura;
- Módulo Capturador – módulo que trata da persistência de esquemas e de dados. O framework utiliza um *XMLSchema* no *SGBD ORACLE 9i* para armazenar seus dados;
- Módulo Conversor – módulo que tem como funcionalidade a conversão dos esquemas utilizados nas diversas fontes de dados para o padrão *XMLSchema* e dos dados para *XML*. Para realizar estas conversões são utilizados *wrapper*, que são conhecedores das fontes de dados a serem convertidas (esquema e dados);
- Módulo de *Interface* com os *Drivers* de Aplicação – este módulo é implementado da mesma forma que o Módulo Conversor, ou seja, cada *driver* é conhecedor do formato de saída dos dados e é responsável pela implementação das conversões necessárias.

- *BioSig* (*Bioinformatic System for Studying the Mechanism of Inter-Cell Signaling*) [33]

O *framework BioSig* integra inserção de imagens, anotações e hierarquia de abstração de imagens. Tudo isso para criar uma base de dados, orientada a objetos, com esses

elementos. Além disso, o *framework* possui *plug-in* com ferramentas de estatísticas e visualização para permitir testes de hipóteses e mineração de dados.

O *framework BioSig* está dividido em camadas:

- Camada de banco de dados – para capturar anotações e dados de análises das imagens. Os resultados são gerados no formato *xml* para a camada de apresentação;
- Gerenciador de apresentação – para consulta e atualização dos dados. Interface responsável pela interação com os usuários e acesso ao banco de dados. Esta camada tem que ser flexível, com o objetivo de ser o mais simples possível para a manipulação do biólogo;
- Gerenciador de consulta – responsável pelo mapeamento em alto nível das consultas dos usuários para objetos *Java* que implementam o modelo de dados.

Embora esses *frameworks* tenham mais flexibilidade do que os sistemas anteriores, eles ainda têm limitações quanto ao reuso e acesso às diferentes fontes de dados distribuídas.

4.3 ANÁLISE DOS SISTEMAS E *FRAMEWORKS*

Para facilitar o entendimento das principais funcionalidades dos sistemas e *frameworks* apresentados, a Tabela 1 apresenta um resumo das características dos ambientes analisados. Através desse resumo, podem-se verificar as principais necessidades no desenvolvimento e manutenção de um sistema do domínio de biologia molecular.

Este resumo foi realizado com base na análise das documentações disponíveis para cada ambiente.

De acordo com a Tabela 1, as características analisadas de cada ambiente possuem 2 saídas: “S” - significa que o ambiente suporta a característica, embora não quantificada; “N” significa que o ambiente não suporta tal característica.

TABELA 1 – ANÁLISE DOS SISTEMAS E *FRAMEWORKS* EXISTENTES.

CARACTERÍSTICAS	SRS	CPL/ KLEISLI	TAMBIS	ESTIMA	ESTAP	BIOSIG	BIO-AXS
Acessa diversos bancos	S	S	S	S	S	S	S
Acessa diversos tipos de arquivos	S	S	S	S	N	N	S
Acessa ferramentas (*Blast*, ...)	S	S	N	S	N	N	S
Disponibiliza interface *web*	S	N	N	S	S	S	S
Desenvolvido baseado em componentes de *software*	N	N	N	N	N	N	S

O *framework BIO-AXS* é o que possui o maior número de características analisadas, mas ainda assim possui algumas limitações quanto ao reuso e padronização de seus componentes de *software*.

5 FRAMEEST: *FRAMEWORK* PARA O DOMÍNIO DE BIOLOGIA MOLECULAR

Nas seções anteriores foram apresentados conceitos, idéias e tecnologias utilizadas no desenvolvimento do FrameEST. Sua arquitetura, bem como seu processo de desenvolvimento, são apresentados a seguir.

5.1 ARQUITETURA DO *FRAMEWORK*

A arquitetura geral do FrameEST baseou-se no padrão *MVC*. A Figura 9 apresenta uma visão geral dessa arquitetura organizada segundo as três camadas do padrão *MVC*: Controle (*Control*), Visualização (*View*) e Modelo (*Model*).

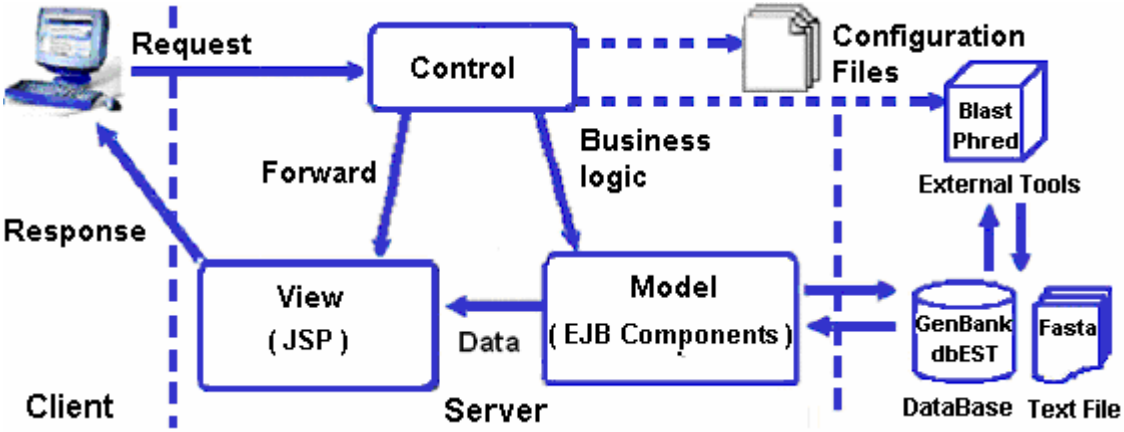

FIGURA 9 - ARQUITETURA DO FRAMEEST

O **Controle** é a porta de entrada por onde passam as requisições (*Request*) das aplicações clientes (*Client*). O controle provê os acessos aos componentes do Modelo, responsável pela lógica de negócios (*Business logic*). Utiliza um *servlet* denominado *FacesServlet*, integrado da tecnologia *JavaServer Faces* (*JSF*), para receber e direcionar as requisições das aplicações clientes para o modelo. É responsável ainda pelo encaminhamento (*Forward*) das respostas (*Response*) para o cliente na camada visualização. Utiliza arquivos *XML* (*Configuration Files*) para descrever: configurações de segurança de acesso às bases de dados; associações entre as páginas *JSP* e componentes *JavaBeans* (denominados *BackBeans*); as configurações que disponibilizam os componentes para reuso em um servidor

de aplicações. Conforme ilustra a Figura 9, através do Controle podem-se também utilizar as ferramentas disponíveis para extração, análise e formatação de dados do Modelo.

Na **Visualização** estão os componentes responsáveis pela construção das interfaces das aplicações. No FrameEST estes componentes foram importados da tecnologia *JSF*, e disponibilizados para o desenvolvimento de *JavaServer Pages* (*JSP*).

No **Modelo** estão os *BackBeans* responsáveis pelo gerenciamento dos componentes *JSF*, e os componentes *Enterprise JavaBeans* (*EJB3.0*) que manipulam informações das diferentes fontes de dados (*DataBase, Text File*) do domínio de biologia molecular. Os componentes *EJB3.0* dessa camada analisam e formatam os dados, que compõem as respostas para as aplicações clientes na camada Visualização.

5.2 MODELAGEM

O FrameEST foi modelado com a tecnologia *UML 2.0* seguindo as idéias do método *Catalysis*: Definir Domínio do Problema, Especificar Componentes, Projetar Componentes e Implementar Componentes.

No passo Definir Domínio do Problema foram identificados os requisitos e as regras de negócio do domínio de Biologia Molecular. Os requisitos foram obtidos a partir do estudo de vários ambientes desse domínio, entre eles: *SRS, CPL/Kleisli, TAMBIS, ESTIMA, ESTAP, Bio-AXS* e o *BioSig*. O capítulo 4 mostra o estudo realizado nesses ambientes.

Reuniões e entrevistas, também foram empregados no levantamento e identificação dos requisitos. Todas as experiências foram importantes para conhecer o domínio de Biologia Molecular, facilitando o desenvolvimento do *Framework*.

Nesta primeira etapa da modelagem, os principais modelos utilizados foram os de Colaborações e Casos de Uso. Os requisitos identificados foram inicialmente especificados com base nas regras de negócio que expressam o entendimento do domínio do problema. Conforme mostra a figura 10, devido ao grande volume de requisitos levantados e para uma

melhor compreensão e organização dos mesmos, foram definidas duas ações principais para o ator biólogo (*Biologist*): *Uso Funcional e Operacional*. A primeira (*Functional Use*), se refere aos requisitos específicos do domínio de biologia molecular e a segunda (*Operational Use*), se refere aos requisitos de estrutura e controle do *framework*, a parte estática do ambiente.

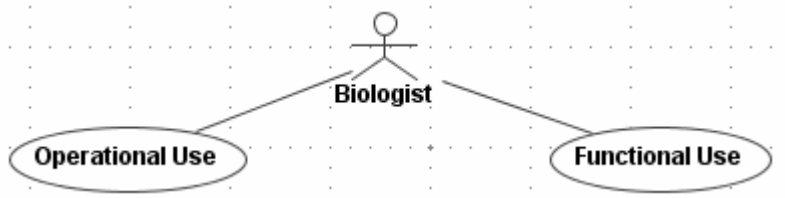

FIGURA 10 – MODELO DE COLABORAÇÃO

Através do refinamento dos modelos de colaborações, buscando um melhor entendimento das funcionalidades do *framework*, especificaram-se os casos de uso. A figura 11 apresenta um dos modelos de Casos de Uso que especifica a parte funcional do *framework*.

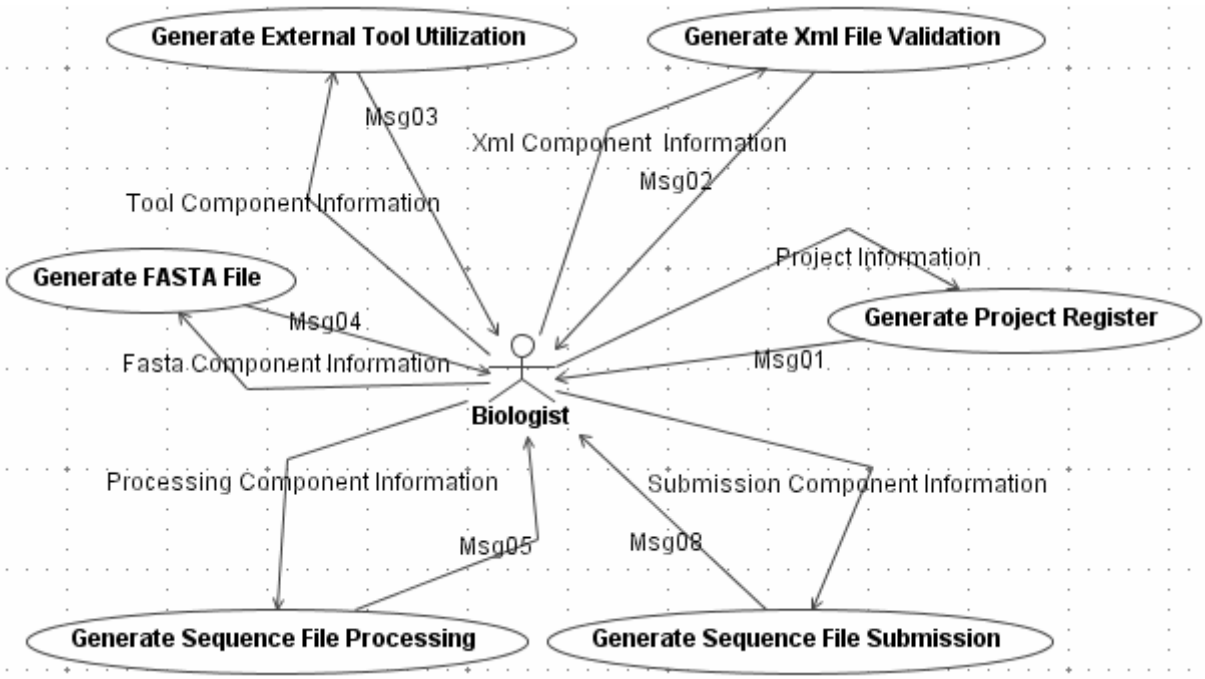

FIGURA 11 – UM DOS MODELOS DE CASOS DE USO: PARTE FUNCIONAL

Conforme ilustra a figura 11, o ator Biólogo (*Biologist*), interage para gerar o componente de: validação de arquivos no formato *Xml* (*Generate Xml File Validation*), registro de informações dos Projetos (*Generate Project Register*), submissão de arquivos de seqüências (*Generate Sequence File Submission*), processamento de arquivos de seqüência

(*Generate Sequence File Processing*), arquivos no formato *FASTA* (*Generate FASTA File*) e utilização de ferramentas externas (*Generate External Tool Utilization*).

Outro modelo de casos de uso refere-se à parte operacional do *framework*, ou seja, responsável pela interação entre o biólogo e o *framework*, com o objetivo de disponibilizar um ambiente amigável de desenvolvimento das aplicações. Conforme mostra a figura 12, o ator Biólogo (*Biologist*) pode iniciar uma aplicação (*Initiate Application*), criar uma visualização (*Create Application View*) ou uma conexão a um banco de dados (*Create Application Connection*) para a aplicação iniciada através do acesso a *templates* pré-estabelecidos (*Access Template*). O Biólogo pode ainda acessar os componentes do *framework* (*Access Component*) de duas formas: instanciação (*Instance Component*) ou especialização (*Specialize Component*). Para disponibilizar as aplicações desenvolvidas com o FrameEST, o biólogo faz o *deploy* da aplicação (*Deploy Application*), isto é, torna a aplicação disponível para uso.

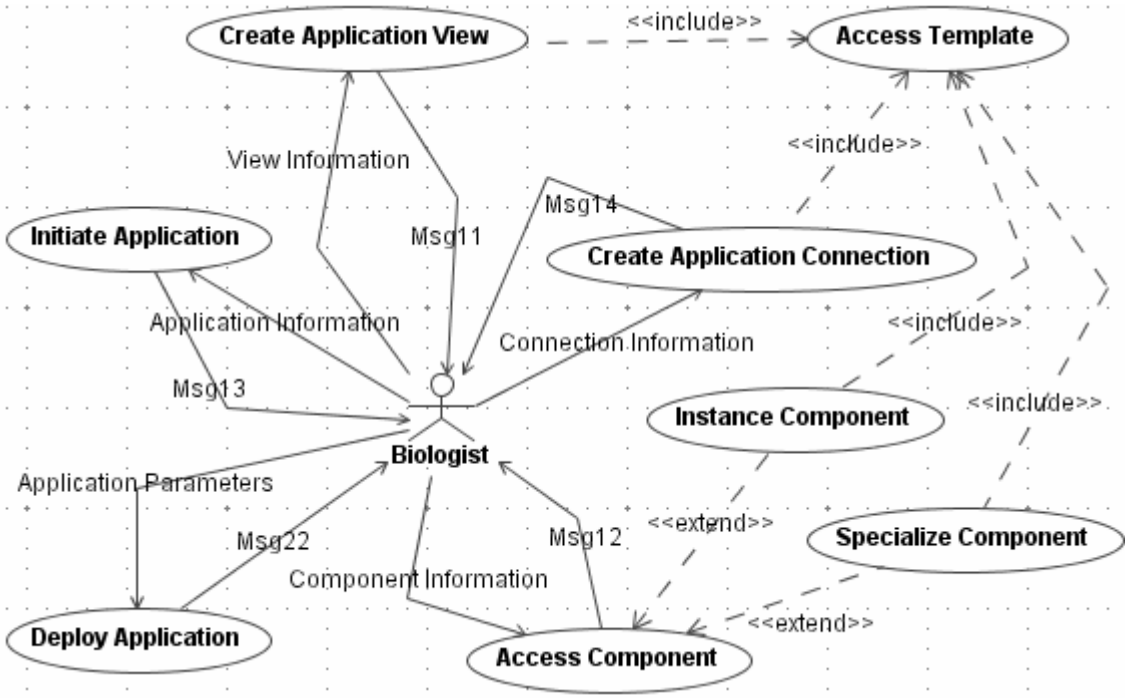

FIGURA 12 - UM DOS MODELOS DE CASOS DE USO: PARTE OPERACIONAL

Após especificar os casos de uso, inicia-se o segundo passo de *Catalysis*, Especificar Componentes, onde foram definidos os comportamentos externos dos

componentes, com suas responsabilidades e escopos, operações e interfaces. Um dos modelos especificado nesta etapa é o de Tipos, que descreve os objetos com seus principais atributos, relacionamentos e dependências. Os tipos estão relacionados através de associações, agregações ou herança, com as respectivas cardinalidade, que expressam as ocorrências mínimas e máximas de objetos participantes de um relacionamento.

A figura 13 apresenta um dos modelos de tipos referente à parte funcional do *framework*. Os *placeholders* (< >) indicam quais tipos podem ser reutilizados nas aplicações desenvolvidas a partir do *framework*.

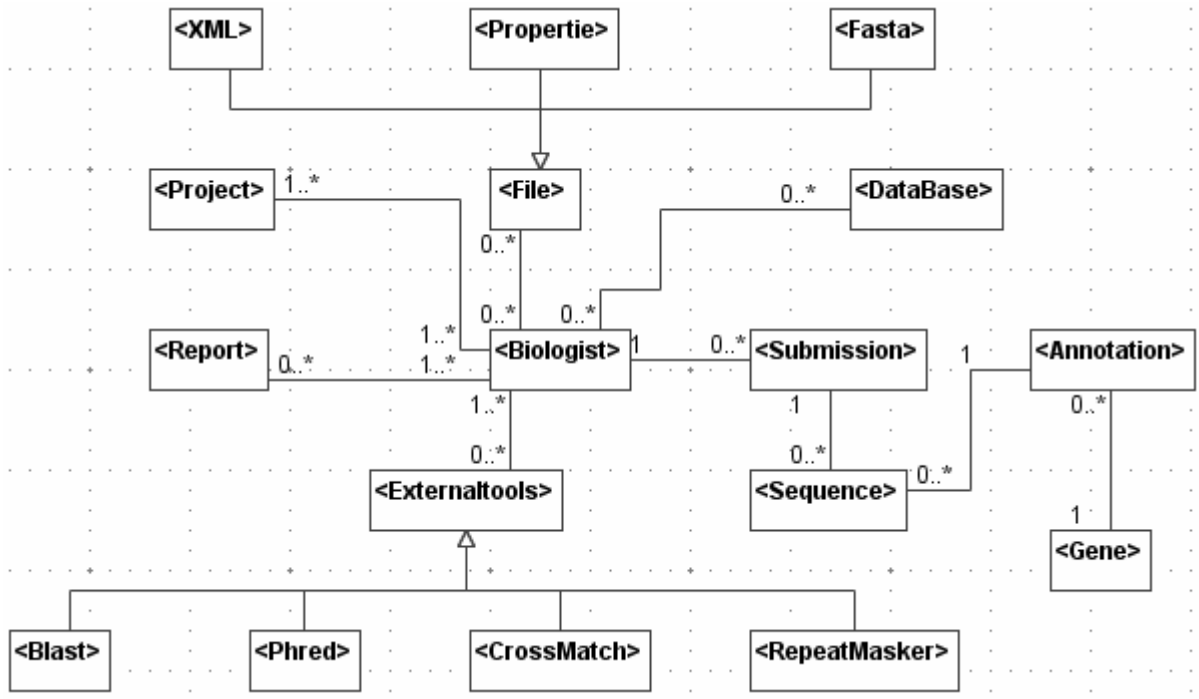

FIGURA 13 – UM DOS MODELOS DE TIPOS: PARTE FUNCIONAL

Nessa figura o tipo <*Biologist*> possui um relacionamento com os tipos:

- <*Externaltools*>: tipo que possui uma generalização, isto é, ferramentas externas que pode ser um <*Blast*>, <*Phred*>, <*CrossMatch*> ou <*RepeatMasker*>;
- <*Report*>: relatórios diversos requisitados pelos biólogos;
- <*Project*>: configuração do projeto da aplicação;
- <*DataBase*>: acesso a vários bancos de dados;
- <*Submission*>: submissões de arquivos de seqüências de *DNA*;

- <*Sequence*>: seqüências de *DNA* que são analisadas pelos biólogos;
- <*Annotation*>: anotações referentes às seqüências de *DNA*;
- <*Gene*>: dados genéticos relativos às anotações das seqüências;
- <*File*>: tipo que também possui uma generalização. Refere-se aos diferentes arquivos de dados e configurações nos formatos <*XML*>, <*Properties*> e <*Fasta*> disponíveis no domínio de biologia molecular.

Outro modelo especificado nesta etapa é o de Interações, representado em diagramas de seqüência, que detalham o comportamento dos casos de uso. A figura 14 apresenta o diagrama de seqüência referente ao Caso de Uso "*Generate Sequence File Submission*", curso normal.

Nesse diagrama o ator Biólogo (*Biologist*) interage com o <*wizard*>, solicitando iniciar a aplicação. A aplicação (<*Application*>) é criada e os componentes do FrameEST são carregados e disponibilizados para reuso. Uma mensagem de confirmação é enviada ao biólogo (msg13). Após iniciar a aplicação, o biólogo solicita acessar o componente de submissão de arquivo de Seqüências de *DNA*, processo que também é realizado através de um objeto <*wizard*> que cria um componente *javabean* (<*Component*>) baseado em um *template* (<*Template*>). Esse componente *javabean*, instancia o componente *ejb* desejado, no caso o componente de submissão (<*Submission*>), responsável por checar e salvar o arquivo de submissão (<*File*>). No final do processo é enviada uma mensagem de confirmação para o biólogo (*Msg08*).

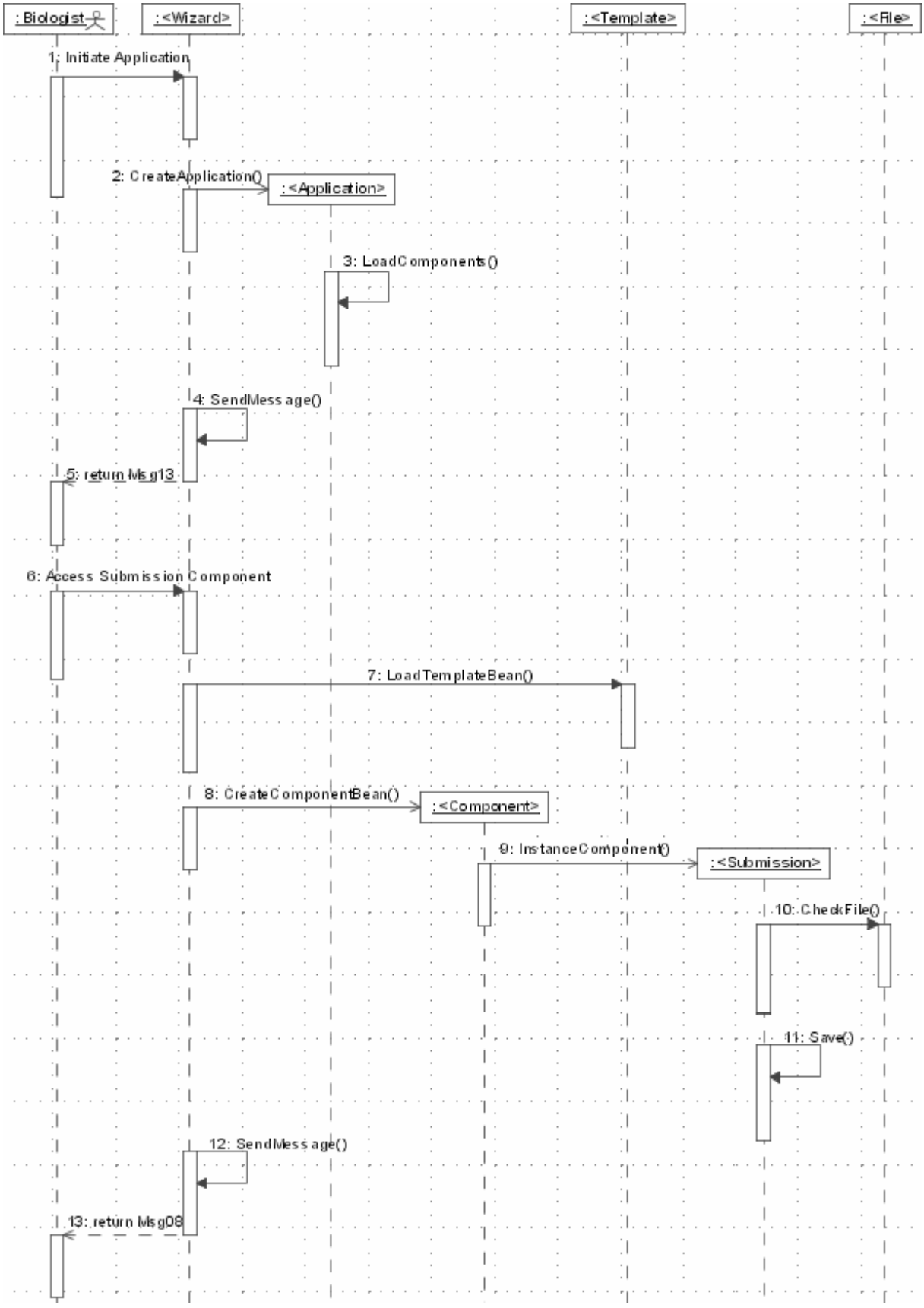

FIGURA 14 – DIAGRAMA DE SEQÜÊNCIA: *GENERATE SEQUENCE FILE SUBMISSION*

Refinando-se os modelos de tipos e de seqüências e considerando os comportamentos dos casos de uso obtém-se o terceiro passo da modelagem, Projetar Componentes. A partir dessa etapa, a análise e a modelagem do *framework* passaram a considerar o padrão *MVC*. Dessa forma, os componentes do *framework* ficaram organizados em 3 pacotes: modelo (*model*), visualização (*view*) e controle (*control*). A metodologia utilizada foi a seguinte: componentes com interações (diretas ou indiretas) com bancos de dados ficam na camada *modelo*; componentes que possuem interações com a parte de apresentação das aplicações ficam na camada *visualização* e na camada *controle*, ficam os componentes que controlam o fluxo de dados e estrutura do *framework*.

Considerando esses pontos e prosseguindo com a modelagem, dentro dos modelos desta etapa, Projetar Componentes, destacam-se os Modelos de: Classes de componentes, Componentes, e Pacotes de componentes.

A figura 15 mostra um dos Modelos de Classes do *framework*, com seus relacionamentos e interfaces.

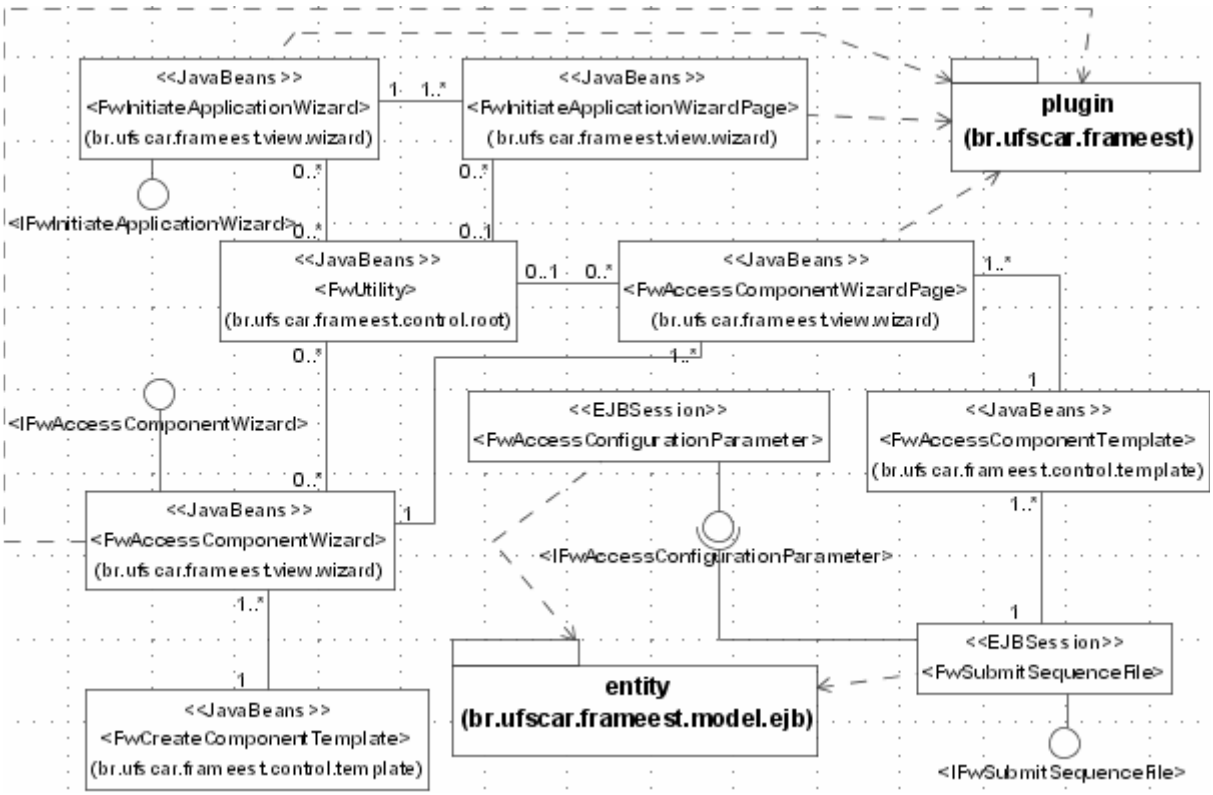

FIGURA 15 - MODELO DE CLASSES DO FRAMEEST: *SUBMIT SEQUENCE FILE*

Interfaces do tipo *Wizards* foram especificadas para tornar mais amigáveis as interações no reuso dos componentes do FrameEST. Assim, por exemplo, no modelo da figura 15 têm-se as classes e interfaces que suportam a construção de aplicações que realizam a submissão de arquivos de seqüências genômicas. Interagindo com o *wizard* provido pela interface *IFwInitiateApplicationWizard* (da classe *FwInitiateApplicationWizard*) o biólogo, inicialmente, faz a iniciação das aplicações. Em seguida, através do *wizard IFwAccessComponentWizard*, acessa as classes *FwCreateComponentTemplate, FwUtility, FwAccessComponentWizardPage* e *FwAccessComponentTemplate* para construção da aplicação. A classe *FwSubmitSequenceFile* provê os serviços para submissão de arquivos de seqüências genômicas através de sua interface *IFwSubmitSequenceFile*. Além dessas, reutiliza-se outras classes dos pacotes *br.ufscar.frameest.plugin* e *br.ufscar.frameest.entity*, indicado pela dependência (seta tracejada).

A partir do Modelo de Classes obtêm-se o modelo de Componentes, que representa a arquitetura física dos componentes, com suas interfaces providas e requeridas. A figura 16 mostra um exemplo de modelo de componentes do FrameEST obtido a partir do modelo de classes da figura 15.

Nesse modelo o componente *FwSubmitSequenceFileBean*, constituído das classes *FwSubmitSequenceFile, FwUser, FwQueue e FwSubmission*, disponibiliza seus serviços através da interface *IFwSubmitSequenceFile*, e utiliza os serviços providos pela interface *IFwAccessConfigurationParameter* do componente *FwAccessConfigurationParameterBean* (constituído das classes *FwAccessConfigurationParameter, FwConfProject* e *FwConfApplication*).

Padrões de projetos foram utilizados visando obter componentes mais genéricos, com melhores desempenhos e mais fáceis de serem reutilizados. Assim o padrão *Business Delegate* possibilitou centralizar os acessos, ocultando a complexidade de comunicação

remota, aos componentes da camada de negócio. O padrão *Session Facade* provê os serviços para submissão de arquivos de seqüências genômicas através de sua interface *IFwSubmitSequenceFile*. O padrão *Composity Entity* foi adotado para facilitar a persistência de dados.

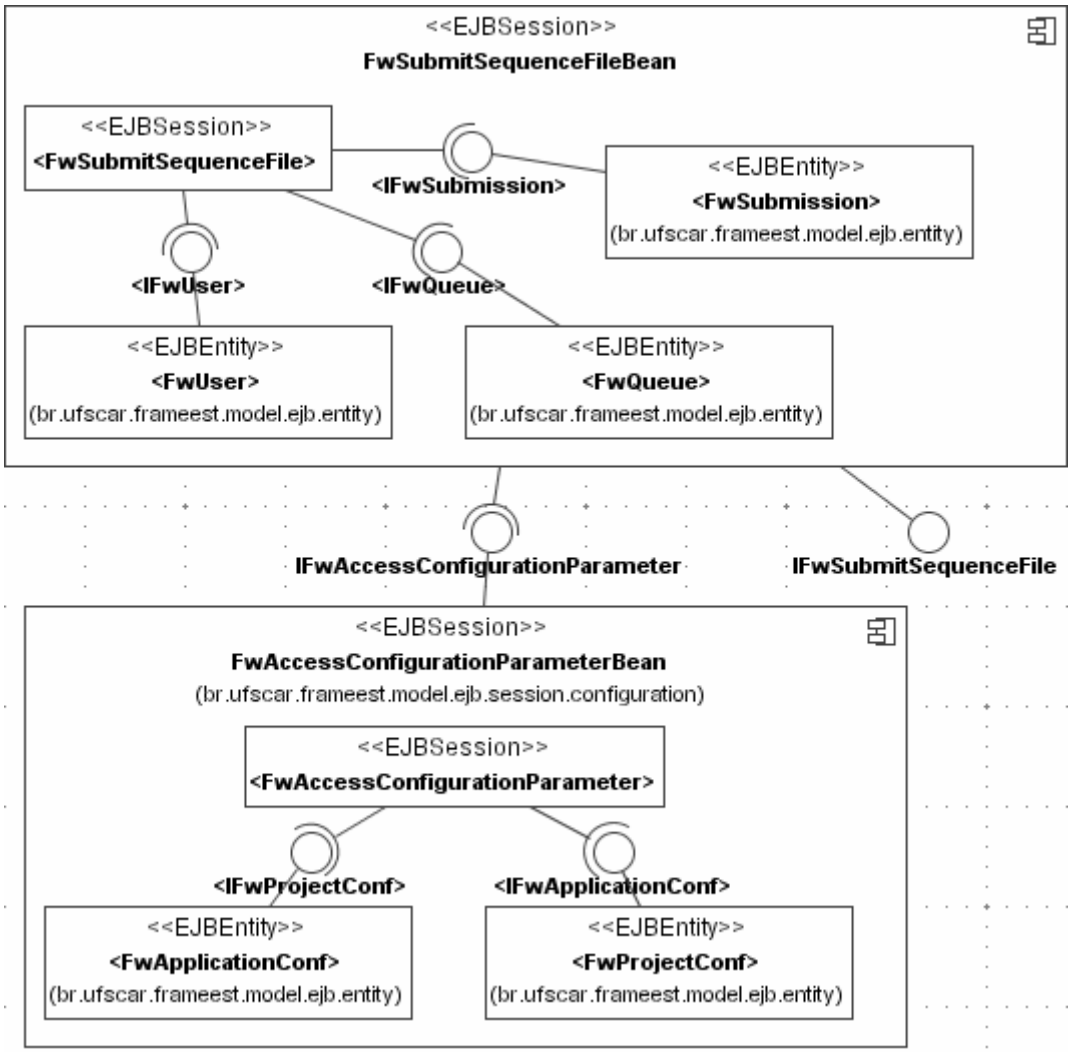

FIGURA 16 – MODELO DE COMPONENTES: *SUBMIT SEQUENCE FILE*

Para facilitar o reuso, os pacotes de componentes do FrameEST foram organizados conforme mostra a Figura 17. O pacote *br.ufscar.frameest.plugin* contêm componentes importados e reutilizados de *plug-ins* da plataforma *Eclipse* [13] [32], responsáveis pela construção da paleta de componentes e geração semi-automática de *Entity Beans EJB3.0* para diferentes *SGBDs* relacionais e relacionais estendidos, como no caso do

PostgreSQL [34], adotado neste projeto de pesquisa. Esse pacote inclui ainda outros componentes *JSF* [4] que auxiliam na construção de páginas *JSP*.

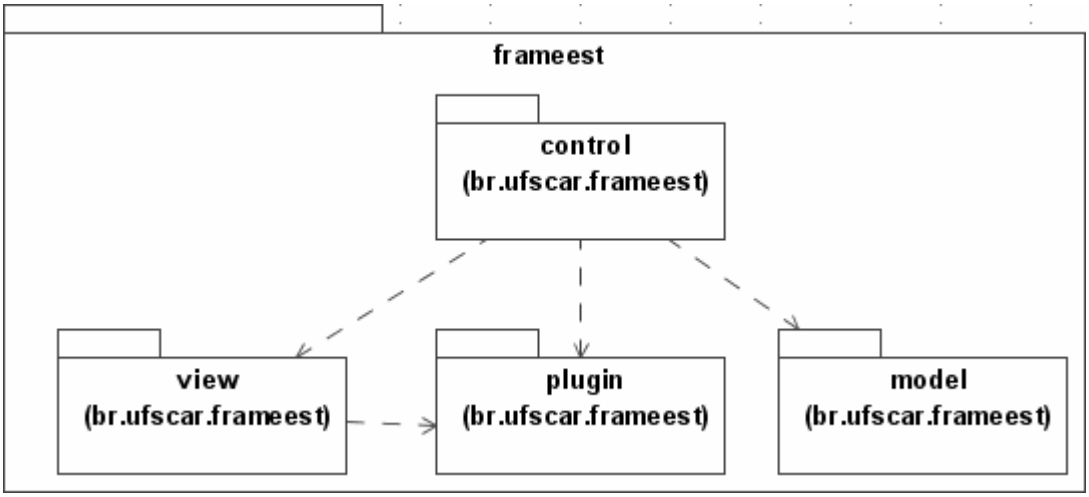

FIGURA 17 – MODELO DE PACOTES DO FRAMEEST

Além dos componentes reutilizados do pacote ***br.ufscar.framesest.plugin***, foram implementados em torno de 100 componentes que atuam nas 3 camadas do modelo *MVC*.

Esses componentes, desenvolvidos com este projeto de pesquisa, foram organizados nos seguintes pacotes:

- ***br.ufscar.frameest.view***: *JavaBeans*, responsáveis por gerar os editores textuais e gráficos do FrameEST. Utiliza também componentes *JSF* que suportam o desenvolvimento de páginas *JSP*;
- ***br.ufscar.frameest.control***: *JavaBeans* para controle e manipulação do *framework*;e
- ***br.ufscar.frameest.model***: *JavaBeans* e *Enterprise Java Beans (EJB3.0*) que representam os modelos do domínio de biologia molecular que compõem o *framewok*.

Esses pacotes contêm componentes dos tipos *Frozen Spots* e *Hots Spots*, sendo que o pacote ***br.ufscar.frameest.control*** é o que contém mais *Frozen Spots*, ou seja, mais componentes reutilizados por instanciação direta. Os componentes *Hots Spots* do pacote ***br.ufscar.frameest.view*** visam dar mais flexibilidade para atender às diferentes visões das aplicações. O pacote ***br.ufscar.frameest.model*** contém principalmente componentes *Hots Spots* que representam os diferentes modelos das aplicações do domínio genoma. Para uma

melhor compreensão esse pacote foi estruturado em outros pacotes, conforme mostra a figura 18.

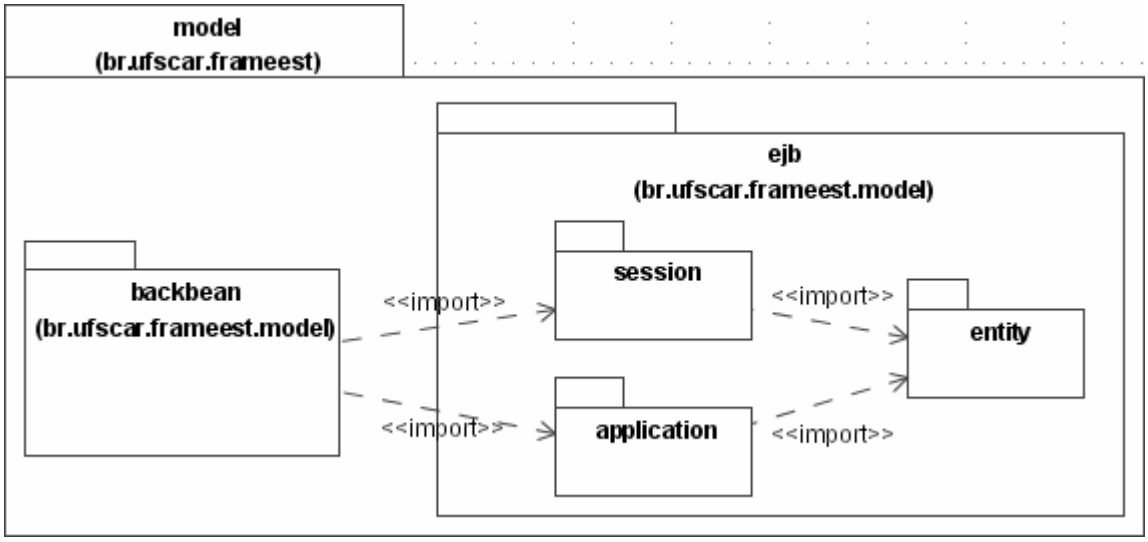

FIGURA 18 – MODELO DE PACOTES: CAMADA *MODEL*

O pacote ***br.ufscar.frameest.model.ejb.session*** contém os *EJB Session Beans* responsáveis pelas regras de negócio relacionadas com os casos de uso do domínio de biologia molecular. Outros componentes desse pacote são os responsáveis pela segurança, e manipulação dos *EJB Entity Beans*.

O pacote ***br.ufscar.frameest.model.ejb.entity*** contém os *EJB Entity Beans* com regras de negócios relacionadas principalmente com o armazenamento e recuperação das informações do domínio de biologia molecular e *SGBDs* relacionais ou relacionais estendidos.

O pacote ***br.ufscar.frameest.model.ejb.application*** contém outros componentes *EJB Sessions Beans* criados para atender casos específicos das aplicações.

Finalmente, o pacote ***br.ufscar.frameest.model.backbean*** contém os *BackBeans* que utilizam os serviços providos pelos componentes *EJB Session Beans* e *JSF*.

Existem ainda no FrameEST outros pacotes não apresentados nestes modelos de pacotes. Por exemplo, os pacotes *br.ufscar.frameest.view.wizard, br.ufscar.frameest.view.palette* e *br.ufscar.frameest.view.editors* contêm componentes responsáveis pela editoração textual e gráfica das aplicações, e os pacotes

br.ufscar.frameest.control.editors.editparts, br.ufscar.frameest.control.editors.models, br.ufscar.frameest.control.root, br.ufscar.frameest.control.template e br.ufscar.frameest.control.validator contêm componentes responsáveis pelo controle e gerenciamento do *framework*.

Baseado nessa modelagem fez-se o último passo da técnica de *Catalysis*, isto é, a implementação de um protótipo do FrameEST, apresentado a seguir.

5.3 IMPLEMENTAÇÃO

Na implementação do protótipo do FrameEST, foi utilizada a linguagem *Java*, com as tecnologias *JSF*, *JavaBeans*, e *EJB3.0*. Para persistência dos dados, foram utilizados o Sistema Gerenciador de Bancos de Dados (*SGBD*) relacional estendido *PostgreSQL*, arquivos de propriedades dos componentes (*Properties*) e arquivos texto no formato *XML*. O FrameEST possui um modelo de dados *default*, representado em forma de componentes *ejb entity*, gerados a partir do banco de dados *PostgreSQL*. O Modelo de Dados permite definir as tabelas do banco de dados que irá armazenar os dados das aplicações. Além disso, o *framework* tem a flexibilidade de gerar, automaticamente, novos componentes *ejb entity* a partir de qualquer banco de dados relacional. A figura 19 mostra o modelo de dados correspondente aos componentes *ejb entity* da camada modelo (*br.ufscar.frameest.model.ejb.entity*) do FrameEST.

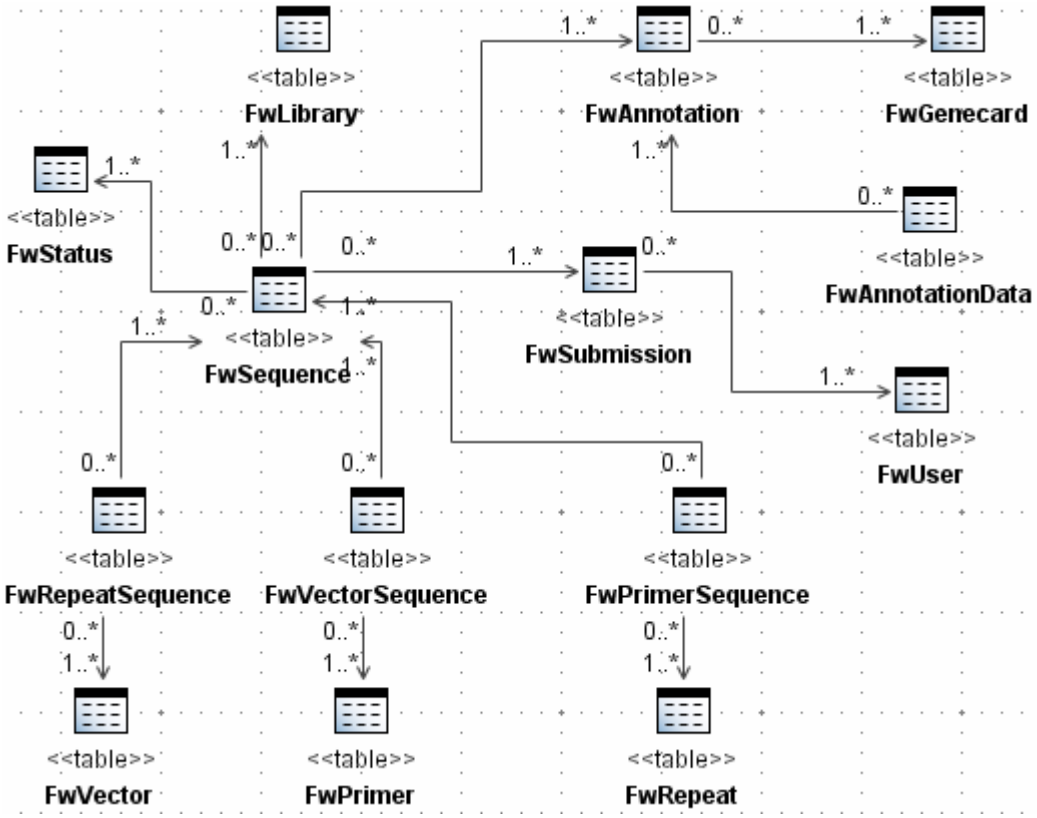

FIGURA 19 – MODELO DE DADOS

Na implementação do FrameEST também foram utilizadas funcionalidades dos *plug-ins Html Editor* [13] e componentes *JSF* do projeto *Apache MyFaces* [4].

Como mostra a figura 20, o *framework* foi implementado no ambiente *Eclipse* [12] em forma de um *plug-in*, com o objetivo de facilitar a sua disponibilização para reuso. No lado esquerdo da figura, são exibidos os pacotes do *framework*, organizados segundo o padrão *MVC* – controle (*br.ufscar.frameest.control*), visualização (*br.ufscar.frameest.view*) e modelo (*br.ufscar.frameest.model*).

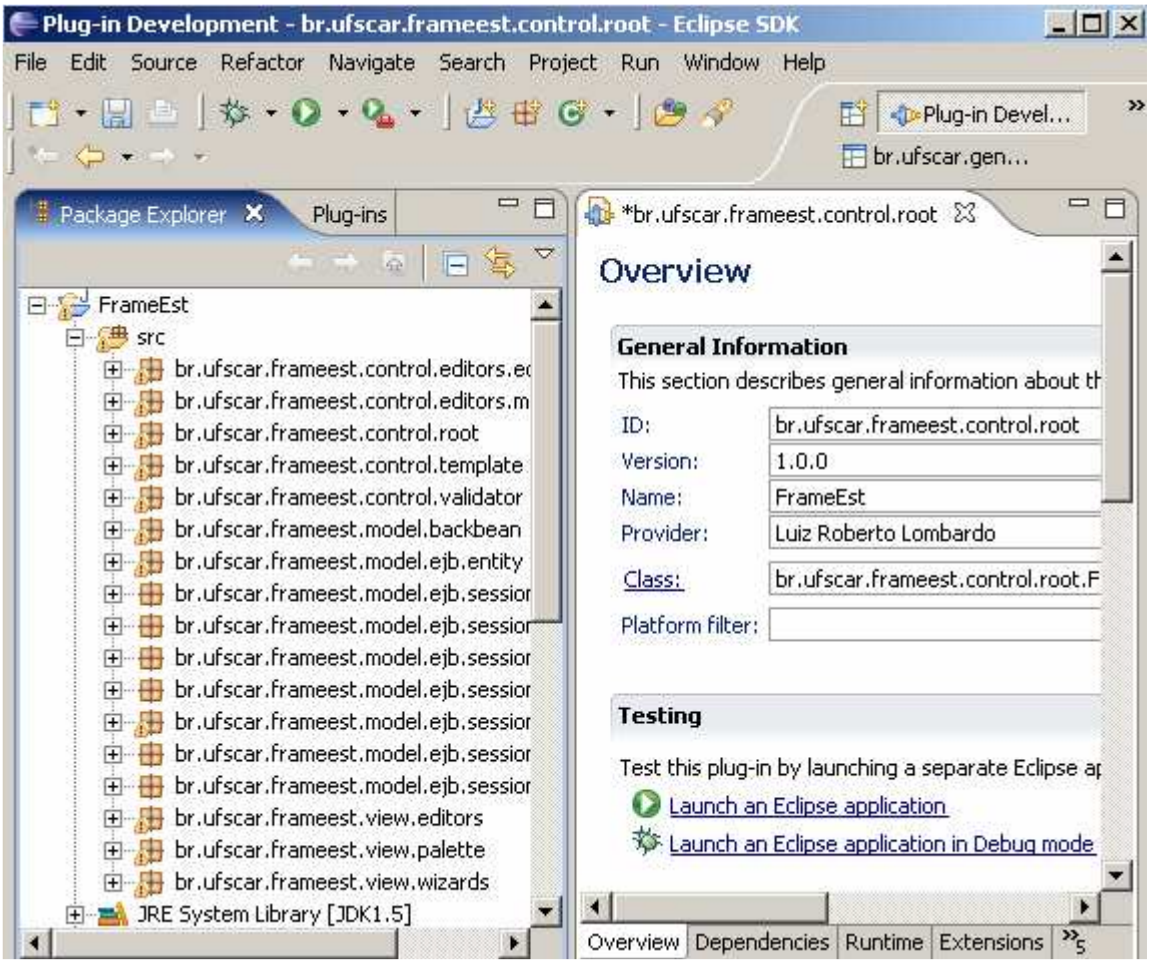

FIGURA 20: ESTRUTURA IMPLEMENTADA EM FORMA DE *PLUG-IN* DO FRAMEEST

Toda a estrutura do FrameEST, conforme exibido na figura 20, foi empacotada em um arquivo com a extensão "*.jar*", facilitando sua disponibilização para reuso. A figura 21 ilustra a estrutura desse arquivo, denominado *br.ufscar.frameest_1.0.0.jar*. A configuração dos componentes e as dependências dos *plug-ins* que o FrameEST utiliza, estão descritos nos arquivos *plugin.xml* e *plugin.properties*.

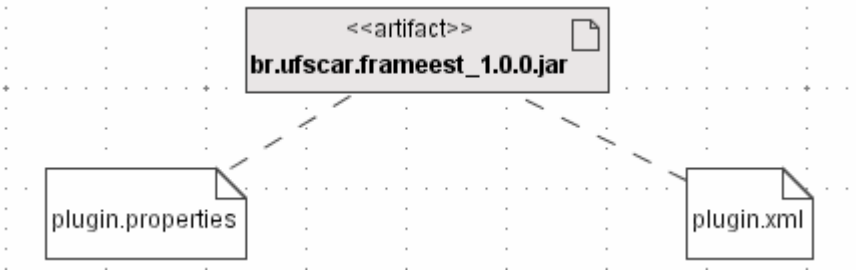

FIGURA 21 – ESTRUTURA DO ARQUIVO "*.JAR*" GERADO PELO *FRAMEWORK*

A próxima seção apresenta os dois níveis de reuso do FrameEST.

5.4 TRANSIÇÃO

Esta fase envolve a disponibilização do *framework* para o ambiente de produção e a realização de testes com os usuários, corrigindo defeitos encontrados e realizando treinamentos. O objetivo geral é garantir que o *framework*, bem como sua documentação, estejam disponíveis para reuso, sendo também necessário simular o ambiente do usuário para realizar um ajuste fino do produto com base no seu "*feedback*".

Uma vez construído, o FrameEST foi testado através do desenvolvimento de aplicações do domínio de biologia molecular. Para facilitar o reuso, no nível de modelagem das aplicações, o FrameEST foi disponibilizado em uma ferramenta *CASE* que utiliza o padrão *XMI* [43]. No caso foi utilizada a ferramenta *MagicDraw* [30]. A figura 22 ilustra a estrutura disponível para reuso nessa ferramenta. Para exemplificar, o componente <*FrameEstLogin*> está sendo reusado na modelagem de uma aplicação.

Após a modelagem, o FrameEST é reusado na implementação através de um *plug-in* para a ferramenta *Eclipse*, fornecendo um ambiente visual e amigável para o desenvolvimento das aplicações nesse domínio.

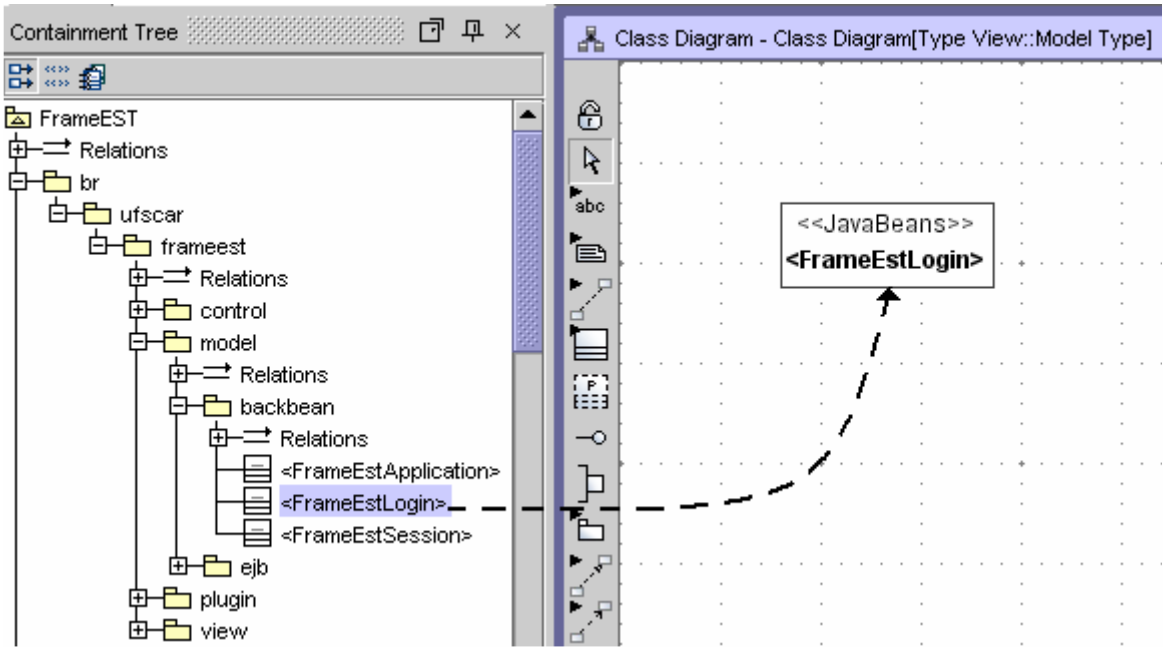

FIGURA 22 : ESTRUTURA PARA REUSO DO FRAMEEST : MODELAGEM

Conforme mostra a figura 23, o FrameEST disponibiliza seus pacotes de componentes de duas formas: através de estruturas de pacotes e paletas de componentes, demonstrados respectivamente no lado esquerdo e direito da figura.

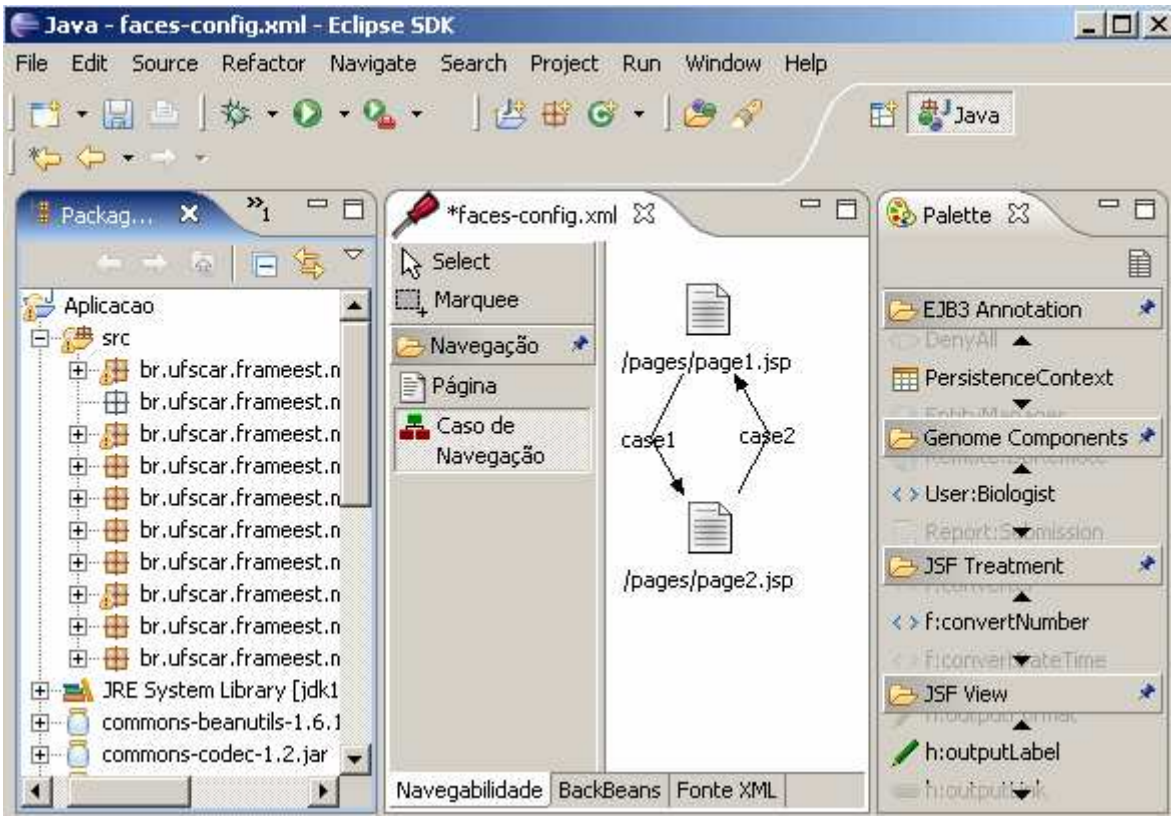

FIGURA 23: ESTRUTURA PARA REUSO DO FRAMEEST: IMPLEMENTAÇÃO

No Centro da figura 23 é exibido um dos editores visuais do FrameEST, responsável por construir a navegação das páginas das aplicações.

Para ilustrar o reuso do FrameEST, apresenta-se na próximo capítulo um estudo de caso, modelado e implementado com o uso do *framework*.

6 ESTUDO DE CASO COM REUSO DO FRAMEEST

Este capítulo apresenta uma estratégia definida por um conjunto de passos, para o desenvolvimento de aplicações, com o reuso do *framework* FrameEST. A Figura 24 mostra os passos dessa estratégia.

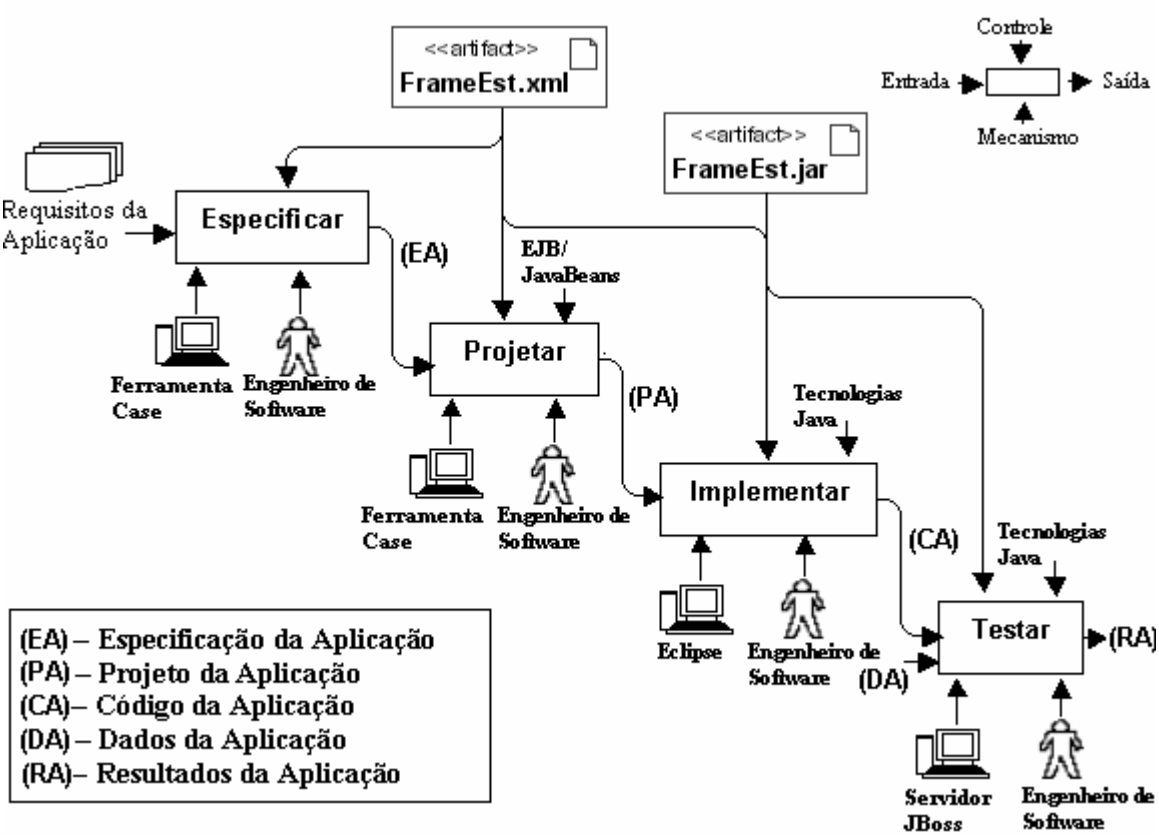

FIGURA 24 – PASSOS DA ESTRATÉGIA DE DESENVOLVIMENTO DE APLICAÇÕES

A estratégia é executada em 4 passos: Especificar, Projetar, Implementar e Testar. Os dois primeiros passos, considerados de modelagem, são desenvolvidos utilizando a tecnologia *UML 2.0*, auxiliada por uma ferramenta *CASE* que suporte o padrão *XMI*, no caso foi utilizado a *MagicDraw*. Partindo dos requisitos da aplicação, o Engenheiro de *Software* especifica a aplicação, utilizando os modelos de Casos de Uso. No próximo passo, as especificações são projetadas, analisando e reutilizando os componentes do *framework* (*FrameEst.xml*), resultando no projeto físico da aplicação. Em seguida, implementa o projeto com tecnologias *Java*, no ambiente *Eclipse* (*FrameEst.jar*), resultando o código da aplicação

e testes num ambiente *Java*, no caso o servidor *JBoss* [20]. Os resultados são verificados conforme os requisitos da aplicação.

A seguir, é apresentado um estudo de caso que mostra cada passo do desenvolvimento de uma aplicação: modelagem, implementação e testes. Trata-se de uma aplicação do projeto genoma do camarão *Litopenaeus vannamei* (*Shrimp EST Project*)[1], cujo objetivo é o sequenciamento de milhares de *EST* (*Expressed Sequence Tags*) deste camarão, a principal espécie de crustáceo cultivada atualmente no Brasil.

6.1 MODELAGEM

Para desenvolver o primeiro passo da estratégia adotada, *Especificar*, inicialmente são pesquisados e analisados os requisitos da aplicação, preocupando-se com suas regras de negócio, com os papéis dos atores e as colaborações para especificar o comportamento de grupo de objetos da aplicação. Nessa etapa da modelagem, o principal modelo de interações utilizado foi o de Casos de Uso. A tabela 2 mostra os principais casos de uso da aplicação, com uma breve descrição, suas entradas e saídas.

TABELA 2 – CASOS DE USO DA APLICAÇÃO *SHRIMP*

Nr	Descrição	Caso de Uso
1	Biólogo é autenticado para acessar a aplicação	*Realize Login*
2	Biólogo submete arquivos de seqüência	*Submit Sequence File*
3	Biólogo valida arquivo com conteúdo XML	*Validate Xml File*
4	Biólogo gera relatório de submissões de seqüência	*Generate Submission Report*
5	Biólogo usa ferramenta externa *CrossMatch*	*Use CrossMatch Tool*
6	Biólogo usa ferramenta externa *Phred*	*Use Phred Tool*
7	Biólogo usa ferramenta externa *Blast*	*Use Blast Tool*

[1] Genoma do Camarão Branco – www.shrimp.ufscar.br/

A Figura 25 mostra, por exemplo, o ator Biólogo (*Biologist*) interagindo com os casos de uso *Realize Login*, *Submit Sequence File*, *Validate Xml File*, *Generate Submission Report*, *Use CrossMatch Tool*, *Use Phred Tool* e *Use Blast Tool*. Neste nível do problema, os relacionamentos dos atores com os casos de uso são indicados dispensando detalhes que não são relevantes para o contexto da aplicação.

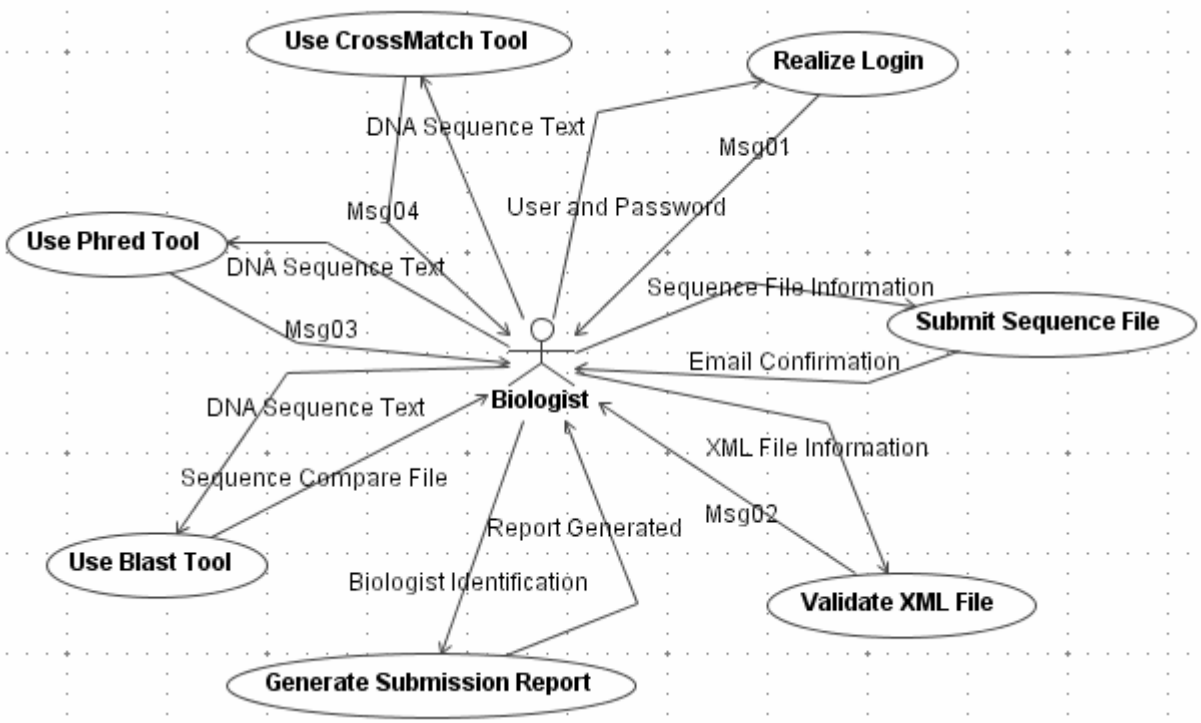

FIGURA 25 – MODELO DE CASO DE USO DA APLICAÇÃO *SHRIMP*

Prosseguindo na modelagem da aplicação especifica-se o modelo de classes com seus atributos, métodos e relacionamentos. O refinamento dos casos de uso possibilita determinar os comportamentos dos objetos em cada classe.

Orientado pelo padrão *MVC* da arquitetura do FrameEST, o desenvolvedor organiza os artefatos da modelagem conforme a estrutura de pacotes mostrada na figura 26. No pacote *br.ufscar.frameest.model* ficam os componentes dos modelos da aplicação. A figura destaca os pacotes *br.ufscar.frameest.model.backbean* e *br.ufscar.frameest.model.ejb.application* que conterão os componentes *BackBeans* e *EJB Session Beans* mais específicos da aplicação.

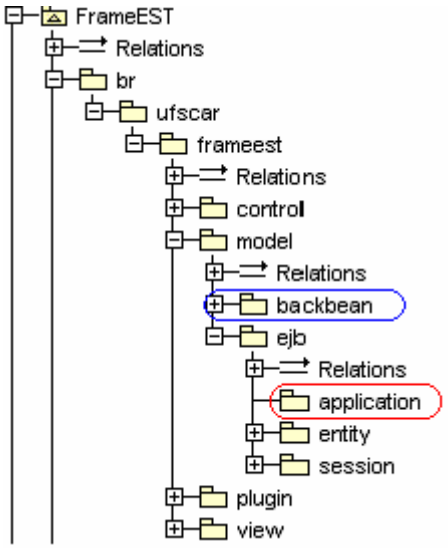

FIGURA 26 – ESTRUTURA DA APLICAÇÃO NO PADRÃO *MVC*

Baseado numa análise dos modelos de classes e de casos de uso da aplicação e do FrameEST, identificam-se as partes do FrameEST a serem reutilizadas. Nesse momento inicia-se o segundo passo da estratégia, *Projetar*. Nessa fase os componentes do FrameEST são reutilizados pelas aplicações. O modelo da figura 27 mostra a reutilização de 6 interfaces de componentes do FrameEST.

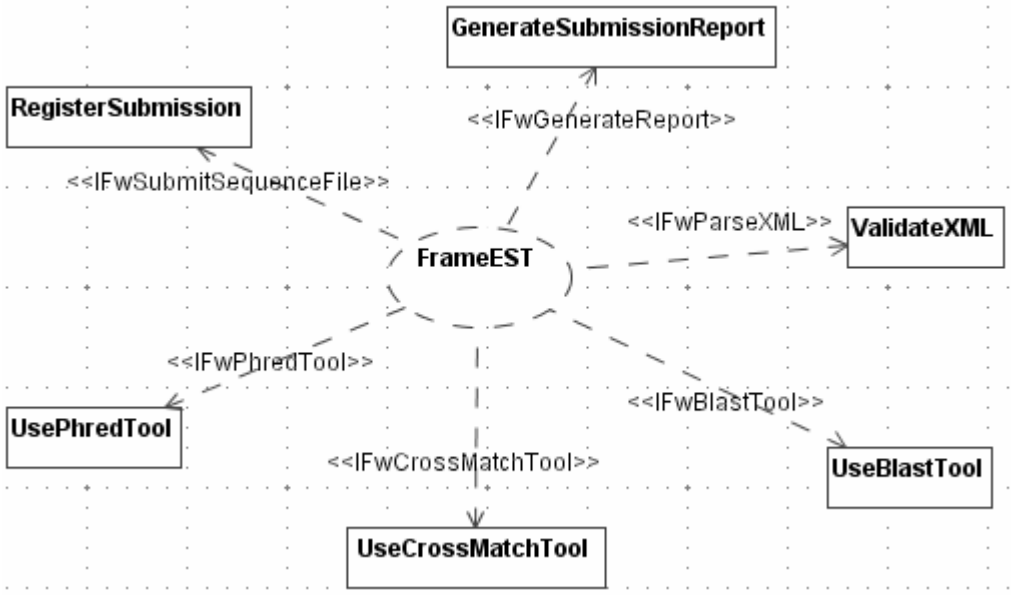

FIGURA 27 – MODELO DA APLICAÇÃO COM REUSO DO FRAMEEST

No caso, *RegisterSubmission, GenerateSubmissionReport, ValidateXml, UseBlastTool*, *UseCrossMatchTool* e *UsePhredTool* reutilizam do FrameEST, as interfaces

<IFwSubmitSequenceFile>, *<IFwGenerateReport>*, *<IFwParseXml>*, *<IFwBlastTool>*, *<IFwCrossMatchTool>* e *<IFwPhredTool>*, respectivamente.

Uma vez definidos os componentes a serem reutilizados especifica-se o modelo de componentes da aplicação, conforme mostra a Figura 28.

Para que se tenha uma idéia melhor da organização dos pacotes e componentes a Figura 28 destaca a composição interna do pacote *br.ufscar.frameest.model*, organizado em dois pacotes: *br.ufscar.frameest.model.backbean* e *br.ufscar.frameest.model.ejb*.

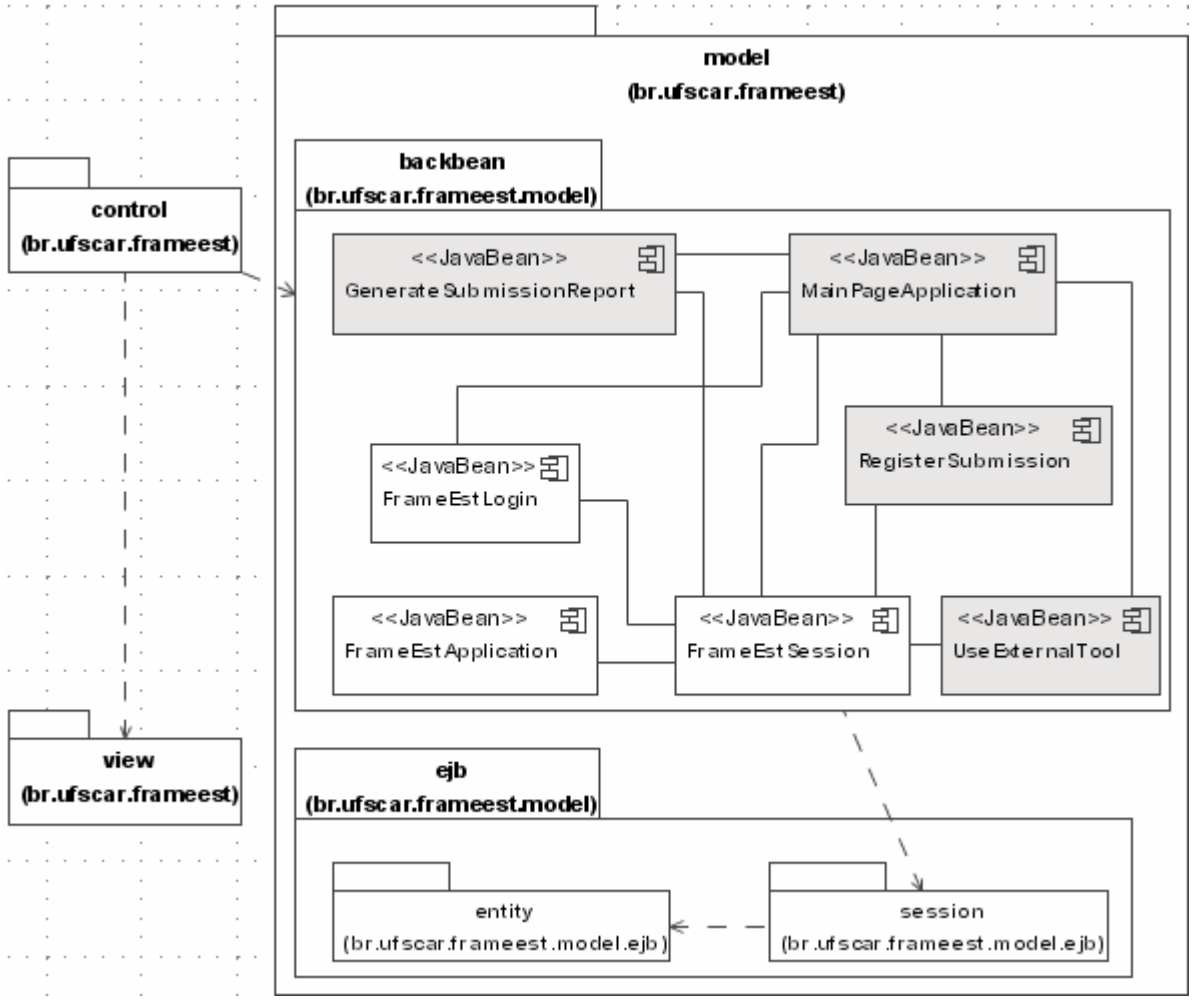

FIGURA 28 – MODELO DE COMPONENTES DA APLICAÇÃO *SHRIMP*

No pacote *br.ufscar.frameest.model.backbean* têm-se componentes reutilizados do FrameEST, e específicos da aplicação (sombreados). No caso, os componentes *FrameEstApplication, FrameEstSession, FrameEstLogin, MainPageApplication,*

GenerateSubmissionReport, *UseExternalTool* e *RegisterSubmission* utilizam serviços providos pelos componentes do pacote *br.ufscar.frameest.model.ejb.session*.

O pacote *br.ufscar.frameest.model.ejb.entity* contém componentes do modelo da aplicação que provêm serviços aos componentes do pacote *br.ufscar.frameest.model.ejb.session*.

6.2 IMPLEMENTAÇÃO

Para implementação das aplicações o FrameEST está disponível através de um *plug-in* na ferramenta *Eclipse*, conforme ilustra a Figura 29.

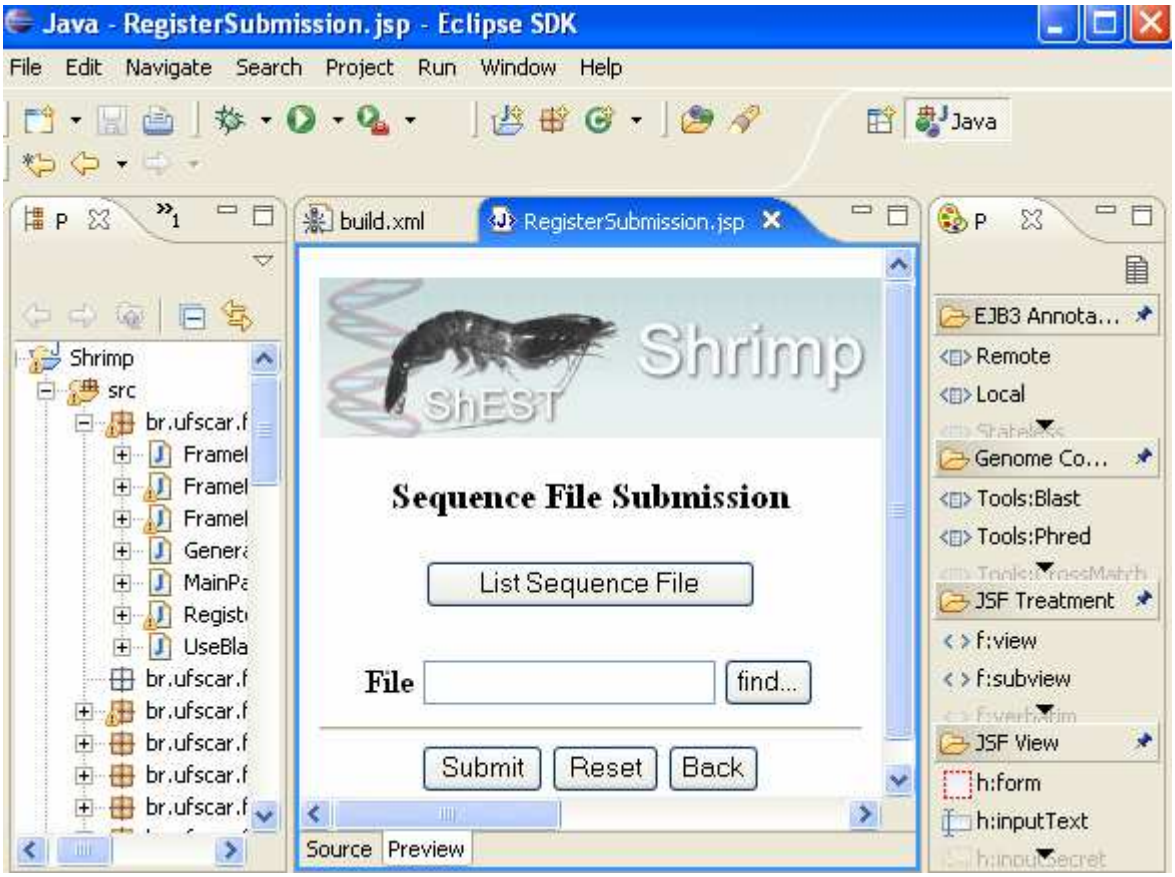

FIGURA 29 – IMPLEMENTAÇÃO DA APLICAÇÃO *SHRIMP* NO *ECLIPSE*

À esquerda da figura, no *browser*, pode-se navegar na estrutura do FrameEST. À direita têm-se as paletas com os componentes do FrameEST disponíveis para reuso da aplicação. No centro tem-se uma página *JSP* da aplicação. Essa página está associada a um

BackBean (*RegisterSubmission*), que acessa os componentes *JSF* e *EJB Session Beans*. No caso, o *BackBean* reutiliza serviços do componente *FwSubmitSequenceFileBean*.

Componentes reutilizados através de instanciações diretas têm suas implementações providas pelo FrameEST. Componentes reutilizados através de especializações têm parte de suas implementações providas pela própria aplicação. Nesta aplicação, o componente *BackBean FrameEstApplication*, e os componentes *EJB Session Beans FwGenerateReportBean, FwSubmitSequenceFileBean, FwBlastToolBean, FwPhredToolBean* e *FwCrossMatchToolBean*, foram reutilizados através de instanciações, e os componentes *FrameEstLogin* e *FrameEstSession*, através de especializações.

Outros componentes foram construídos especificamente para este estudo de caso, como os componentes *BackBeans MainPageApplication, GenerateSubmissionReport, UseExternalTool* e *RegisterSubmission*.

Uma vez implementada, pode-se executar a aplicação para verificar se a mesma atende aos requisitos especificados. Adotou-se o servidor *Jboss* neste projeto de pesquisa por suportar as tecnologias que foram utilizadas pelo FrameEST.

6.3 TESTES

Foram realizados vários testes (estruturais e funcionais) e validações para examinar o comportamento da aplicação, através de sua execução, e assegurar que a aplicação final corresponda aos requisitos especificados.

Para testar a aplicação, o FrameEST disponibiliza um arquivo chamado "*Build.xml*", conforme mostra a figura 30. Este arquivo é responsável pelo *deploy* da aplicação, isto é, pelo empacotamento da aplicação em um único arquivo com a extensão "*.ear*", que é automaticamente transferido para o servidor de aplicações, no caso o *Jboss*.

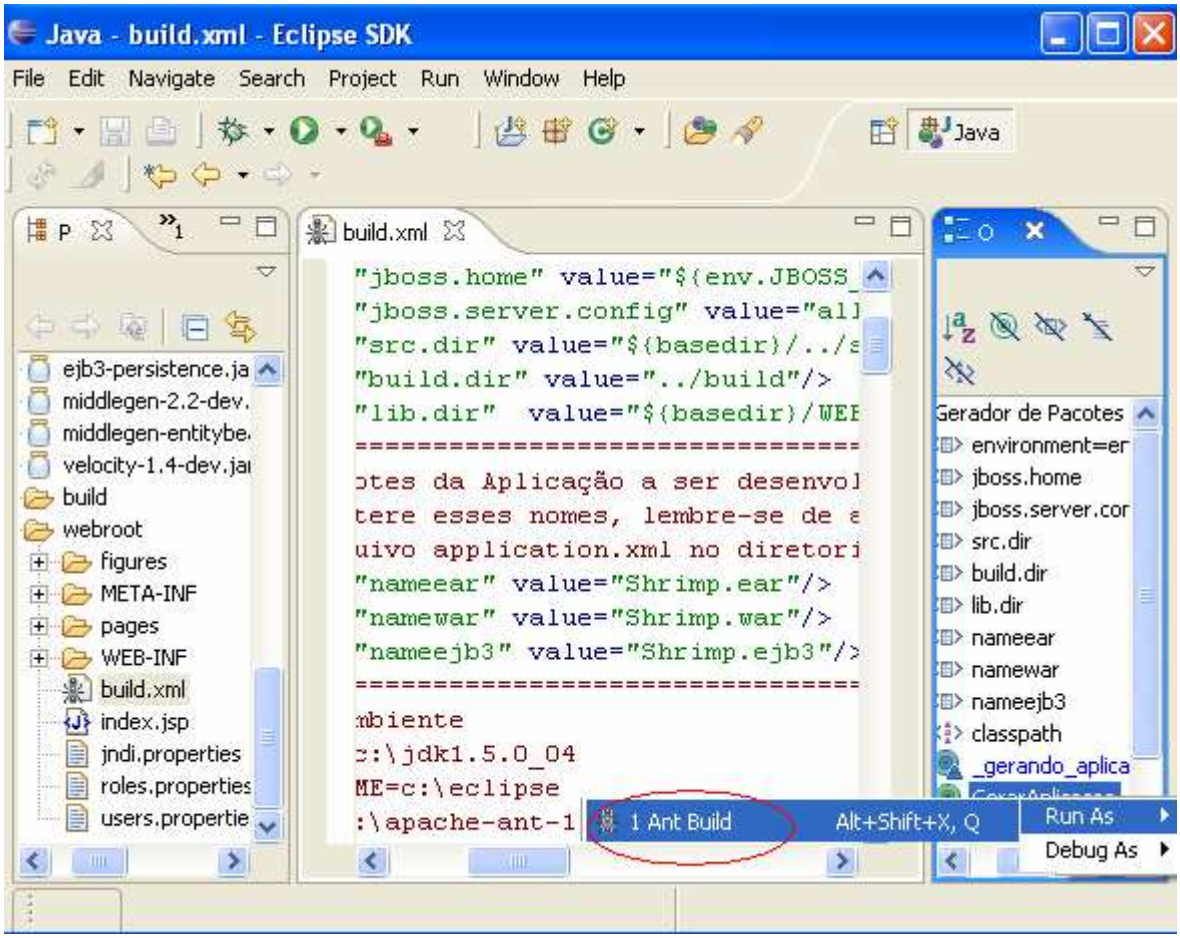

FIGURA 30 – DEPLOYING A APLICAÇÃO *SHRIMP*

Conforme mostra a figura 31, o arquivo "*.ear*", no caso "*Application.ear*", é composto por:

- Arquivo "*.ejb3*" (*Application.ejb3*): nesse arquivo são empacotados todos os componentes *ejb session* e *entity beans* do *framework* ou específicos da aplicação, bem como arquivos de configuração e segurança desses componentes, nos formatos *properties* e *xml*. O arquivo *persistence.xml* é responsável pela configuração de acesso dos *ejb entity beans.*

- Arquivo "*.war*" (*Application.war*): esse arquivo contém todo o restante da aplicação, isto é, bibliotecas *Java*, arquivos *xml* e *properties* de configuração, componentes *javabeans*, figuras e páginas *jsp* e *html*. Os arquivos *faces-config.xml* e *web.xml* são responsáveis pela configuração da aplicação como um todo, desde o gerenciamento dos componentes *javabeans*, *servlets*, páginas *jsp* até a navegação da aplicação no *browser*.

- Arquivo “*.ear*”: arquivo que empacota os dois arquivos anteriores (“*.ejb3*” e “*.war*”). O arquivo *application.xml* é responsável por configurar o acesso ao conteúdo desses arquivos.

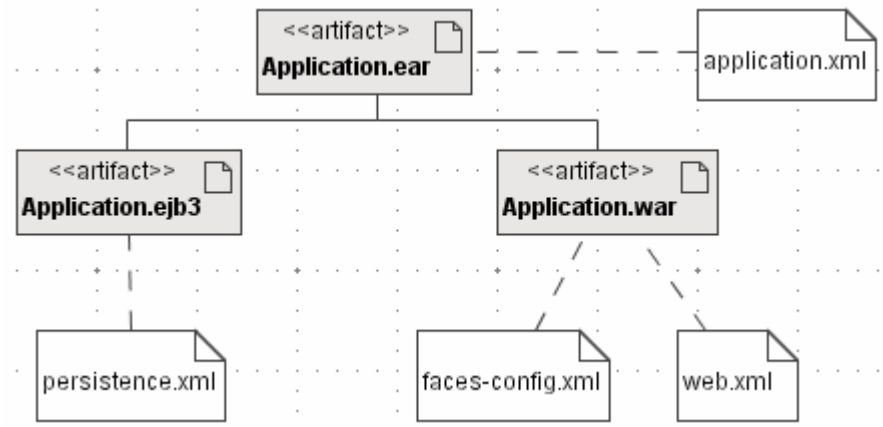

FIGURA 31 – ESTRUTURA DO ARQUIVO “*.EAR*” GERADO PELO *FRAMEWORK*

Após o procedimento de *deploy*, deve-se ativar o servidor *Jboss*, e executar um *browser*. Finalmente, indica-se a página de início da aplicação para começar a execução da aplicação. As duas próximas figuras apresentam algumas telas da execução da aplicação. A figura 32 mostra a tela inicial da aplicação, onde é solicitada a autenticação do biólogo.

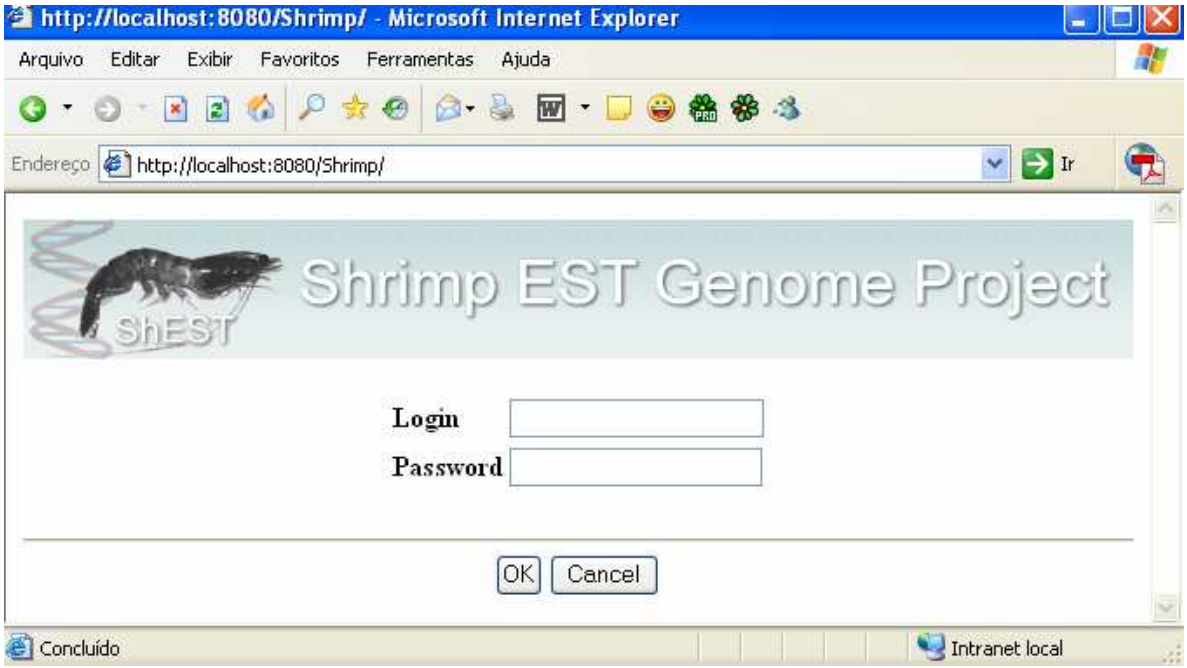

FIGURA 32 – PÁGINA INICIAL DA APLICAÇÃO *SHRIMP*

Já a figura 33 mostra uma tela na qual o biólogo pode listar seus arquivos já submetidos, como também pode submeter novos arquivos com seqüências de *DNA*.

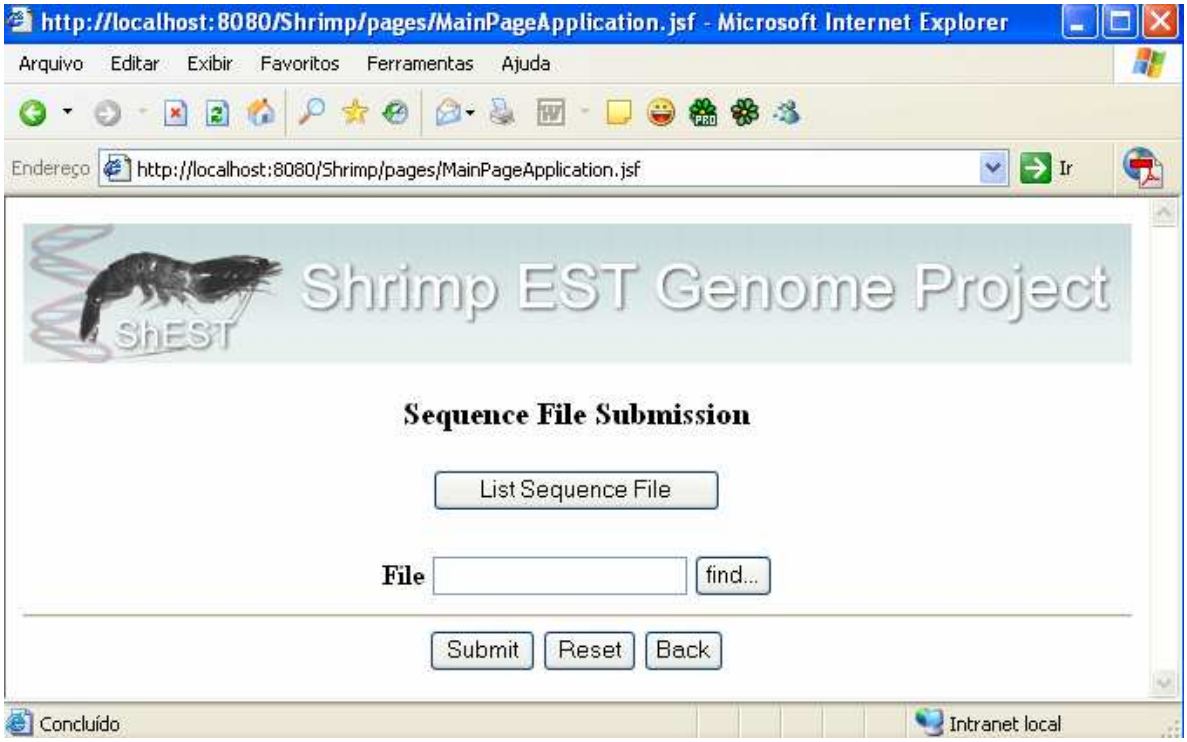

FIGURA 33 – PÁGINA DE SUBMISSÃO DE ARQUIVOS DE SEQÜÊNCIAS DE DNA

Conforme apresentado nesse estudo de caso, partiu-se dos requisitos de uma aplicação do domínio de biologia molecular, modelados conforme técnicas de *UML 2.0*, e obteve-se sua implementação, reutilizando os componentes do *framework* FrameEST, conforme os 4 passos da estratégia de desenvolvimento de aplicações.

No próximo capítulo conclui-se este trabalho, mostrando os resultados obtidos e as propostas de trabalhos futuros.

7 CONCLUSÕES

A produção de dados científicos relacionados à biologia molecular cresce em ritmo acelerado, da mesma forma que a demanda por busca, verificação, recuperação e análise destes dados. Atualmente, essa demanda não pode ser atendida satisfatoriamente pelas ferramentas e aplicativos disponíveis nesse domínio.

Diante dessas necessidades, esta dissertação apresentou um *framework* de componentes, no padrão *MVC*, para a integração de dados e ferramentas do domínio de Biologia Molecular, que vem sendo utilizado em projetos no Departamento de Computação da *UFSCar*, tais como o projeto do Camarão *Litopenaeus vannamei* (*Shrimp EST Project*). A utilização do FrameEST nesses projetos apresenta o "*feedback*" para refinar e melhorar cada vez mais sua estrutura e seus componentes.

A arquitetura desenvolvida oferece aos desenvolvedores maior rapidez, menor custo, mais segurança, flexibilidade e extensibilidade, na construção das diferentes aplicações deste domínio, possibilitando centralizar e estruturar as informações, para uso dos pesquisadores.

Acredita-se que o FrameEST dá um passo importante para melhorar o processo de desenvolvimento de *software* relacionado a pesquisas do domínio da biologia molecular.

7.1 CONTRIBUIÇÕES DO TRABALHO

Este trabalho proporcionou os seguintes resultados:

a) *framework* FrameEST para o domínio de Biologia Molecular, modelado segundo as técnicas de *Catalysis* com a tecnologia *UML 2.0* e implementado com as tecnologias *JSF* e *EJB3.0*;

b) reutilização de componentes de *software* desde a modelagem (no padrão *XMI*) até a implementação (*plug-in* para a ferramenta *Eclipse*) das aplicações;

c) flexibilidade de acesso às diversas fontes de dados heterogêneas e distribuídas, sejam arquivos texto ou bancos de dados, sendo que o acesso a este último, pode ser via *JDBC* ou *EJB Container*;

d) utilização do padrão *MVC* para simplificação e manutenção das 3 camadas: modelo, visualização e controle;

e) *framework* desenvolvido com componentes de *software*, previamente testados e com interfaces bem definidas, que facilitam o reuso;

f) proporciona um ambiente amigável de desenvolvimento das aplicações, através de *wizards* e editores visuais, facilitando a manutenção dos componentes das aplicações;

g) utilização de padrões de projeto no desenvolvimento dos componentes proporcionando desenvolvimento de aplicações cada vez mais padronizadas;

h) facilita a utilização de ferramentas disponíveis no domínio de biologia molecular, tais como *Blast*, *Phred*, dentre outras.

7.2 TRABALHOS FUTUROS

Dentre as idéias para dar continuidade a este trabalho, destacam-se:

a) ampliação das funcionalidades do *framework* FrameEST para suportar consultas multidimensionais com o uso de *Data Warehouses*;

b) gerar código *Java* automaticamente a partir da modelagem do *framework*;

c) construção de componentes para:

- habilitar o desenvolvimento de aplicações para o ambiente *desktop*;
- gerar gráficos estatísticos para os biólogos;

d) a construção de outros *frameworks* na área de biologia molecular que possam ser integrados com o *framework* FrameEST;

e) outras pesquisas que explorem o controle de qualidade e testes do FrameEST.

O FrameEST vem sendo utilizado por diferentes aplicações do projeto Genoma, como a do Camarão *Litopenaeus vannamei* (*Shrimp EST Project*). Está disponível no CD que acompanha esta dissertação e também no endereço http://www.recope.dc.ufscar.br, para toda a comunidade, para auxiliar e facilitar o desenvolvimento baseado em componentes. Sua distribuição é feita de acordo com os termos da licença *Apache License* versão 2.0, disponível em http://www.apache.org/licenses/LICENSE-2.0.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Achard, F.; Vaysseix, G.; Barillot E.; “XML, bioinformatics and data integration”, CRI Infobiogen, France and Généthon, 1 bis rue de I’Internationale, France, V.17, nº 2, p. 115-125, 2001.

[2] Alexander, C.; Ishikawa, S.; Silverstein, M.; Jacobson, M.; Fiksdahl-King, I.; Angel, S.; “A Pattern Language”, *published by Oxford University Press, New York*, 1977.

[3] Alur, D.; Crupi, J.; Malks, D.; “Core J2EE PATTERNS – Best Practice and Design Strategies”, Second Edition, United States, Sun Microsystems Press, 2003.

[4] Apache MyFaces, Disponível: http://www.myfaces.org/, acessado em Jun/2005.

[5] Braga, R.T.V.; “Um Processo para Construção e Instanciação de Frameworks baseados em uma Linguagem de Padrões para um Domínio Específico”; Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação,Universidade de São Paulo, São Carlos, SP, 2002.

[6] Brugali, D.; Sycara, K.; “Frameworks and pattern languages: an intriguing relationship”, Communication of the ACM, V. 32, nº 2, 2000.

[7] Bosch, J.; Molin, P.; Mattsson, M.; Bengtsson, P.; “Object-oriented framework-based software development: problems and experiences”, Communications of the ACM, V. 32, nº 3, 2000.

[8] Coad, Peter; “Object-Oriented Patterns”, Communications of the ACM, V. 35, nº 9, p. 152-159, 1992.

[9] Collection Programming Language - PENN Database Research Group. Disponível: http://db.cis.upenn.edu/, acessado em Jan/2005.

[10] D’Souza, D.F.; Wills, A.C.; “Objects, Components, and Frameworks with UML“, The Catalysis Book, Disponível: http://www.catalysis.org/ books/ocf/index.htm, acessado em Dez/2004.

[11] Documentação JSF on-line, Disponível : http://courses.coreservlets.com/Course-Materials/pdf/jsf/, http://www.guj.com.br/content/articles/jsf/, http://java.sun.com/j2ee/javaserverfaces/, acessado em Nov/2004.

[12] Eclipse, Available at: http://www.eclipse.org/, accessed in Jun/2005.

[13] Eclipse HTML Editor Plugin, Disponível: http://amateras.sourceforge.jp/cgi-bin/fswiki_en/wiki.cgi?page=EclipseHTMLEditor, acessado em Ago/2005.

[14] Fayad, M.E.; “Introduction to the Computing Surveys' Electronic Symposium on Object-Oriented Application Frameworks”, Communication of the ACM, V. 32, nº 1, 2000.

[15] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.; “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1995.

[16] Grand, M.; “Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML”, Volume 1, Wiley Computer Publishing, 1998.

[17] GuangChun, L.; Lu, W.; Hanhong, X.; “A Novel Web Application Frame Developed by MVC”; Information Center of UEST of China, ChengDu, China, ACM, V. 28, nº 2, 2003.

[18] Fayad, M.E.; Schmidt, D.C. (eds) Object-Oriented Application Frameworks, Communications of the ACM, V. 40, nº 10, p. 32-38, 1997.

[19] JavaBeans and Enterprise JavaBeans Technology, Disponível: http://java.sun.com/products/, acessado em Dez/2004.

[20] JBoss, Disponível: http://www.jboss.com/developers/index, acessado em Jul/2005.

[21] Johnson, R; “Components, Frameworks, Patterns”, ACM Computing Surveys, v. 30, p.10-17, 1997.

[22] Johnson, R.; “Documenting Frameworks Using Patterns“, In Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), Vancouver, British Columbia, Canada, 1992.

[23] Johnson, R.; Foote, B.; “Designing Reusable Classes”, Department of Computer Science, University of Illinois, Urbana-Champaign, Journal of Object-Oriented Programming, 1988.

[24] JSR-220 - Enterprise JavaBeans3.0. Disponível: http://www.jcp.org/en/jsr/detail?id=220, acessado em Ago/2005.

[25] JSR-250 - Common Annotations for the Java Platform. Disponível: http://jcp.org/en/jsr/detail?id=250, acessado em Jun/2005.

[26] JSR 252: JavaServer Faces, Disponível: http://www.jcp.org/en/jsr/detail?id=252, acessado em Nov/2004.

[27] KRUEGER, C.W.; “Software Reuse”, ACM Computing Surveys, v. 24, n. 2, p.131-181, 1992.

[28] Kumar, C.G.; LeDuc, R.; Gong, G.; Roinishivili, L.; Lewin, H.A.; Liu, L.; “ESTIMA, a tool for EST management in a multi-project environment”, W.M. Keck Center for Functional and Comparative Genomics, Department of Animal Sciences, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA, 2004.

[29] Kurniawan, B.; "Programando em JavaServer Faces", Editora Ciência Moderna, 2004.

[30] MagicDraw – Architetura Made Simple, Disponível: http://www.magicdraw.com/, acessado em Jun/2005.

[31] Mao, C.; Cushman, J.C.; May, G.D.; Weller, J.W.; “ESTAP, an automated system for the analysis of EST data”, Virginia Bioinformatics Institute, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA, Department of Biochemistry, University of Nevada, Reno, NV 89557, USA, Plant Biology Division, The Samuel Roberts Noble Foundation, Ardmore, OK 73402, USA and School of Computational Sciences, George Mason University, Manassas, VA 20110, USA, 2003.

[32] Middlegen, Disponível : http://boss.bekk.no/boss/middlegen/, acessado em Set/2005.

[33] Parvin, B.; Cong, G.; Fontenay, G.; Taylor, J.; Henshall, R.; Barcellos-Hoff, M.H.; “BioSig: A Bioinformatic System for Studying the Mechanism of Inter-Cell Signaling”, Proc. IEEE Int’l Symp. Bio-Informatic and Biomedical Engineering (BIBE 2000), IEEE Press, Piscataway, N.J., 2000.

[34] PostgreSQL, Disponível: http://www.postgresql.org/, acessado em Set/2005.

[35] Robertis, E.M.F.; HIB, J.; “Bases da Biologia Celular e Molecular”. Editora Guanabara Koogan, 2001.

[36] Roberts, D.; Johnson, R.; “Evolving frameworks: A pattern language for developing object oriented frameworks”, Pattern Languages of Programs Conferences, Allerton Park, Illinois, 1996.

[37] Rumbaugh, J.; “Object-Oriented Modeling and Design”, Prentice-Hall, 1991.

[38] Seibel, L.F.B.; “Bio-AXS: Uma Arquitetura para Integração de Fontes de Dados e Aplicações de Biologia Molecular”; Tese de Doutorado, Dep. de Informática, Pontifícia Universidade Católica, Rio de Janeiro, 2000.

[39] Seibel, L.F.B.; Lemos, M.; Lifschitz, S.; “Bancos de Dados de Genoma”; Simpósio Brasileiro de Banco de Dados (SBBD2000), 15./Tutoriais - SBC/UFPB, 2000.

[40] Sequence Retrieval System. Disponível: http://www.expasy.org/srs5/,acessado em Jan/2005.

[41] Silva, F.H.; “Fundamentos em Biotecnologia”, CBME (Centro de Biotecnologia Molecular Estrutural), Depto de Genética e Evolução, Laboratório de Biologia Molecular, Universidade Federal de São Carlos, São Carlos, SP, 2001.

[42] Transparent Access to Multiple Bioinformatics Information Sources. Disponível: http://imgproj.cs.man.ac.uk/tambis/, acessado em Out/2004.

[43] UML 2.0 – Unified Modeling Language, Disponível: http://www.uml.org/#UML2.0/, acessado em Mai/2005.

[44] White, J.; “Enterprise Java Bean architecture and design issues”, IEEE Computer Society, Washington, DC, USA, p. 731-732, 2001.

[45] Yassin, A.; Fayad, M. E; “Domain-Specific Application Frameworks: Frameworks Experience by Industry “, John Wiley & Sons, p. 615–632, 2000.

APÊNDICE A – IMPLEMENTANDO UMA APLICAÇÃO COM O REUSO DO FRAMEEST.

Os pré-requisitos para a instalação e utilização do FrameEST são:

- *Eclipse* SDK 3.0.x ou 3.1.x;
- *Graphical Editing Framework* (*GEF*). É um projeto Eclipse e está disponível em: http://www.eclipse.org/gef/. *GEF* é atualmente distribuído em 3 partes: *Runtime*, um *SDK*, e exemplos. É necessário instalar somente o *runtime.*
- *EclipseHTMLEditor*. Este é um projeto *SourceForge* e está disponível em: http://amateras.sourceforge.jp/cgi-bin/fswiki_en/wiki.cgi?page=EclipseHTMLEditor.

1) INSTALAÇÃO

Para instalar o *framework* FrameEST, basta copiar o arquivo *br.ufscar.frameest_1.0.0.jar* para a pasta *plugins* da instalação *Eclipse*.

2) INTRODUÇÃO

Para criar uma aplicação utilizando o FrameEST, são necessários os seguintes passos:

1) Criar um projeto Java;
2) Iniciar o FrameEST;
3) Criar as páginas de visualização da aplicação;
4) Acessar os componentes do FrameEST, conforme a necessidade;
5) Criar a navegabilidade entre as páginas da aplicação;
6) Fazer *deploy* da aplicação para o servidor *JBOSS*;

Nas próximas seções é detalhado cada passo para construir uma aplicação simples, fazendo reuso do *framework* FrameEST.

3) CRIAR UM PROJETO JAVA

Primeiramente é necessário criar um projeto *Java* simples no *Eclipse* (*File -> New -> Project -> Java Project*).

OBS: é necessário *JDK 5.0*.

Depois do projeto criado, é necessário criar uma pasta, que será a raiz da aplicação (botão direito na aplicação, *New->Folder*, como exemplo o nome será *web*).

Pronto, a aplicação está disponível para iniciar o FrameEST.

4) INICIANDO O FRAMEEST

Para iniciar o FrameEST, clicar com o botão direito na pasta criada (*web*) *New->Other->*FrameEst->Iniciar Aplicação FrameEst, clicar *Next*. Na tela que se abre, clicar em *Browser*, e selecionar a pasta raiz da aplicação (*web*). Nessa mesma tela têm-se ainda opções para configurar a aplicação. Depois de escolher as configurações, clicar no botão *Finish*.

Após iniciar o FrameEST, falta agora configurar as pastas que irão armazenar o código fonte e compilado da aplicação. Sendo assim, clicar com o botão direito novamente na aplicação e escolher o item *Properties*. Na tela que se abre, escolher o item *Java Build Path* e na paleta *Source*, clicar no botão *Add Folder*, selecionar a pasta *src* e clicar *ok*. Caso dê alguma mensagem de aviso, clique *yes*.

Para configurar a pasta para o código compilado, nessa mesma tela clicar no botão *Browser* e selecionar Aplicação->*web->WEB-INF* e clicar no botão *Create New Folder*, na caixa que se abre, digitar o nome da pasta que irá conter os códigos compilados da aplicação, que nesse exemplo será “classes”. Agora basta clicar *ok*, caso dê novamente alguma mensagem de aviso, clique *yes*.

Após essa etapa, a aplicação está pronta para começar a desenvolver as interfaces de acesso e o reuso dos componentes do FrameEST.

5) CRIAR AS PÁGINAS DE VISUALIZAÇÃO DA APLICAÇÃO

Nesse exemplo, a aplicação terá 3 páginas, *login.jsp*, *principal.jsp* e *erro.jsp*. Para criá-las, selecione a pasta *pages* que está dentro da pasta *web* da aplicação. Clique com o botão direito *New->Other->*FrameEst->Criar Visualização da aplicação FrameEst. Na tela que se abre, colocar o nome da página, por exemplo, *login.jsp* e clique em *Finish*. Faça o mesmo procedimento para as outras páginas.

As páginas serão construídas utilizando componentes *JSF*, para facilitar o reuso desses componentes, foi desenvolvido uma paleta de componentes. Para visualizá-la, faça o seguinte: no ambiente *Eclipse* clique *Window->Show View->Other->Amateras->Palette*. Será mostrada uma paleta de componentes *JSF (JSF View* e *JSF Treatment)* e *EJB3 (Genome Components* e *EJB3 Annotation).* Para a página *login.jsp* preencha de acordo com a figura 1.

```
<h:form>
 <center>
   <br/>
   <h3>Aplicação com reuso do FrameEST</h3>
   <br/>
   <h:outputText value="Login "/><h:inputText binding="#{FrameEstLogin.htmlUser}"/>
   <br/>
   <h:outputText value="Senha "/><h:inputSecret binding="#{FrameEstLogin.htmlPass}"/>
   <br/>
   <h:commandButton value="Ok" action="#{FrameEstLogin.checkLogin}"/>
   <h:commandButton value="Cancelar"/>
 </center>
</h:form>
```

FIGURA 1 - CÓDIGO DA PÁGINA LOGIN.JSP.

Os códigos para as outras páginas são:

- Página *principal.jsp*, conforme a figura 2;

```
<h:form>
 <center>
    <br/>
        <h2>Página Principal da Aplicação</h2>
    <br/>
        <h4>Seja Bem vindo à Página Principal</h4>
 </center>
</h:form>
```

FIGURA 2 - CÓDIGO DA PÁGINA PRINCIPAL.JSP.

- Página *erro.jsp*, conforme a figura 3;

```
<h:form>
  <center>
    <br/>
        <h2>Página de Erro</h2>
    <br/>
        <h3>Você teve problemas no acesso ao sistema<br/>
            Por favor, volte e verifique seu usuario e senha !!</h3>
  <h:commandButton value="Voltar" action="back"/>
  </center>
</h:form>
```

FIGURA 3 - CÓDIGO DA PÁGINA ERRO.JSP.

Será criada também uma página inicial chamada *index.jsp*, conforme mostra a figura 4. Esta página é criada a partir da pasta raiz da aplicação *(web)* e não dentro da pasta *pages.*

```
<html>
  <head>
      <meta http-equiv="Content-Type" content="text/html; c
      <title></title>
      <link href="/figures/stylesheet.css" rel="stylesheet"
  </head>
  <body style="background-color: rgb(204, 204, 204); -rave-
          <jsp:forward page="/pages/login.jsf"/>
  </body>
</html>
```

FIGURA 4 - CÓDIGO DA PÁGINA INDEX.JSP.

Notem que o *forward* vai para *login.jsf* e não *login.jsp*, isso porque "*.jsf*" está mapeado no arquivo *web.xml* da aplicação.

6) ACESSAR OS COMPONENTES DO FRAMEEST;

Notem que na figura 1 e 3, aparecem, na opção *action* dos componentes *buttons*, o nome "*FrameEstLogin.checkLogin*", fazendo o reuso do componente *FrameEstLogin*, responsável pela validação de usuário e senha. Para demonstrar flexibilidade, esse componente será customizado, que está na pasta *src-> br.ufscar.frameest.model.backbean->FrameEstLogin.java* deixando-o de acordo com a figura 5.

```
public String checkLogin(){
    String lg = new String(htmluser.getValue().toString());
    String pw = new String(htmlpass.getValue().toString());
    /*if (user.checkUser(lg,pw)){
        getFrameEstSession().setUser(lg);
        getFrameEstSession().setPasswd(pw);*/
    if (lg.equals("luiz") && pw.equals("123")) {
        return "SuccessLogin";
    }else{
        return "FaultLogin";
    }
}
```

FIGURA 5 – CUSTOMIZANDO O COMPONENTE FRAMEESTLOGIN.

Como neste exemplo ainda não se utiliza banco de dados, basta comentar a linha que acessa o *ejbsession* de usuário conforme mostra a figura 6.

```
//private IFwRegisterBiologist user;
private HtmlInputText htmluser = new HtmlInputText();
private HtmlInputText htmlpass = new HtmlInputText();

public FrameEstLogin(){
    try{
    //  javax.naming.InitialContext ctx = new javax.naming.InitialCon
    //  user = (IFwRegisterBiologist) ctx.lookup(IFwRegisterBiologist
    } catch (Exception e) {
        throw e instanceof javax.faces.FacesException ? (javax.faces.
    }
}
```

FIGURA 6 – COMENTANDO O CÓDIGO REFERENTE AO *EJBSESSION* DE USUÁRIO

7) CRIAR A NAVEGABILIDADE ENTRE AS PÁGINAS DA APLICAÇÃO

Nesse exemplo, o objetivo é o biólogo preencher usuário e senha, e caso tenha digitado corretamente, é direcionado para a página principal, caso contrário, vai para a página de erro. Essa lógica é construída através de mapeamentos *xml* que indica a navegabilidade da aplicação. O FrameEST disponibiliza um editor para tal função. Para acessá-lo, clique com o botão direito no arquivo *web->WEB-INF->faces-config.xml*, e siga o caminho *Open With->FrameEst faces-config.xml*. Será aberto um editor responsável por:

- Navegabilidade - para construir a navegabilidade das páginas da aplicação;
- *BackBeans* – Nome dos *JavaBeans* que acessam os componentes;

- Fonte *xml* – código *xml* gerado pelos 2 editores anteriores. É o conteúdo do arquivo *faces-config.xml*, que é utilizado pelo FrameEST para definir o controle dos *Javabeans* e da navegabilidade das aplicações.

A partir desse editor será construída a navegabilidade da aplicação, conforme mostra a figura 7. Nesta, a página *login.jsp* direciona o usuário para a página *principal.jsp*, caso tenha sucesso na validação do *login* (*SuccessLogin*), e para a página erro, caso tenha problemas (*FaultLogin*).

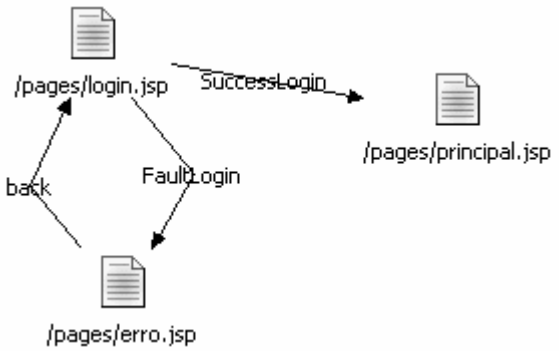

FIGURA 7 – NAVEGABILIDADE DA APLICAÇÃO

8) FAZER DEPLOY DA APLICAÇÃO PARA O SERVIDOR JBOSS;

O FrameEST faz *deploy* das aplicações para o servidor *Jboss*, versão 4.0 ou superior, que tenha o suporte para componentes *EJB* versão 3.0.

Para facilitar o processo de *deploy*, o FrameEST disponibiliza um arquivo chamado *build.xml*, que fica dentro da pasta *web*. Este arquivo é responsável por "empacotar" e configurar a aplicação que será disponibilizada para o *JBoss*. Conforme mostra a figura 8, o FrameEST empacota os arquivos da aplicação em 3 níveis. O primeiro nível empacota os componentes *EJB3* em um arquivo com a extensão "*.ejb3*", neste exemplo "*Aplicacao.ejb3*". O segundo nível é o empacotamento de todos os arquivos da aplicação, exceto os componentes *EJB3*, em um arquivo com a extensão "*.war*", neste caso "*Aplicacao.war*". E finalmente o arquivo "*Aplicacao.ear*", empacota os dois arquivos gerados anteriormente.

```
<!-- ======================================================== -->
<!-- Nome dos Pacotes da Aplicação a ser desenvolvida         -->
<!-- OBS: Caso altere esses nomes, lembre-se de alterar       -->
<!-- também o arquivo application.xml no diretorio META-INF   -->
  <property name="nameear" value="Aplicacao.ear"/>
  <property name="namewar" value="Aplicacao.war"/>
  <property name="nameejb3" value="Aplicacao.ejb3"/>
```

FIGURA 8 – PARTE DO CÓDIGO DO ARQUIVO BUILD.XML

Para alterar o nome da aplicação na operação de *deploy*, além de alterar o código do arquivo *build.*xml, conforme mostrado na figura 8, também é necessário alterar outro arquivo chamado *application.xml*, que fica em *web->META-INF*. Esse arquivo é responsável pelo mapeamento dos arquivos que são empacotados e também pela definição do nome da aplicação no *browser*. A figura 9 mostra o código do arquivo *application.xml*

```
<application>
   <display-name>Aplicacao</display-name>

   <module>
      <web>
         <web-uri>Aplicacao.war</web-uri>
         <context-root>/Aplicacao</context-root>
      </web>
   </module>
   <module>
      <ejb>Aplicacao.ejb3</ejb>
   </module>
</application>
```

FIGURA 9 – CÓDIGO DO ARQUIVO APPLICATION.XML

Para executar o *deploy* da aplicação, abra o arquivo *build.xml* com duplo clique, vá até o menu W*indow->Show View->Outline,* aparecerá uma lista de funções disponíveis nesse arquivo. Clique com o botão direito na opção *GerarAplicacao, Run As->Ant Build* . Serão mostradas as mensagens de execução de *deploy* conforme mostra a figura 10.

```
Buildfile: C:\eclipse\wkp_ap\Aplicacao\web\build.xml
GerarAplicacao:
    [mkdir] Created dir: C:\eclipse\wkp_ap\Aplicacao\build
      [jar] Building jar: C:\eclipse\wkp_ap\Aplicacao\build\AplicacaoFrameEst.ejb3
      [zip] Building zip: C:\eclipse\wkp_ap\Aplicacao\build\AplicacaoFrameEst.war
      [zip] Building zip: C:\eclipse\wkp_ap\Aplicacao\build\AplicacaoFrameEst.ear
     [copy] Copying 1 file to C:\jboss-4.0.2\server\all\deploy
     [copy] Copying 1 file to C:\jboss-4.0.2\server\all\deploy
BUILD SUCCESSFUL
Total time: 2 seconds
```

FIGURA 10 – MENSAGENS DA EXECUÇÃO DE DEPLOY

OBS: antes de executar a operação de *deploy* da aplicação, é preciso já ter instalado e configurado as variáveis de ambiente *JAVA_HOME*, *JBOSS_HOME* e *ANT_HOME*, correspondentes ao *Java JDK5.0*, servidor *JBOSS* com suporte para *EJB3* e o aplicativo *ANT*.

Para testar a aplicação, basta iniciar o servidor de aplicações, neste caso JBOSS, abrir um *browser* e digitar a *url* http://localhost:8080/Aplicacao. Caso tenha seguido todos os passos corretamente, deverá aparecer a tela de *login* da aplicação.